build 10 Flutter 3.0 Apps in 100 Days

SANJIB SINHA

Build 10 Flutter 3.0 Apps in 100 Days

A Step by Step Guide to build Apps and Master Flutter

Sanjib Sinha

This book is for sale at http://leanpub.com/flutter-artisan

This version was published on 2022-07-06

Contents

1. Getting Started with Flutter 3.0

Before getting started, let me tell you one thing. Always use the latest Provider package - https://pub.dev/packages/provider[1] for state management. And always maintain the Null Safety - https://flutter.dev/docs/null-safety[2].

I also strongly recommend to read the latest and updated articles on Flutter - https://sanjibsinha.com/category/flutter/ [3].

Download or Upgrade to the latest Flutter 3.0

To start with, we need to download the Flutter framework.

That is our first task. We need to go to The installation page of Flutter - https://flutter.dev/docs/get-started/install[4] page from where we will download and install Flutter according to your operating system.

If you have been working with Flutter 2.*, then just issue the following command:

```
flutter upgrade
```

[1]https://pub.dev/packages/provider
[2]https://flutter.dev/docs/null-safety
[3]https://sanjibsinha.com/category/flutter/
[4]https://flutter.dev/docs/get-started/install

However, Flutter gets updated and it wants to be upgraded often. For example, Text Buttons to take inputs from users are no longer the same now.

We will start with Windows, first.

Want to read more old and archived Flutter related Articles and resources?

For more old and archived Flutter related Articles and Resources[5]

Before that we want to make one thing clear.

What is new in Flutter 3.0

A few days ago Google announced the official release of Flutter 3.0. Let's see what is new in Flutter 3.

Firstly, with reference to mobile application development, there has not been a great change. Structurally what we have been doing, will continue to do.

Certainly a change in here and there had taken place. In the next section we'll discuss that and will take a deep dive.

Secondly, flutter web sections have got the makeover. Certainly it has become better.

Finally, a lot of changes have taken place in the desktop part.

Three months ago Google Flutter and Dart team announced Flutter support for Windows.

Flutter 3.0 is stable for macOS and Linux.

With reference to macOS and Linux, a lot of changes have taken place.

[5]https://sanjibsinha.com

Now it's ready for production on all desktop platforms. As a result, we can now create platform-rendered menu bars on macOS. To do that we can use the "PlatformMenubar" widget that inserts platform-only menus.

It also supports accessibility services such as screen-readers, accessible navigation, and inverted colours.

Full support for international text input on all desktop platforms is also there.

Web updates in flutter 3.0

Let's talk about the Web updates in flutter 3.0.The "ImageDecoder" API plays an important role in web applications.

Therefore flutter web now uses the "ImageDecoder" API in browsers that support it.

And there are more.

We can also use other widgets in flutter web.

For example we can think of the splash screen, loading indicator etc.

Most importantly, Flutter 3.0 has improved the performance.

It is faster than before.

As an outcome it builds frames 20 percent faster which is a significant progress.

Last but not least, we should mention two interesting things that have caught our attention.

Firstly, Flutter 3.0 supports foldable mobile devices. Secondly, Flutter 3.0 supports Material Design 3.

In addition, now we can test any Flutter app using Chrome Borwser by issuing the following command.

```
flutter run
```

From Flutter 3.0 we don't have to issue this command any more.

```
flutter run -d chrome --web-renderer html
```

Before Flutter 3.0, with only "flutter run" command we couldn't load an image from another domain.

1 How Widget tree follows the design and layout principles

After installing Dart and Flutter in our system, we are ready to go. We'll see how a beginner can use Flutter to design beautiful Application that will run in web, mobile and desktop at the same time, from a single codebase.

Not only that, we can build the same mobile application that will run in iOS and Android at the same time.

Moreover, Flutter is extremely beginner friendly software development kit, or framework. All you need to have a basic idea about object oriented programming and Dart language which acts as a tool for Flutter.

The full code repository for this section[6]

Moreover, for updated flutter tutorials don't forget to visit:

Updated Flutter Tutorials[7]

Firstly, let's check that we have the latest flutter and dart in our system.

Secondly, use your terminal and issue the following commands one after another.

[6]https://github.com/sanjibsinha/flutter_artisan/tree/layout-basic
[7]https://flutter.sanjibsinha.com

```
dart --version
...
flutter --version
```

Finally, it comes up with the following output for the Dart programming language that we need for Flutter.

```
dart --version
Dart SDK version: 2.15.0 (stable) (Fri Dec 3 14:23:23 202\
1 +0100) on "linux_x64"
```

Next, we get informed about our latest Flutter version.

```
flutter --version
Flutter 2.8.0 • channel stable • https://github.com/flutt\
er/flutter.git
Framework • revision cf44000065 (3 days ago) • 2021-12-08\
 14:06:50 -0800
Engine • revision 40a99c5951
Tools • Dart 2.15.0
```

If you're a beginner, and having little, or no knowledge of Dart programming, one question will definitely come to your mind.

Does Flutter require coding?

The answer is yes.

To start with you don't have to code a lot. But as you progress, you'll see that good Dart programming knowledge will help you to adopt Flutter fast. Moreover, you can build complex Flutter apps quite easily.

However, all said and done, let's learn to design Flutter Application step by step. Slowly, but definitely.

At the very beginning you don't have to code. We must have a good idea about what Widgets are and how do they play important role in building any Flutter Application.

Therefore, in this section we'll only learn two things to use Flutter as a beginner.

What is Widget in Flutter

How we can use Widget to build our first Flutter Application.

Let's answer the first question first.

Widget is a class that describes how the Flutter application should look like.

Consider the Center Widget. We can use Center Widget to keep its child Widget in the middle of the screen.

As we progress, we'll learn that every Widget either has a child Widget, or has multiple Widgets as its children.

That means, the parent Widget that initially comes up with a child Widget may have multiple Widgets as its children. Then each child belonging to the children Widgets may have more Widgets as its child or children. And the Widget tree gets bigger as we go down.

This Widget tree starts with a ROOT widget. As a result, the "main.dart" file is our entry point where we declare: run the application.

```
import 'package:flutter/material.dart';

void main() {
runApp(
    const OurFirstApp(),
);
}
```

OurFirstApp is the custom Widget that we'll build with other Flutter Widgets, such as Center, Container and Text.

As a result the code looks like the following.

```
class OurFirstApp extends StatelessWidget {
const OurFirstApp({Key? key}) : super(key: key);

@override
Widget build(BuildContext context) {
    return Center(
    child: Container(
        margin: const EdgeInsets.all(5),
        child: const Text(
        'Wlcome to our first flutter application',
        textDirection: TextDirection.ltr,
        ),
    ),
    );
}
}
```

Now we go through the above code line by line to understand how it works.

Firstly, the custom Widget OurFirstApp that we've declared and passed through runApp() method, is a class which extends StatelessWidget.

As we progress, we'll see what is StatelessWidget, and what is StatefulWidget. But at present, let's concentrate on the term "extends".

It means, our custom Widget OurFirstApp can use many properties and methods of a StatelessWidget that Flutter gives us to use to build our Application.

As a result, the next thing that attracts our attention is build() method that passes an object "context". Moreover, it returns a Center Widget.

From here we see that a small Widget tree has grown.

Secondly, the Center Widget has a child Widget Container which again has a child Widget Text that passes a String value through its constructor. Not only that, it has a named parameter "textDirection" that declares that the direction of Text should from left to right.

Now we can run our first Flutter Application, and see the screenshot.

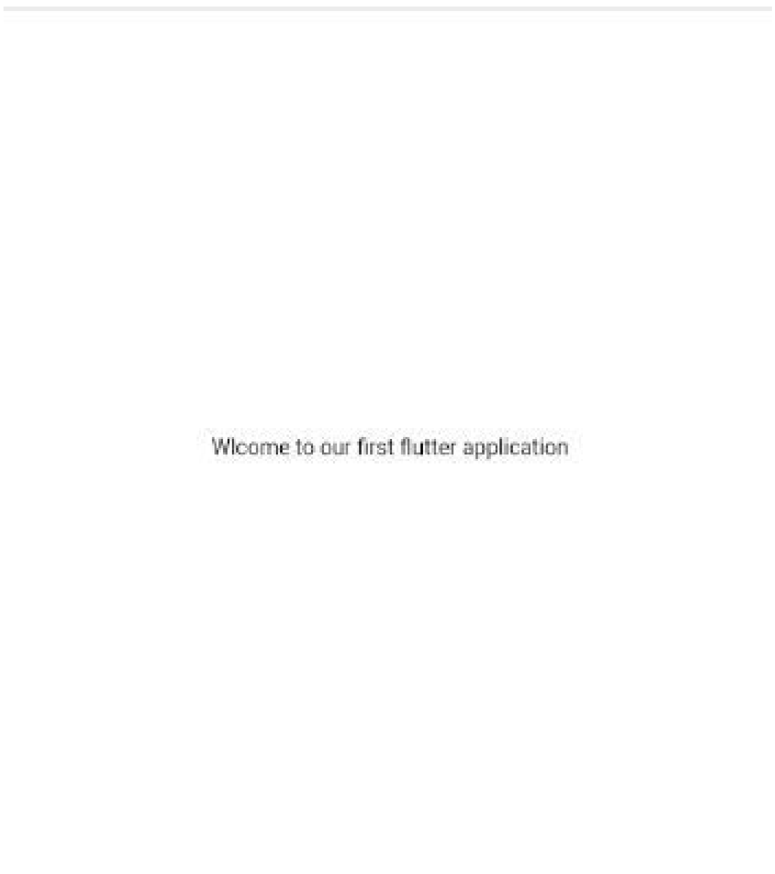

Wlcome to our first flutter application

Figure 1.1 – A Basic Non-Material Flutter App

By the way, if we're absolute beginners without having any prior programming knowledge, we've already encountered a few unknown words, such as class, constructor, method, named parameter, etc.

Therefore, it is advisable, that we should learn the basic Dart programming language and after that we start learning Flutter. As a result, we'll pick up Flutter quite easily.

In the next section, we'll learn what are basic Widgets that we always need in building a Flutter Application. Then we'll dive deep to learn and understand more complex layout and design techniques.

How to use Widget

Firstly, widget in flutter is a class that describes how our flutter app should look like by creating its instances.

Secondly, the central idea behind using widgets is to build our user interface using widgets. To clarify, widgets are like boxes on the mobile, tab or desktop screen. Consequently, since each box has a size, every widget has constraints that deal with width and height.

Therefore, finally, we can define a widget as a class that builds or rebuilds its description; and, our flutter app works on that principle.

For a beginner, we need to add something more with this definition.

In Flutter everything is widget. As a result, we must think widget as a central hierarchy in a Flutter framework.

Let us see a minimalist Flutter App to understand this concept first.

```
import 'package:flutter/material.dart';
import 'widgets/first_flutter_app.dart';

void main() {
runApp(
    const FirstFlutterApp(),
);
}
```

The above code shows us that the runApp() function takes the given Widget FirstFlutterApp() and makes it the root of the widget tree.

And this is our first custom widget that will sit on the top of the tree.

With reference to the main() function we must also add that to work it properly we need to import Material App library. However, we don't want to dig deep now. Just to make it simple, let's know that we need to have a material design so we can show our widget boxes. Right?

Further, we have created a sub-directory called "widgets" in our "lib" directory and keep the code of FirstFlutterApp(). Remember that also represents a class of hierarchy. Therefore we need to import that local library too.

That is why we need to import that also.

```
import 'widgets/first_flutter_app.dart';
```

Now, we can take a look at the custom widget that we've built.

```
import 'package:flutter/material.dart';

class FirstFlutterApp extends StatelessWidget {
const FirstFlutterApp({
    Key? key,
}) : super(key: key);

@override
Widget build(BuildContext context) {
    return const MaterialApp(
    home: Center(
        child: Text(
        'Hello, Flutter!',
        ),
    ),
    );
}
}
```

The FirstFlutterApp() extends Stateless widget. Widgets have state. But, don’t worry, we’ll discuss it later.

The widget tree consists of three more widgets. The MaterialApp, Center widget and its child, the Text widget.

As a consequence, the framework forces the root widget to cover the screen. It places the text “Hello, Flutter” at the Center of the screen.

Since we have used MaterialApp, we don’t have to worry about the text direction. The MaterialApp will take care of that.

Since each widget is a Dart class, we need to instantiate each widget and want them to show up at the particular location. This is done through the Element class. This is nothing but an instantiation of a Widget at a particular location in the tree.

What do we see?

A text “Hello, Flutter”.

However, it is displayed on the particular position in the widget tree. Now, this widget tree can be a complex one.

As a result, widgets can be inflated into more elements as the tree grows in size.

Before we close down, one thing to remember. A widget is an immutable description of part of a user interface.

We’ll discuss that later when we will discuss Element class in detail.

Difference between Non-Material and Material Widgets

Non material widgets cannot adopt the material design and layout principles.

Certainly there are non material widgets in Flutter, and we, knowingly or unknowingly use them. However, we cannot imagine Flutter without Material design.

Whether we want to navigate to another screen, or we want to use a particular theme, or we want to add localization support, we cannot do it without the assistance of Material App widget. It includes many material design specific features that we’ll learn as we progress.

Firstly, MaterialApp is a widget that introduces many captivating tools and fascinating features. As we’ve just reminded, there are Navigator or Theme to help you develop our Flutter Application.

There seems to be some confusions about MaterialApp and Material class in Flutter. However, they are not same. Although they are linked.

Let us build a simple Flutter Application where we use some simple basic widgets and we’ll follow the Material design. As a result, we’ll place MaterialApp near root widget.

After watching the code, we’ll discuss Material principles in great detail.

```
import ‘package:flutter/material.dart’;

void main() { runApp( const OurSecondApp(), ); }

class OurSecondApp extends StatelessWidget { const OurSecondApp({Key? key}) : super(key: key);
```

```
@override
Widget build(BuildContext context) {
  return const MaterialApp(
    title: 'First Material App',
    debugShowCheckedModeBanner: false,
    home: OurSecondAppHome(),
  );
}
```

}

class OurSecondAppHome extends StatelessWidget { const OurSecondAppHome({Key? key}) : super(key: key);

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('First Material App'),
    ),
    body: const OurSecondAppBody(),
  );
}
```

}

class OurSecondAppBody extends StatelessWidget { const OurSecondAppBody({Key? key}) : super(key: key);

```
@override
Widget build(BuildContext context) {
  /// Here we're going to use basic widgets
  /// such as, Text, Container, Row, Column, etc.
  ///
  return Column(
    children: [
      Center(
        child: Row(
          children: [
            Container(
              padding: const EdgeInsets.all(10),
              child: const Text(
                'ID',
                textAlign: TextAlign.left,
              ),
            ),
            Container(
              padding: const EdgeInsets.all(10),
              child: const Text(
                'Name',
                textAlign: TextAlign.left,
              ),
            ),
            const Expanded(
              child: Text(
                'Phone Number',
                textAlign: TextAlign.center,
              ),
            ),
            Container(
              padding: const EdgeInsets.all(10),
              child: const Text(
                'Gender',
                textAlign: TextAlign.left,
              ),
            ),
            Container(
              padding: const EdgeInsets.all(10),
              child: const Text(
                'Country',
                textAlign: TextAlign.left,
              ),
            ),
          ],
        ),
      ),
      Center(
        child: Row(
          children: [
            Container(
              padding: const EdgeInsets.all(10),
              child: const Text(
                '1',
                textAlign: TextAlign.left,
              ),
            ),
            Container(
              padding: const EdgeInsets.all(10),
              child: const Text(
                'John',
                textAlign: TextAlign.left,
              ),
            ),
            const Expanded(
              child: Text(
                '123645',
                textAlign: TextAlign.center,
              ),
            ),
            Container(
              padding: const EdgeInsets.all(10),
              child: const Text(
                'Male',
                textAlign: TextAlign.left,
              ),
            ),
            Container(
              padding: const EdgeInsets.all(10),
              child: const Text(
                'Germany',
                textAlign: TextAlign.left,
              ),
            ),
          ],
        ),
      ),
      Center(
        child: Row(
          children: [
            Container(
              padding: const EdgeInsets.all(10),
              child: const Text(
                '2',
                textAlign: TextAlign.left,
              ),
            ),
            Container(
              padding: const EdgeInsets.all(10),
              child: const Text(
                'Jenifer',
                textAlign: TextAlign.left,
              ),
            ),
            const Expanded(
              child: Text(
                '652341',
                textAlign: TextAlign.center,
              ),
            ),
            Container(
              padding: const EdgeInsets.all(10),
              child: const Text(
                'Female',
                textAlign: TextAlign.left,
              ),
            ),
            Container(
              padding: const EdgeInsets.all(10),
              child: const Text(
                'France',
                textAlign: TextAlign.left,
              ),
            ),
          ],
        ),
      ),
    ],
  );
}
```

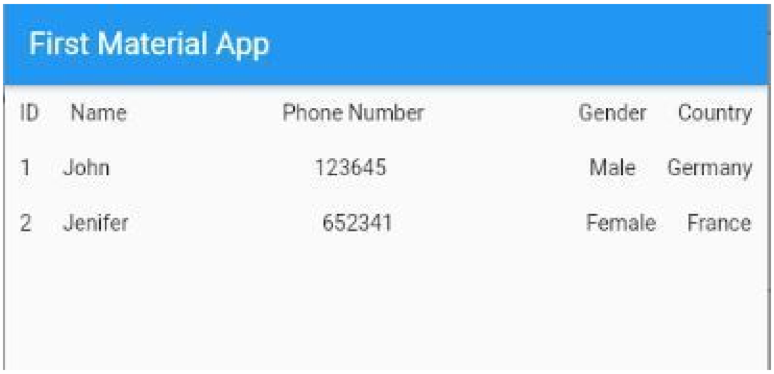
}

Quite a long code where we have used some basic widgets like Scaffold, Column, Row, Container, Expanded, Text, etc.

Moreover, we've placed the MaterialApp Widget near root widget.

return const MaterialApp(title: 'First Material App', debugShowCheckedModeBanner: false, home: OurSecondAp-pHome(),);

Therefore, we get this screenshot.



First Material App

ID	Name	Phone Number	Gender	Country
1	John	123645	Male	Germany
2	Jenifer	652341	Female	France

Figure 1.2 – Basic widgets in Flutter

Why do we use MaterialApp in Flutter?

There are many reasons to use MaterialApp. But one of the most important reasons is MaterialApp gives constraints to the child widget to fit into the screen.

Now we can refactor the above code as there are many use cases, such as we've used Container widget many times.

As a consequence, we can refactor a model Container class.

import 'package:flutter/material.dart';

void main() { runApp(const OurThirdApp(),); }

class OurThirdApp extends StatelessWidget { const OurThirdApp({Key? key}) : super(key: key);

```
@override
Widget build(BuildContext context) {
  return const MaterialApp(
    title: 'First Material App',
    debugShowCheckedModeBanner: false,
    home: OurThirdAppHome(),
  );
}
```

}

class OurThirdAppHome extends StatelessWidget { const OurThirdAppHome({Key? key}) : super(key: key);

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('First Material App'),
    ),
    body: const OurThirdAppBody(),
  );
}
```

}

class OurThirdAppBody extends StatelessWidget { const OurThirdAppBody({Key? key}) : super(key: key);

```
@override
Widget build(BuildContext context) {
  /// Here we're going to use basic widgets
  /// such as, Text, Container, Row, Column, etc.
  ///
  return Column(
    children: const [
      ColumnOne(),
      ColumnThree(),
      ColumnTwo(),
    ],
  );
}
```

}

class ModelContainer extends StatelessWidget { const ModelContainer({ Key? key, required this.modelText, }) : super(key: key);

```
final Text modelText;

@override
Widget build(BuildContext context) {
  return Container(
    padding: const EdgeInsets.all(10),
    color: Colors.amber,
    child: modelText,
  );
}
```

}

class ColumnOne extends StatelessWidget { const ColumnOne({ Key? key, }) : super(key: key);

```
@override
Widget build(BuildContext context) {
  return Center(
    child: Row(
      children: const [
        ModelContainer(
          modelText: Text('ID'),
        ),
        ModelContainer(
          modelText: Text('Name'),
        ),
        ModelContainer(
          modelText: Text('Phone'),
        ),
        ModelContainer(
          modelText: Text('Gender'),
        ),
        ModelContainer(
          modelText: Text('Country'),
        ),
      ],
    ),
  );
}
```

}

class ColumnTwo extends StatelessWidget { const ColumnTwo({ Key? key, }) : super(key: key);

```
@override
Widget build(BuildContext context) {
  return Center(
    child: Row(
      children: const [
        ModelContainer(
          modelText: Text('2'),
        ),
        ModelContainer(
          modelText: Text('Juliet'),
        ),
        ModelContainer(
          modelText: Text('100023'),
        ),
        ModelContainer(
          modelText: Text('Female'),
        ),
        ModelContainer(
          modelText: Text('Britain'),
        ),
      ],
    ),
  );
}
```

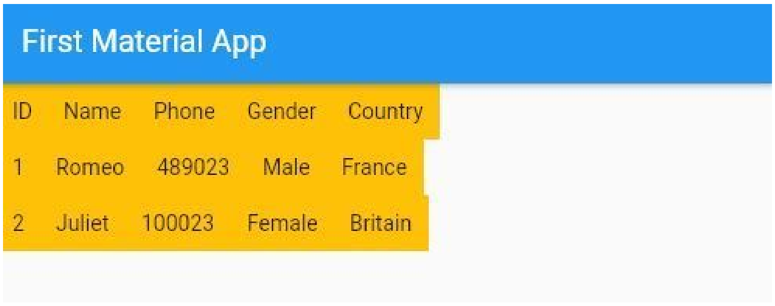
}

class ColumnThree extends StatelessWidget { const ColumnThree({ Key? key, }) : super(key: key);

```dart
@override
Widget build(BuildContext context) {
  return Center(
    child: Row(
      children: const [
        ModelContainer(
          modelText: Text('1'),
        ),
        ModelContainer(
          modelText: Text('Romeo'),
        ),
        ModelContainer(
          modelText: Text('489023'),
        ),
        ModelContainer(
          modelText: Text('Male'),
        ),
        ModelContainer(
          modelText: Text('France'),
        ),
      ],
    ),
  );
}
```

}

Now, as a result, we can design the model Container widget and that will change the look of the whole app.


First Material App

ID	Name	Phone	Gender	Country
1	Romeo	489023	Male	France
2	Juliet	100023	Female	Britain

Figure 1.3 – Material App second sample

To put it simply, each widget now knows how to behave and the instruction comes from the MaterialApp.

As a result, we don't have to bother about the text direction property of any Text Widget anymore. MaterialApp handles that through material design principles.

What is the difference between material and MaterialApp in flutter?

Actually, there is more friendship than difference between these two Widgets. One needs another. And we should use them both.

Many widgets, such as Scaffold, AppBar, Card, Dialog, Floating-Button, and many more, which are Material instances. Through Material guidelines they define User Interface elements that respect Material rules.

If we don't define Material guidelines near any Text widget, the screen will be black and the text will have a yellow underline. It takes as a fallback theme.

That is the reason, why we use MaterialApp near the root of our Flutter Application.

To sum up, we need to use Material and MaterialApp widgets both. As far as Flutter design and layout are concerned, we need to use them both. If we want that a Text widget should adopt a certain theme, we need to introduce a Material instance like Scaffold widget first.

As we progress, we'll learn more about these features, in great detail.

What are constraints in flutter

When you plan to learn Flutter, it starts with Layout. Right? Now, you cannot learn or understand flutter layout without understanding constraints. Therefore, our flutter learning starts by answering this question first – what are constraints in flutter?

Firstly, let me warn you at the very beginning. Flutter layout is not like HTML layout.

Secondly, if you come from HTML or web development background, don't try to apply those CSS rules here. Why so? Because, HTML targets a large screen. Whereas, we're dealing with a Mobile screen. So, layout should not be same. And, there are other reasons too.

Finally, flutter is all about widgets. As a result, we need to understand Flutter layout keeping widgets in our mind.

Let's start with a Material App, and, start building a simple Container widget with a Text widget as its child.

```
import 'package:flutter/material.dart';

class ConstraintSample extends StatelessWidget {
const ConstraintSample({Key? key}) : super(key: key);

@override
Widget build(BuildContext context) {
    return const MaterialApp(
    title: 'Constraint Sample',
    debugShowCheckedModeBanner: false,
    home: ConstraintSampleHomme(),
    );
}
}
@override
Widget build(BuildContext context) {
    return Container(
    width: 150,
    height: 200,
    child: const Text('Constraint Sample'),
    );
}
```

Let’s run this simple flutter app, and see what happens.

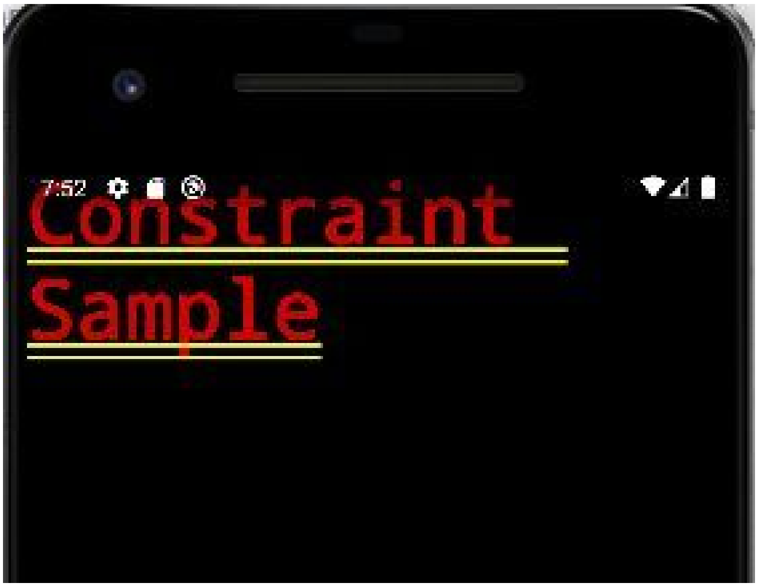


Figure 1.4 – Constraint in flutter example one

What is the problem here?

We've returned a Container widget mentioning its width and height.

```
return Container(
    width: 150,
    height: 200,
    child: const Text('Constraint Sample'),
    );
```

However, that didn't work at all.

Now we know that the constraint in Flutter is all about the size of the box, which is nothing but a widget.

A size always deals with width and height.

In the above case, the material app takes the width and height of the whole screen and directs its immediate child Container to take that size.

That is why, although we've mentioned the size of the Container widget, it doesn't work.

Therefore, we can conclude that any Widget gets its constraint or size from its immediate parent. After that, it passes that constraint to its immediate child.

And this practice goes on as the number of Widgets increases in the widget tree.

In Flutter, a parent widget always controls the immediate child's size. However, when the parent becomes grand-parent, it cannot affect the constraint or size of the grand-child.

Why?

Because, the grand child has its parent that inherits the size from its parent and decides what should be the size of its child.

We can compare this mechanism with wealth. If a person inherits some wealth from her parent, she is in a position to decide how much of that wealth she will give to her child or children.

And this process goes on.

One after another widget tells its children what their constraints or sizes are.

How do you use constraints in flutter?

Since we have got the idea, now we can apply and see how we can use constraints in flutter.

Our next code snippet is like the following.

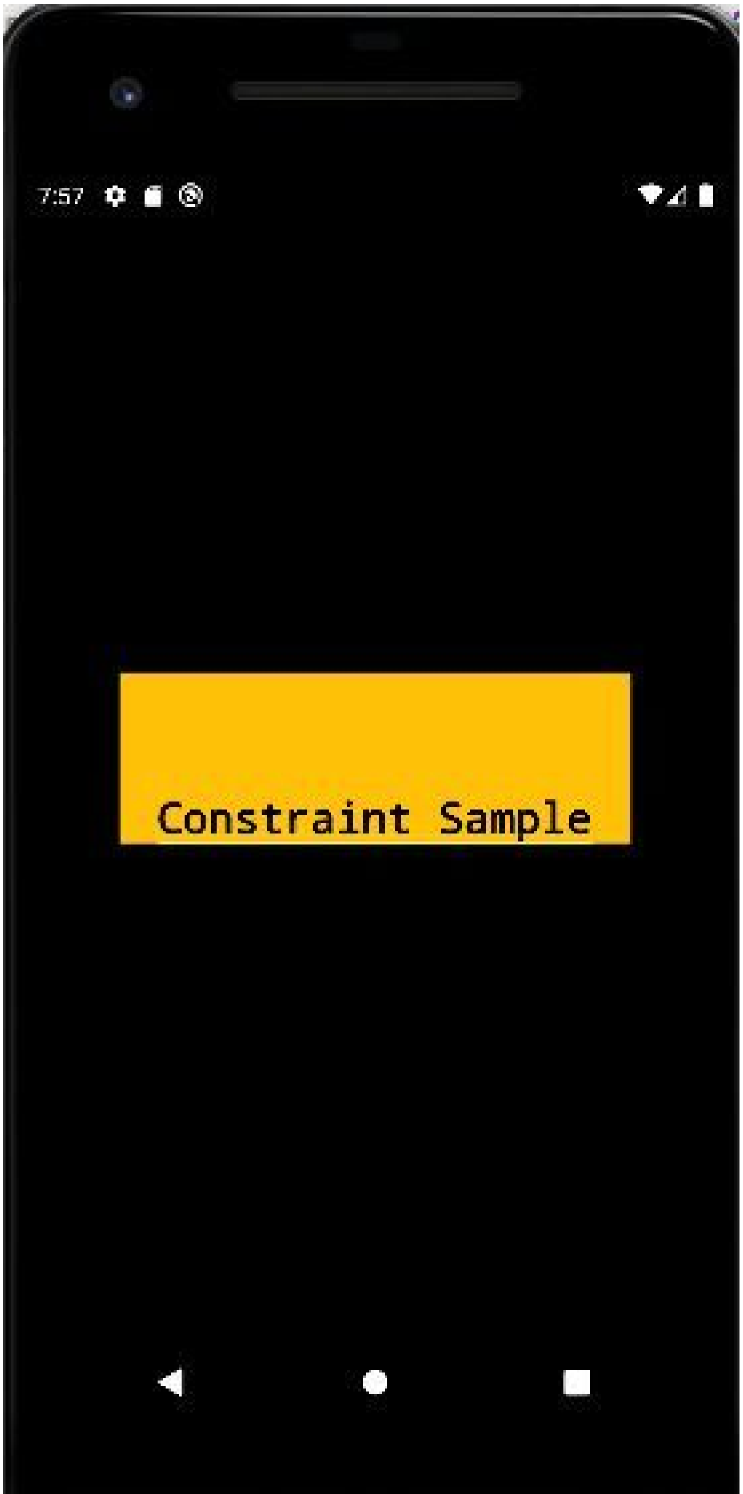
```
@override
Widget build(BuildContext context) {
    return Center(
    child: Container(
        width: 300,
        height: 100,
        color: Colors.amber,
        alignment: Alignment.bottomCenter,
        child: const Text(
        'Constraint Sample',
        style: TextStyle(
            fontSize: 25,
            fontWeight: FontWeight.bold,
            color: Colors.black,
        ),
        ),

    ),
    );
}
```

Now, we've wrapped the Container widget with a Center widget. As a result, the Center widget gets the full size now. And, after that, it asks immediate child Container widget how big it wants to be.

The Container said, "I want to be 300 in width and 100 in height. Not only that, I want to place my child at the bottom Center position. And, I also want my child should be of color amber."

The Center widget says, "Okay. No problem. Get what you want because I have inherited the whole screen-size from my parent. Take what you need."

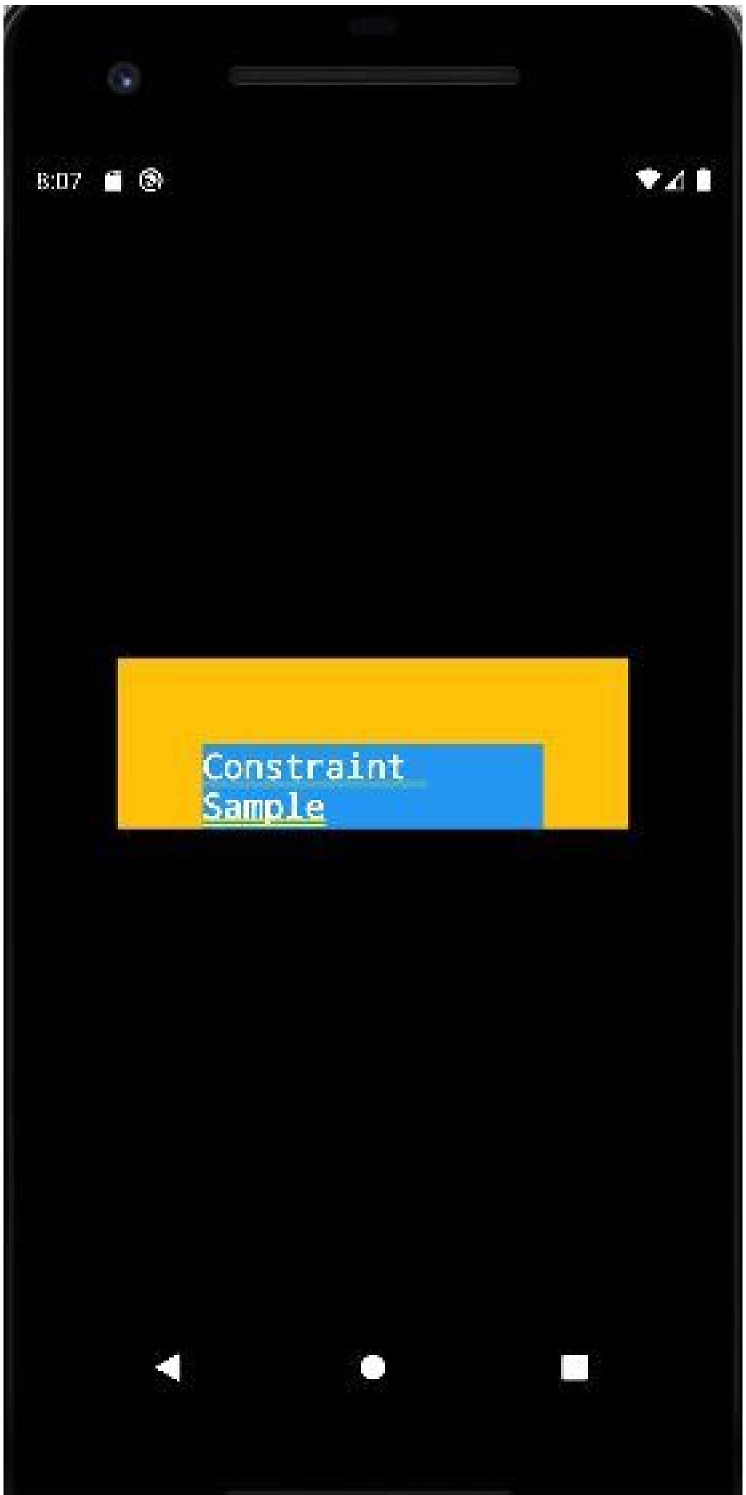
As a result, we see this on the screen.


7:57
Constraint Sample

Next, we change our code to this:

```
@override
Widget build(BuildContext context) {
    return Center(
    child: Container(
        width: 300,
        height: 100,
        color: Colors.amber,
        alignment: Alignment.bottomCenter,
        child: Container(
        width: 200,
        height: 50,
        color: Colors.blue,
        alignment: Alignment.center,
        child: const Text(
            'Constraint Sample',
            style: TextStyle(
            fontSize: 20,
            fontWeight: FontWeight.bold,
            color: Colors.white,
            ),
        ),
        ),
    ),
    );
}
```

Let's see the effect on the screen first. Then we'll discuss the code.


8:07
Constraint
Sample

The above image is displayed because the code says that the child Container with width 300, height 100 and color amber has a child, which is another Container and that has color blue, width 200 and height 50. However, the parent Container decides that it will show its child in the bottom Center position.

Align the child at the bottom Center means, we would pass this child Container a tight constraint that is bigger than the child's natural size, with an alignment of Alignment.bottomCenter.

As a result, the child Container gets the width 200 and height 50 and is placed at the bottom Center alignment. It happens smoothly because the parent Container's width and height is bigger than the child Container. Therefore, it allocates the exact size that it's asked for.

But it didn't happen, if the parent Container didn't set the alignment and said that, "Okay, child container, you can place yourself anywhere you like. Even you may decide your alignment."

So the code is like the following:

```
@override
Widget build(BuildContext context) {
    return Center(
    child: Container(
        width: 300,
        height: 100,
        color: Colors.amber,
        child: Container(
        width: 200,
        height: 50,
        color: Colors.blue,
        alignment: Alignment.center,
        child: const Text(
            'Constraint Sample',
            style: TextStyle(
            fontSize: 20,
            fontWeight: FontWeight.bold,
            color: Colors.white,
            ),
        ),
        ),
    ),
    );
}
```

As a result, the child Container decides to looks upward and tries to find if its grand-parent has any alignment already allocated for it.

While looking upward, it finds that the grand-parent is a Center widget itself. Therefore, it decides to take the Center position, as it has the Center alignment itself; and not only that, while doing so, it ignores its own width and height, and it takes the width and height of the immediate parent, which eventually is another Container.

As a result it overlaps completely the parent Container.

To sum up, constraints are basically sizes of width and height that any Widget gets from its parent. However, in case, padding is added, the constraint may change. Although we have not added that feature, still you may test that on your own and see how it affects the sizes of the child.

What are BoxConstraints in Flutter

Imagine each Widget as a box. So it has a size or constraints that defines width and height.

In the previous section we have discussed what constraints are. In Flutter, every widget is rendered by their underlying RenderBox objects. As a result, for each boxes constraints are BoxConstraints.

For a Flutter beginner we need to say one thing at the very beginning. The constraints actually represent sizes of the rendered boxes, which are nothing but widgets.

In that light of the previous discussion, we must try to understand what BoxConstraints are.

We've already seen that widgets pass their constraints, which consist of minimum and maximum width and height, to their children. Moreover, each child may vary in size.

As a result we can say that render tree actually passes a concrete geometry, which is size.

Subsequently the widget tree grows in sizes and for each boxes the constraints are BoxConstraints. And, it consists of four numbers – a minimum width minWidth, a maximum width maxWidth, a minimum height minHeight, and a maximum height maxHeight. Therefore we can set a range of width and height.

As we've said before, the geometry of boxes consists of a Size. Consequently, the Size satisfies the constraints.

Each child in the rendered widget tree, gets BoxConstraints from its parent. After that, the child picks us the size that satisfies the BoxConstraints adjusting with the parent's size.

Certainly, with the size, position changes. A child does not know its position.

Why?

Because, if the parent adds some padding the child's position changes with it.

We'll see that in a minute.

Consider a Flutter app where Scaffold widget acts as the immediate child of Material App widget.

As a result, the Scaffold takes the entire screen from its immediate parent Material App.

Next, the Scaffold passes its constraints to its immediate child Center widget. And then the Center takes the constraints or size from Scaffold. However, as the Scaffold widget allocates some space for the App Bar widget, therefore, Center doesn't get the whole screen.

Let us see the code.

import 'package:flutter/material.dart';

class BoxConstraintsSample extends StatelessWidget { const BoxConstraintsSample({Key? key}) : super(key: key);

```
@override
Widget build(BuildContext context) {
  return const MaterialApp(
    title: 'BoxConstraints Sample',
    debugShowCheckedModeBanner: false,
    home: BoxConstraintsSampleHomme(),
  );
}
```

}

class BoxConstraintsSampleHomme extends StatelessWidget { const BoxConstraintsSampleHomme({Key? key}) : super(key: key);

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('BoxConstraints Sample'),
    ),
    body: Center(
      child: Container(
        color: Colors.redAccent,
        padding: const EdgeInsets.all(
          20,
        ),
        child: const Text(
          'Box',
          style: TextStyle(
            fontFamily: 'Allison',
            color: Colors.black38,
            fontSize: 60,
            fontWeight: FontWeight.bold,
          ),
        ),
        constraints: const BoxConstraints(
          minHeight: 70,
          minWidth: 70,
          maxHeight: 200,
          maxWidth: 200,
        ),
      ),
    ),//Center
  );
}
```

}

If we run the code, we see the following screenshot.

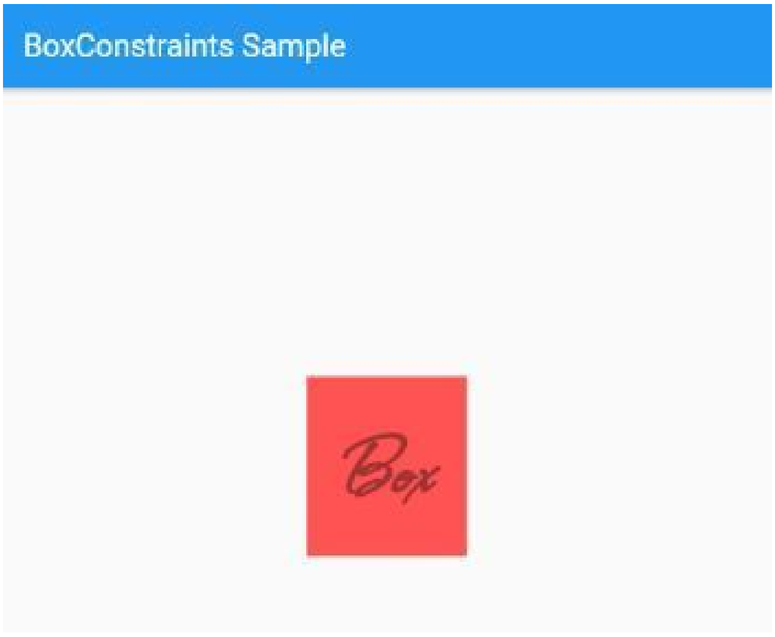


Figure 1.7 – BoxConstraints flutter example one

The above code tells us about the Container's constraints that point to BoxConstraints, this way.

constraints: const BoxConstraints(minHeight: 70, minWidth: 70, maxHeight: 200, maxWidth: 200,),

The Container widget has a constraints parameter that points to the BoxConstraints widget, which simply defines the minimum and maximum width, height. And the range is between 70px to 200px.

As a result, if we try to make the Text widget bigger, that will not display the whole text, in that case.

Why?

Because, Container passes its constraints to its child Text widget and it gets that exact value. That means, the size of Text widget must remain in between 70px to 200px.

What happens if we try to pass the same constraints to another Container, which will act as the immediate child.

Let's change our code.

How do you use box constraints in Flutter?

To use box constraints in Flutter, we must understand how BoxConstraints widget acts, maintaining the range of width and height.

Let's change our above code and it will look like the following, now.

import 'package:flutter/material.dart';

class BoxConstraintsSample extends StatelessWidget { const BoxConstraintsSample({Key? key}) : super(key: key);

```
@override
Widget build(BuildContext context) {
  return const MaterialApp(
    title: 'BoxConstraints Sample',
    debugShowCheckedModeBanner: false,
    home: BoxConstraintsSampleHomme(),
  );
}
```

}

class BoxConstraintsSampleHomme extends StatelessWidget { const BoxConstraintsSampleHomme({Key? key}) : super(key: key);

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('BoxConstraints Sample'),
    ),
    body: Center(
      child: Container(
        color: Colors.redAccent,
        padding: const EdgeInsets.all(
          20,
        ),
        constraints: const BoxConstraints(
          minHeight: 170,
          minWidth: 170,
          maxHeight: 400,
          maxWidth: 400,
        ),
        child: Container(
          color: Colors.blueAccent[200],
          padding: const EdgeInsets.all(
            20,
          ),
          child: const Text(
            'Box',
            style: TextStyle(
              fontFamily: 'Allison',
              color: Colors.white,
              fontSize: 60,
              fontWeight: FontWeight.bold,
            ),
          ),
          constraints: const BoxConstraints.expand(
            height: 100,
            width: 100,
          ),
        ),
      ), //container
    ), //Center
  );
}
```

}

In the above code, we find two Container widgets. Both have constraints defined. The first Container have constraints like the following:

constraints: const BoxConstraints(minHeight: 170, minWidth: 170, maxHeight: 400, maxWidth: 400,),

And its child, the second Container has constraints like the following:

constraints: const BoxConstraints.expand(height: 100, width: 100,),

The first Container's constraints define a range of width and height. However,the second Container's constraints point to BoxConstraints constructor BoxConstraints.expand.

What does this mean, as long as the size of the child Container is concerned?

We can explain it this way.

Since the child Container receives its constraints from its parent Container. Within that range it can expand its width and height up to 100px. Not more than that.

Moreover, this expansion process starts from the Center.

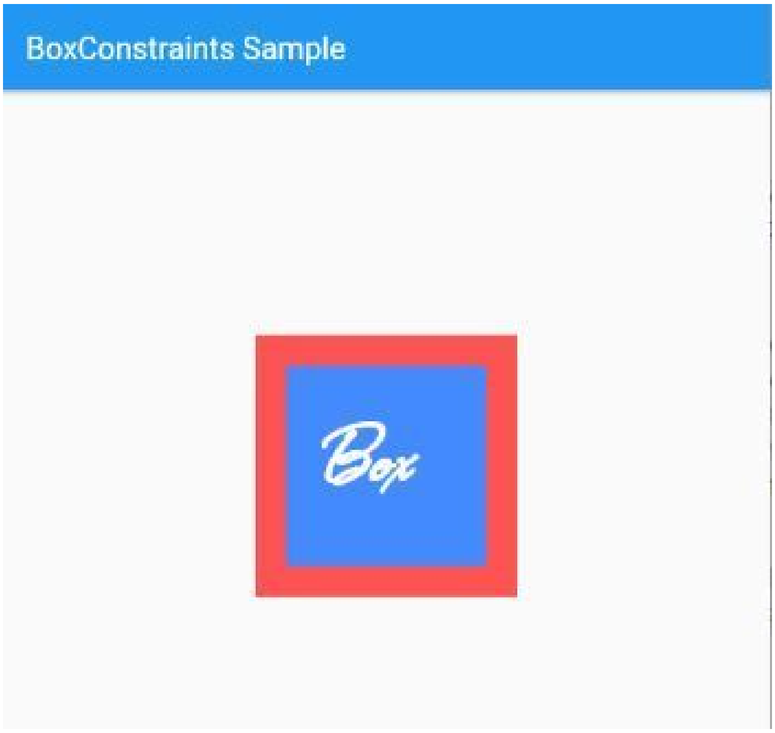


Figure 1.8 – BoxConstraints flutter example two

Understanding how these widgets or boxes handle the constraints in flutter is very important for the beginners.

In general, there are three kind of boxes that we'll encounter while we learn Flutter.

Firstly, the widgets like Center and ListView; they always try to be as big as possible.

Secondly, the widgets like Transform and Opacity always try to take the same size as their children.

And, finally, there are widgets like Image and Text that try to fit to a particular size.

As we'll progress, we'll find, how these constraints vary from widget to widget.

As example, we can remember the role of Center widget. It always maintains the maximum size. The minimum constraint does not work here.

What is Align in Flutter

Align widget allows us to place a child widget anywhere we want inside another widget.

Align is a widget that aligns its child within itself. Moreover, based on the child's size, it optionally sizes itself.

For instance, let us think about a minimal Flutter app that aligns a Flutter logo at top right.

Center(child: Column(children: [Container(height: 120.0, width: 120.0, color: Colors.blue[50], child: const Align(alignment: Alignment.topRight, child: FlutterLogo(size: 60,),),),

If we run the app, it looks like the following screenshot.

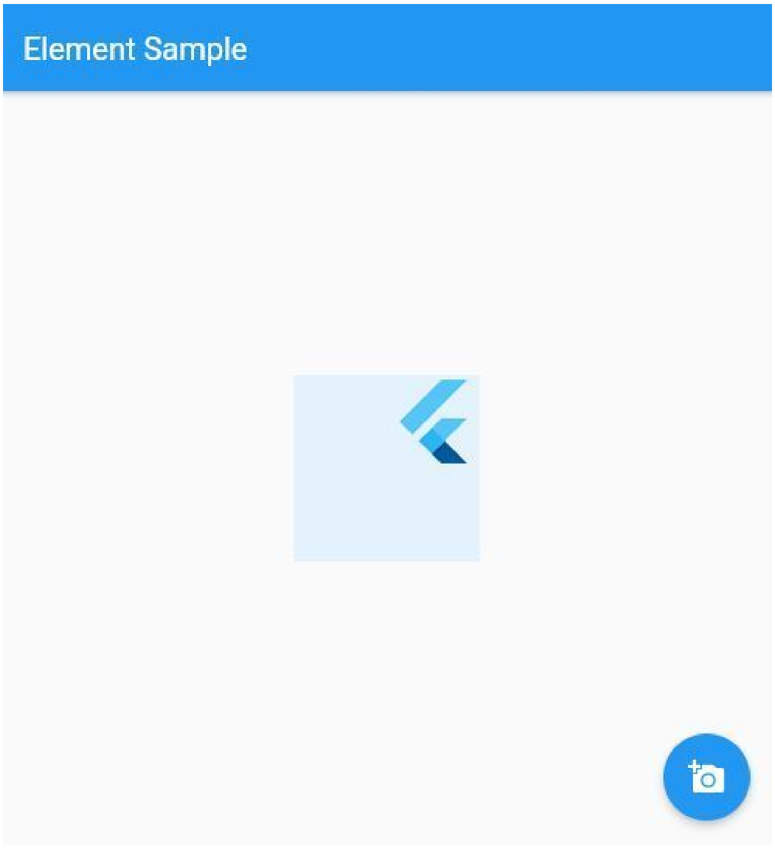


Figure 1.9 – Align widget at top right

The code is quite simple. It gives the Container a light blue color. Besides, it has definite width and height, which are tight constraints.

Now, as a child the Align widget places its child, a Flutter logo at the top right corner inside the Container.

However, we could have placed it with an alignment of Alignment.bottomRight.

To do that we should have given the Container a tight constraint, just like before, and that constraint should be bigger than the Flutter logo.

Next, we consider another piece of code that might place three Flutter logo at three different positions inside the same Container.

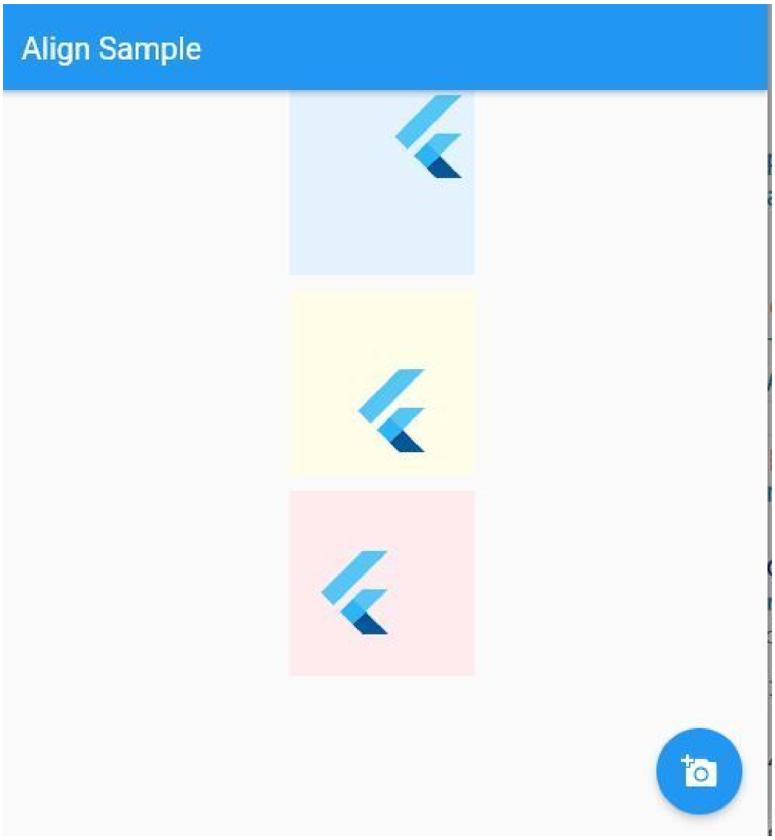


Figure 1.10 – Align widget aligns child at different positions

As we find in the above screenshot, three Flutter logos show up at three different positions.

Let's see the full code snippet first. After that, we'll discuss the code.

import 'package:flutter/material.dart';

class AlignSample extends StatelessWidget { const AlignSample({Key? key}) : super(key: key);

```dart
@override
Widget build(BuildContext context) {
  return const MaterialApp(
    title: 'Align Sample',
    debugShowCheckedModeBanner: false,
    home: AlignSampleHomme(),
  );
}
```

}

class AlignSampleHomme extends StatelessWidget { const AlignSampleHomme({Key? key}) : super(key: key);

```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Align Sample'),
    ),
    body: Center(
      child: Column(
        children: [
          Container(
            height: 120.0,
            width: 120.0,
            color: Colors.blue[50],
            child: const Align(
              alignment: Alignment.topRight,
              child: FlutterLogo(
                size: 60,
              ),
            ),
          ),
          const SizedBox(
            height: 10,
          ),
          Container(
            height: 120.0,
            width: 120.0,
            color: Colors.yellow[50],
            child: const Align(
              alignment: Alignment(0.2, 0.6),
              child: FlutterLogo(
                size: 60,
              ),
            ),
          ),
          const SizedBox(
            height: 10,
          ),
          Container(
            height: 120.0,
            width: 120.0,
            color: Colors.red[50],
            child: const Align(
              alignment: FractionalOffset(0.2, 0.6),
              child: FlutterLogo(
                size: 60,
              ),
            ),
          ),
        ],
      ),
    ),
    floatingActionButton: FloatingActionButton(
      onPressed: () {},
      child: const Icon(Icons.add_a_photo),
    ),
  );
}
```

}

The alignment property describes a point in the child's coordinate system and a different point in the coordinate system of this widget.

After that, the Align widget positions the child, here a Flutter logo, in a way so that both points are lined up on top of each other.

In the first case, the Align widget uses one of the defined constants from Alignment, which is Alignment.topRight.

Container(height: 120.0, width: 120.0, color: Colors.blue[50], child: const Align(alignment: Alignment.topRight, child: FlutterLogo(size: 60,),),),

As a result, this constant value places the FlutterLogo at the top right corner of the parent blue Container.

However, the second case is different, where the Alignment defines a single point.

Container(height: 120.0, width: 120.0, color: Colors.yellow[50], child: const Align(alignment: Alignment(0.2, 0.6), child: FlutterLogo(size: 60,),),),

It calculates the position of the Flutter logo in a different way.

The formula is, the result of (0.2 * width of FlutterLogo/2 + width of FlutterLogo/2) comes to a whole number, that is 36.0.

And this is a point in the coordinate system of Flutter logo.

The next point is defined by this formula – (0.6 * height of FlutterLogo/2 + height of FlutterLogo/2), which is equal to 48.0. Moreover, it's point in the coordinate system of the Align widget.

As a result Align will place the FlutterLogo at (36.0, 48.0) according to this coordinate system.

Although in the third example, the calculation goes in the same direction; yet the result differs with the previous one.

alignment: FractionalOffset(0.2, 0.6)

Consequently, the position of Flutter logo changes.

How to use aspect ratio widget

The AspectRatio Widget tries to find the best size to maintain aspect ratio of a child widget.

The AspectRatio widget attempts to size the child to a specific aspect ratio.

Suppose we have a Container widget with width 100, and height 100. In that case the aspect ratio would be 100/100; that is, 1.0.

Now, each Widget has its own constraints. As a result, the AspectRatio Widget tries to find the best size to maintain aspect ratio. However, while doing so it respects it's layout constraints.

Let's see a screenshot where we have used three different types of aspect ratio.

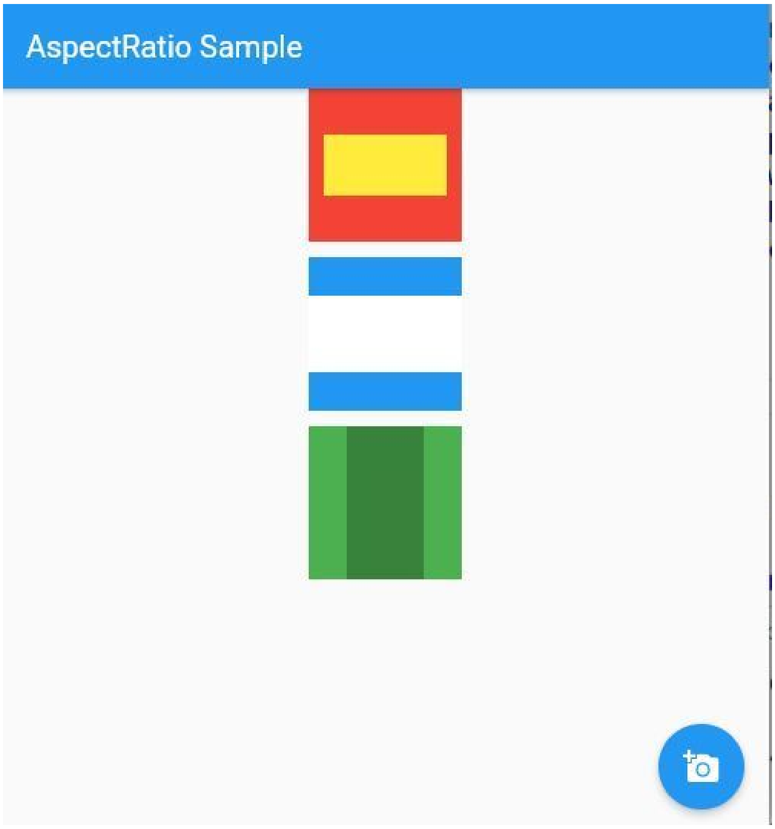


Figure 1.11 – AspectRatio widget in Flutter

A Container widget has an AspectRatio widget, which has a child Container in a different color.

As a result, we see different types of color combination.

Let’s see the full code now.

import ‘package:flutter/material.dart’;

class AspectRatioSample extends StatelessWidget { const AspectRatioSample({Key? key}) : super(key: key);

```
@override
Widget build(BuildContext context) {
  return const MaterialApp(
    title: 'AspectRatio Sample',
    debugShowCheckedModeBanner: false,
    home: AspectRatioSampleHomme(),
  );
}
```

}

class AspectRatioSampleHomme extends StatelessWidget { const AspectRatioSampleHomme({Key? key}) : super(key: key);

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('AspectRatio Sample'),
    ),
    body: Center(
      child: Column(
        children: [
          Container(
            color: Colors.red,
            alignment: Alignment.center,
            padding: const EdgeInsets.all(10),
            width: 100.0,
            height: 100.0,
            child: AspectRatio(
              aspectRatio: 2.0,
              child: Container(
                width: 50.0,
                height: 50.0,
                color: Colors.yellow,
              ),
            ),
          ),
          const SizedBox(
            height: 10,
          ),
          Container(
            color: Colors.blue,
            alignment: Alignment.center,
            width: 100.0,
            height: 100.0,
            child: AspectRatio(
              aspectRatio: 2.0,
              child: Container(
                width: 80.0,
                height: 70.0,
                color: Colors.white,
              ),
            ),
          ),
          const SizedBox(
            height: 10,
          ),
          Container(
            color: Colors.green,
            alignment: Alignment.center,
            width: 100.0,
            height: 100.0,
            child: AspectRatio(
              aspectRatio: 0.5,
              child: Container(
                width: 100.0,
                height: 50.0,
                color: Colors.black26,
              ),
            ),
          ),
        ],
      ),
    ),
    floatingActionButton: FloatingActionButton(
      onPressed: () {},
      child: const Icon(Icons.add_a_photo),
    ),
  );
}
```

}

Remember, in each case, the AspectRatio widget tries to find the best possible size and adjusts the child accordingly.

It comes to our help, when we try to change the size of an image on the fly.

What is Baseline in Flutter

When we try to position a child widget inside or outside of parent widget, Baseline helps us.

Baseline is a widget that positions its child according to the child widget's baseline.

Does it not make any sense?

Well, truly it didn't make any sense to me also, when I first encountered this widget.

In fact, the above statement doesn't really make any sense if we don't turn this abstraction into a concrete example.

Therefore, firstly, let's see one screenshot of a simple Flutter app where we've used Baseline widget.

Secondly, we'll look into the code and try to understand how this widget works.

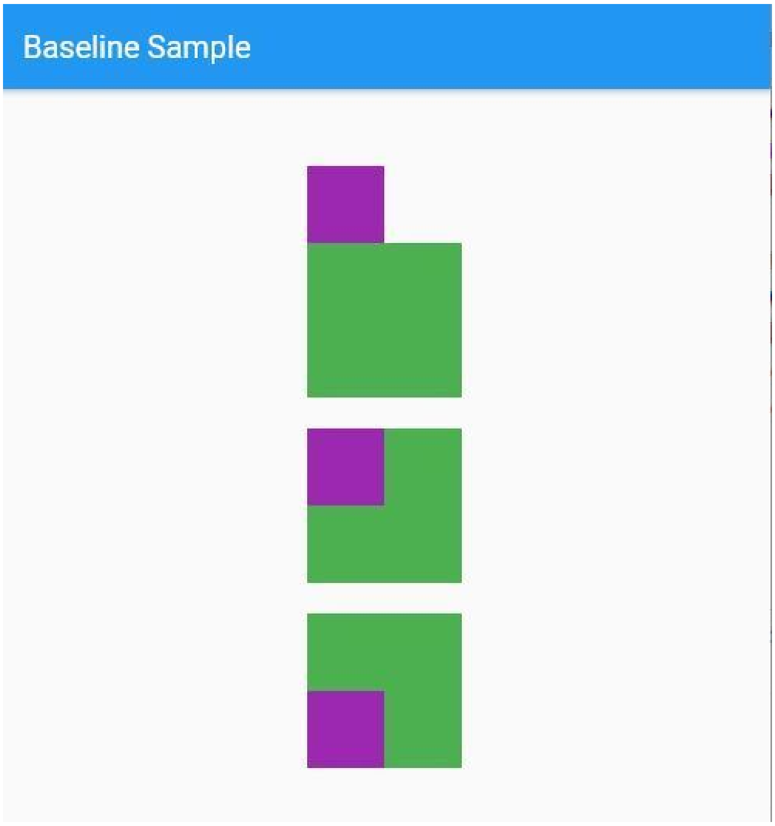


Figure 1.12 – Baseline widget first example

In the above image we see three Baseline examples. The first one consists of two Container widgets.

Let's see the code.

Container(width: 100, height: 100, color: Colors.green, child: Baseline(baseline: 0, baselineType: TextBaseline.alphabetic, child: Container(width: 50, height: 50, color: Colors.purple,),),),

The Baseline widget has two required parameters. The baseline, and the baselineType. The second parameter means the type of the baseline.

As a result, we need to supply value to those parameters.

The baseline parameter plays the most important role, of course. It requires a double value.

In the above case, the baseline is zero.

So it sits on top of the parent Container.

How does it happen?

It happens because the Baseline widget tries to shift the Child Container's bottom or baseline by calculating the distance from the top of the parent Container. Since it's zero, it cannot shift it. So it sits on its top.

As a result, it cannot enter the parent Container.

In the second case, the Child Container's baseline is 50.

Container(width: 100, height: 100, color: Colors.green, child: Baseline(baseline: 50, baselineType: TextBaseline.alphabetic, child: Container(width: 50, height: 50, color: Colors.purple,),),),

Therefore, the baseline logical pixels below the top of the parent Container is 50px. As a result, the bottom of the child Container shifts 50px inside the parent Container.

And the parent Container contains the child in the middle.

Take a look at the screenshot above, you'll understand how it works.

How do you use baseline in flutter?

Most importantly, we use Baseline widget when we want to position the child widget's bottom according to the distance from the top of the parent widget.

When the child Container's baseline is 100px, it moves its bottom 100px exactly, from the top of the parent Container.

As a consequence, the child's bottom merges with the parent's bottom. The above screenshot displays the same thing.

Container(width: 100, height: 100, color: Colors.green, child: Baseline(baseline: 100, baselineType: TextBaseline.alphabetic, child: Container(width: 50, height: 50, color: Colors.purple,),),), If we start increasing the value of the baseline, and make it 110px, what happens?

The next screenshot shows that.

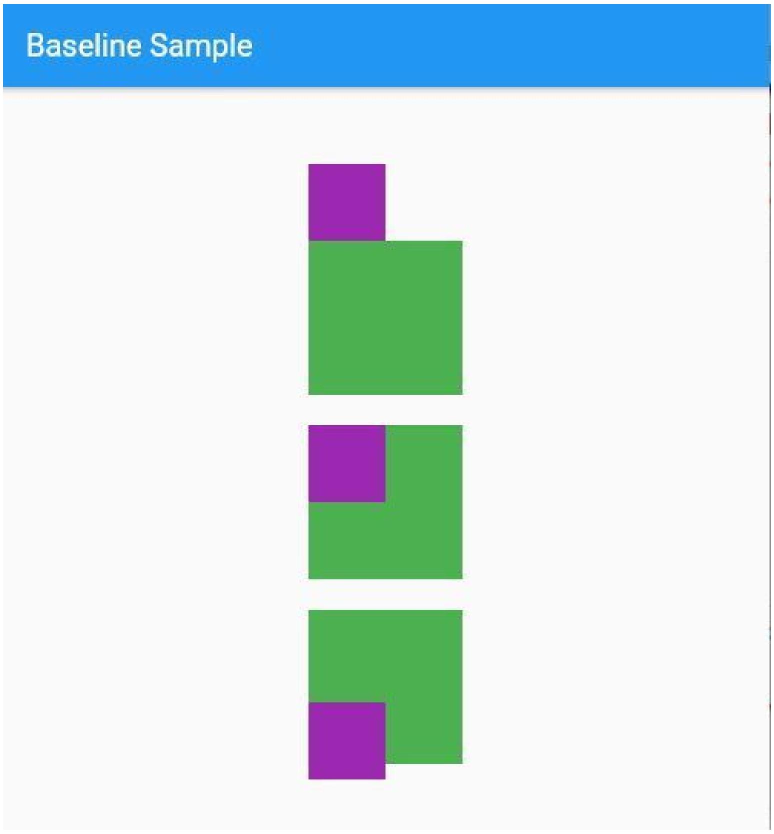


Figure 1.13 – Baseline widget second example

The child Container moves outside the parent Container.

Moreover, from the top of the parent Container's to the bottom of the child Container, the distance is 110px exact.

Let's take a look at the full code finally.

import 'package:flutter/material.dart';

class BaselineSample extends StatelessWidget { const BaselineSample({Key? key}) : super(key: key);

```
@override
Widget build(BuildContext context) {
  return const MaterialApp(
    title: 'Baseline Sample',
    debugShowCheckedModeBanner: false,
    home: BaselineSampleHomme(),
  );
}
```

}

class BaselineSampleHomme extends StatelessWidget { const BaselineSampleHomme({Key? key}) : super(key: key);

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Baseline Sample'),
    ),
    body: Center(
      child: Column(
        children: [
          const SizedBox(
            height: 100,
          ),
          Container(
            width: 100,
            height: 100,
            color: Colors.green,
            child: Baseline(
              baseline: 0,
              baselineType: TextBaseline.alphabetic,
              child: Container(
                width: 50,
                height: 50,
                color: Colors.purple,
              ),
            ),
          ),
          const SizedBox(
            height: 20,
          ),
          Container(
            width: 100,
            height: 100,
            color: Colors.green,
            child: Baseline(
              baseline: 50,
              baselineType: TextBaseline.alphabetic,
              child: Container(
                width: 50,
                height: 50,
                color: Colors.purple,
              ),
            ),
          ),
          const SizedBox(
            height: 20,
          ),
          Container(
            width: 100,
            height: 100,
            color: Colors.green,
            child: Baseline(
              baseline: 110,
              baselineType: TextBaseline.alphabetic,
              child: Container(
                width: 50,
                height: 50,
                color: Colors.purple,
                ),
              ),
            ),
          ],
        ),
      ),
    );
  }
```

}

We can provide a negative value to the baseline of the child Container. Try to make it -50.

At that instance, the bottom of the child Container moves away upward and goes out of the parent Container in the upward direction.

Try it. Happy fluttering.

2. How to implement a design by building layouts

To implement a design we need to start with custom paint object.

The full code repository for this section, please check the respective branches[8]

How to paint in Flutter?

CustomPaint painter parameter implements CustomPainter interface to paint in flutter.

The Custom Paint widget gives us opportunities to paint in Flutter.

You may wonder, what does paint mean in Flutter? Does it mean painting literally with colors and brushes?

Almost that is true.

The CustomPaint widget takes help from CustomPainter abstract class. Why?

Because CustomPainter again extends Listenable so that it not only gives us a canvas where we can paint on, but it creates a custom painter that repaints whenever repaint notifies listeners.

How does it take place?

Through Listenable object the Custom Painter object maintains a list of listeners.

[8]https://github.com/sanjibsinha/flutter_artisan/

The listeners are typically used to notify clients that the object has been updated.

Let’s take a look at the screenshot first.

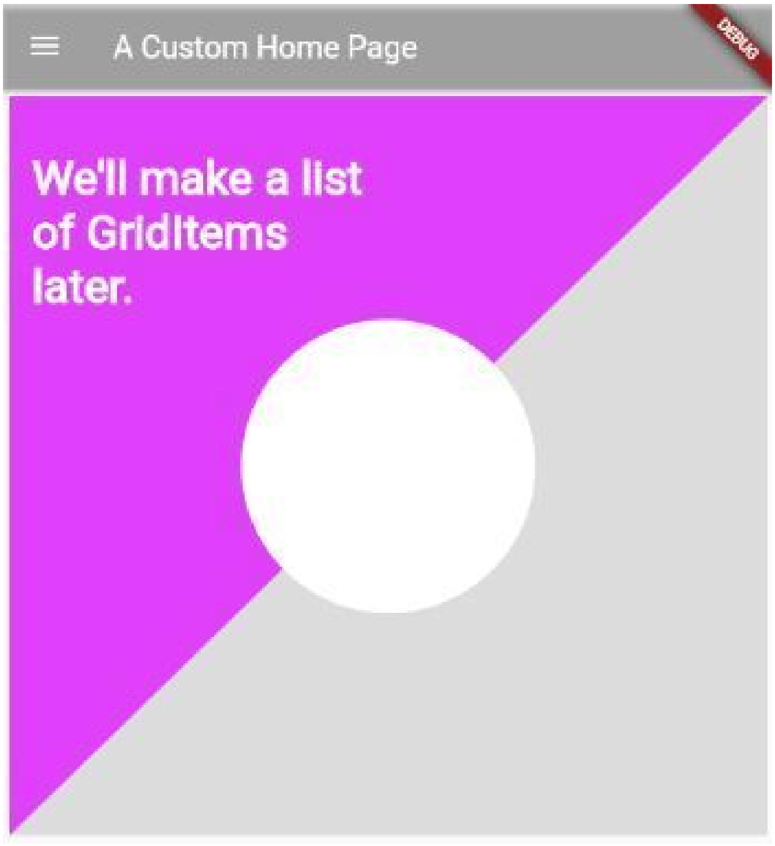


Figure 2.1 – The Back ground by custom paint object

How we have done that?

Take a look at the code where a subclass extends CustomPainter. Later we’ll use the Shaping Painter custom paint object as a parameter of Custom Paint widget.

```
class ShapingPainter extends CustomPainter {
@override
void paint(Canvas canvas, Size size) {
    final paint = Paint();

    /// setting the paint color grayish
    /// so it could cover the lower half of the screen
    ///
    paint.color = Colors.black12;

    /// Creating a rectangle with size and width same as \
the canvas
    /// It'll be going to cover the whole screen
    ///
    var rect = Rect.fromLTWH(0, 0, size.width, size.heigh\
t);

    /// Drawing the rectangle using the paint
    ///
    canvas.drawRect(rect, paint);

    /// Covering the upper half of the rectangle
    ///
    paint.color = Colors.purpleAccent;
    // Firstly, creating a path to form the shape
    var path = Path();
    path.lineTo(0, size.height);
    path.lineTo(size.width, 0);
    // Secondly, closing the path to form a bounded shape
    path.close();
    canvas.drawPath(path, paint);
    // Setting the color property of the paint
    paint.color = Colors.white;
    // Center of the canvas is (x,y) => (width/2, height/\
2)
```

```
    var center = Offset(size.width / 2, size.height / 2);
    // Finally, drawing the circle with center having rad\
ius 95.0
    canvas.drawCircle(center, 95.0, paint);
}

@override
bool shouldRepaint(CustomPainter oldDelegate) => false;
}
```

Please read the comments above. Hopefully that will clarify how we gradually paint the home page screen according our plan.

Now, we can set this paint as our background.

Moreover, we can change it as we wish.

Here, the CustomPaint and RenderCustomPaint has used the CustomPainter interface.

The paint method is called whenever the custom object needs to be repainted.

Now we can use the subclass in the following way.

```
import 'package:flutter/material.dart';

void main() => runApp(const MyApp());

class MyApp extends StatelessWidget {
const MyApp({Key? key}) : super(key: key);

@override
Widget build(BuildContext context) {
    //var size = MediaQuery.of(context).size;
    return const MaterialApp(
    title: 'A Custom Home Page',
    home: DashBoard(
        // size: size,
        ),
    );
}
}

class DashBoard extends StatelessWidget {
//final Size size;
const DashBoard({
    Key? key,
    //required this.size,
}) : super(key: key);

@override
Widget build(BuildContext context) {
    var size = MediaQuery.of(context).size;
    return MaterialApp(
    title: 'A Custom Home Page',

    /// ignore: todo
    ///TODO: we'll make a custom global theme later
    ///
    theme: ThemeData(
        primarySwatch: Colors.blue,
    ),
    home: Scaffold(
        appBar: AppBar(
        backgroundColor: Colors.black12,
        leading: const Icon(Icons.menu),
        title: const Text(
            "A Custom Home Page",
            textAlign: TextAlign.center,
        ),
        ),
        body: Stack(
        children: <Widget>[
            Container(
            padding: const EdgeInsets.all(5),
            child: CustomPaint(
                painter: ShapingPainter(),
                child: Container(
                height: size.height / 1,
                ),
            ),
            ),
            Container(
            margin: const EdgeInsets.only(top: 40),
            child: Padding(
                padding: const EdgeInsets.only(left: 20, \
right: 20),
                child: GridView.count(
                crossAxisCount: 2,
                children: const <Widget>[
                    Text(
                    'We\'ll make a list of GridItems late\
r.',
                    style: TextStyle(
                        color: Colors.white,
                        fontSize: 30,
                        fontWeight: FontWeight.bold,
                    ),
                    ),
                ],
                ),
            ),
            )
        ],
        ),
    ),
    );
```

```
}
}
```

In the above code, we've used the Stack and as a Background we used the CustomPaint painter parameter. That implements CustomPainter interface.

CustomPaint is a widget that provides a canvas on which to draw during the paint phase.

Firstly, CustomPaint asks its painter to paint on the current canvas, then it paints its child, and then, after painting its child, it asks its foregroundPainter to paint.

Secondly, the coordinate system of the canvas matches the coordinate system of the CustomPaint object.

What is layout in Flutter

Design and layout in Flutter consists a mixture of visible and invisible widgets.

Layout in Flutter is controlled by Widgets. Some of them are visible such as Text, Image or Icon. However, some of them are also invisible, like Row, Column, or Stack.

Although invisible, yet Row, Column or Stack play very important role in building layout in Flutter.

As we know, almost everything in Flutter is Widget; therefore, layout model is no exception. According to the layout model, the invisible widgets use constraint, align, aspect ratio, baseline and other Widgets to arrange visible widgets.

We always create a layout by composing Widgets to build more complex Widget structures, or tree.

Consider the following screenshot. That will explain how we've built this layout by mixing visible and invisible widgets.

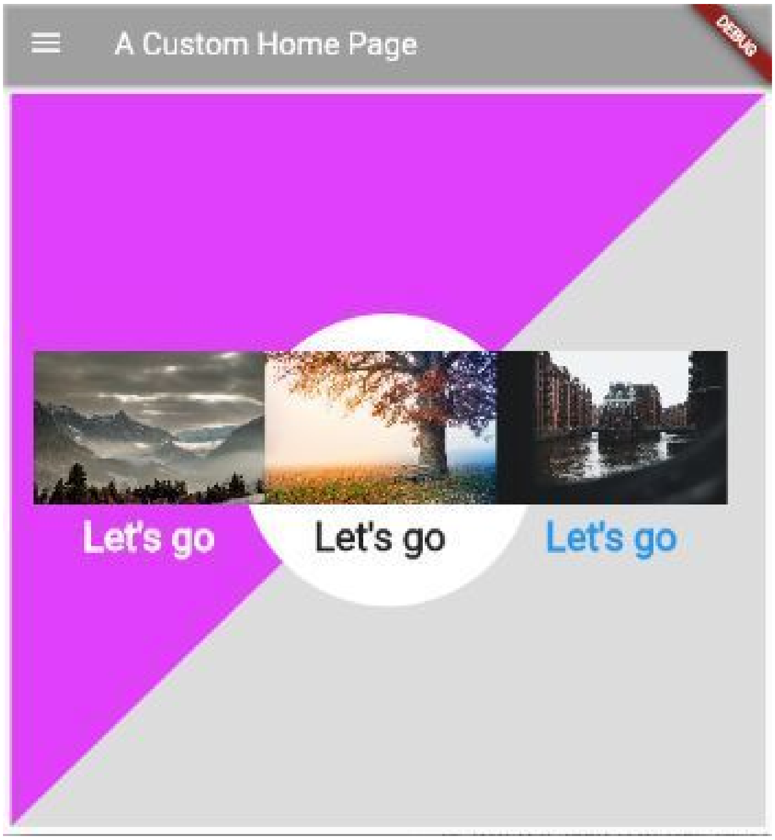


Figure 2.2 – Basic Layout widgets example in Flutter

In the previous section we've discussed how we can paint our canvas or screen in Flutter. As an extension, we've used the same background; and, after that we design our layout on top of that background.

How do I create a layout in Flutter?

Now, the time has come to take a look at the full code snippet.

```
import 'package:flutter/material.dart';

void main() => runApp(const MyApp());

class MyApp extends StatelessWidget {
const MyApp({Key? key}) : super(key: key);

@override
Widget build(BuildContext context) {
    return const MaterialApp(
    title: 'A Custom Home Page',
    home: DashBoard(
        // size: size,
        ),
    );
}
}

class DashBoard extends StatelessWidget {
//final Size size;
const DashBoard({
    Key? key,
    //required this.size,
}) : super(key: key);

@override
Widget build(BuildContext context) {
    var size = MediaQuery.of(context).size;
    return MaterialApp(
    title: 'A Custom Home Page',

    /// ignore: todo
    ///TODO: we'll make a custom global theme later
    ///

    theme: ThemeData(
        primarySwatch: Colors.blue,
    ),
    home: Scaffold(
        appBar: AppBar(
        backgroundColor: Colors.black12,
        leading: const Icon(Icons.menu),
        title: const Text(
            "A Custom Home Page",
            textAlign: TextAlign.center,
        ),
        ),
        body: Stack(
        children: <Widget>[
            Container(
            padding: const EdgeInsets.all(5),
            child: CustomPaint(
                painter: ShapingPainter(),
                child: Container(
                height: size.height / 1,
                ),
            ),
            ),
            Container(
            margin: const EdgeInsets.all(10),
            child: Padding(
                padding: const EdgeInsets.all(10),
                child: Row(
                children: [
                    Column(
                    mainAxisAlignment: MainAxisAlignment.\
center,
                    children: [
                        Image.network(
                        'https://cdn.pixabay.com/photo/20\
21/12/05/10/28/nature-6847175_960_720.jpg',
                        width: 150,
                        height: 100,
                        ),
                        Container(
                        padding: const EdgeInsets.all(7),
                        child: const Text(
                            'Let\'s go',
                            style: TextStyle(
                            fontSize: 25,
                            fontWeight: FontWeight.bold,
                            color: Colors.white,
                            ),
                        ),
                        ),
                    ],
                    ),
                    Column(
                    mainAxisAlignment: MainAxisAlignment.\
center,
                    children: [
                        Image.network(
                        'https://cdn.pixabay.com/photo/20\
21/11/13/23/06/tree-6792528_960_720.jpg',
                        width: 150,
                        height: 100,
                        ),
                        Container(
                        padding: const EdgeInsets.all(7),
                        child: const Text(
                            'Let\'s go',
                            style: TextStyle(
                            fontSize: 25,
                            fontWeight: FontWeight.bold,
                            ),
                        ),
                        ),
                    ],
                    ),
                    Column(
                    mainAxisAlignment: MainAxisAlignment.\
center,
                    children: [
                        Image.network(
                        'https://cdn.pixabay.com/photo/20\
21/12/12/20/26/flow-6866055_960_720.jpg',
                        width: 150,
                        height: 100,
                        ),
                        Container(
                        padding: const EdgeInsets.all(7),
                        child: const Text(
                            'Let\'s go',
                            style: TextStyle(
                            fontSize: 25,
                            fontWeight: FontWeight.bold,
                            color: Colors.blue,
                            ),
                        ),
                        ),
                    ],
                    ),
                ],
                ),
            ),
            )
        ],
        ),
    ),
    );
```

```
}
}

class ShapingPainter extends CustomPainter {
@override
void paint(Canvas canvas, Size size) {
    final paint = Paint();

    /// setting the paint color greyish
    /// so it could cover the lower half of the screen
    ///
    paint.color = Colors.black12;

    /// Creating a rectangle with size and width same as \
the canvas
    /// It'll be going to cover the whole screen
    ///
    var rect = Rect.fromLTWH(0, 0, size.width, size.heigh\
t);

    /// Drawing the rectangle using the paint
    ///
    canvas.drawRect(rect, paint);

    /// Covering the upper half of the rectangle
    ///
    paint.color = Colors.purpleAccent;
    // Firstly, creating a path to form the shape
    var path = Path();
    path.lineTo(0, size.height);
    path.lineTo(size.width, 0);
    // Secondly, closing the path to form a bounded shape
    path.close();
    canvas.drawPath(path, paint);
    // Setting the color property of the paint
    paint.color = Colors.white;
    // Center of the canvas is (x,y) => (width/2, height/\
2)
    var center = Offset(size.width / 2, size.height / 2);
    // Finally, drawing the circle with center having rad\
ius 95.0
    canvas.drawCircle(center, 95.0, paint);
}

@override
bool shouldRepaint(CustomPainter oldDelegate) => false;
}
```

If we read the code carefully, what we'll find?

We find that we've designed a layout that starts with the Stack widget.

We'll discuss Stack in great detail later, so don't worry. You can always use the search button in the website to find out what you need.

Stack widget basically keeps other widgets on top of the base widget. One after another. Therefore, here we've painted our background as the base widget.

```
body: Stack(
        children: <Widget>[
            Container(
            padding: const EdgeInsets.all(5),
            child: CustomPaint(
                painter: ShapingPainter(),
                child: Container(
                height: size.height / 1,
                ),
            ),
            ),
...
/// the following widgets will be placed on top of this C\
ontainer
```

As a result, first we've designed a background layout first. To do that, we've first created a Custom Painter Widget "ShapingPainter" and passed it as the named parameter of CustomPaint widget.

```
child: CustomPaint(
                painter: ShapingPainter(),
                child: Container(
                height: size.height / 1,
                ),
            ),
...
```

As a result, the base widget of our Stack works as a background. For the respective code snippet please visit the GitHub repository.

Next, on top of this base background, we've created the Widget tree using visible and invisible widgets.

Now, as we might expect, it looks like that. Each Column inside the Row widget, has Image, Container, Text, and TextStyle widgets. We've used a Container to add padding, margin, alignment etc.

For instance, each Text widget is placed inside a Container to add padding or margin.

The rest of the design and layout follow the rules that we've set by controlling different properties.

Let's think about Row, or Column. Although they are invisible, still they can specify how their children will be aligned, vertically, or horizontally. Moreover, they can set how much space the children widgets should occupy.

How we build Flutter Layout

Building a Flutter Layout is easy. However, we need to improvise and place widgets wisely.

In our previous section we've learned some key components of a layout model in Flutter. Now, in this section, we would like to build a layout on top of that.

According to the layout model, the invisible widgets use constraint, align, aspect ratio, baseline and other Widgets to arrange visible widgets such as Text, Image or Icon. However, some of them, again are also invisible.

We have seen how invisible widgets like Row, Column, or Stack have been used in our previous section – What is Layout in Flutter?

If you're a beginner and have interest in learning Flutter layout from scratch, you might take a look at the respective GitHub repository.

Now, we want to move forward. Therefore, we want to make it sure that, we can add more rows like the above screenshot and, moreover, we can scroll down to the lower part, as well.

To make it happen, we need a scrolling widget like ListView. Although the Column widget places its children widgets in its main axis vertically, but after a certain limit it exhausts. And the overflow error is thrown.

Just to avoid such accidental hiccups, let us break our code in several custom widgets, first.

We keep the custom Custom Paint object in the model folder. This pain object extends abstract class Custom Painter, and builds our background.

```
import 'package:flutter/material.dart';

class ShapingPainter extends CustomPainter {
@override
void paint(Canvas canvas, Size size) {
    final paint = Paint();

    /// setting the paint color greyish
    /// so it could cover the lower half of the screen
    ///
    paint.color = Colors.black12;

    /// Creating a rectangle with size and width same as \
the canvas
    /// It'll be going to cover the whole screen
    ///
    var rect = Rect.fromLTWH(0, 0, size.width, size.heigh\
t);

    /// Drawing the rectangle using the paint
    ///
    canvas.drawRect(rect, paint);

    /// Covering the upper half of the rectangle
    ///
    paint.color = Colors.purpleAccent;
    // Firstly, creating a path to form the shape
    var path = Path();
    path.lineTo(0, size.height);
    path.lineTo(size.width, 0);
    // Secondly, closing the path to form a bounded shape
    path.close();
    canvas.drawPath(path, paint);
    // Setting the color property of the paint
    paint.color = Colors.white;
    // Center of the canvas is (x,y) => (width/2, height/\
2)
    var center = Offset(size.width / 2, size.height / 2);
    // Finally, drawing the circle with center having rad\
ius 95.0
    canvas.drawCircle(center, 95.0, paint);
}

@override
bool shouldRepaint(CustomPainter oldDelegate) => false;
}
```

Now, let's start building the layout with the root app.

```
import 'package:flutter/material.dart';

import 'view/my_app.dart';

void main() => runApp(const MyApp());
The My App follows the Material Design guidelines and ext\
ends those properties to a custom widget Dashboard Home.

import 'package:flutter/material.dart';

import 'dash_board.dart';

class MyApp extends StatelessWidget {
const MyApp({Key? key}) : super(key: key);

@override
Widget build(BuildContext context) {
    //var size = MediaQuery.of(context).size;
    return const MaterialApp(
    title: 'A Custom Home Page',
    home: DashBoardHome(
        // size: size,
        ),
    );
}
}
```

The Dashboard Home scaffolds the body where we keep three Container widgets inside a ListView.

```
import 'package:flutter/material.dart';
import '/model/shaping_painter.dart';

import 'all_containers.dart';

class DashBoardHome extends StatelessWidget {
const DashBoardHome({
    Key? key,
}) : super(key: key);

@override
Widget build(BuildContext context) {
    var size = MediaQuery.of(context).size;
    return Scaffold(
    appBar: AppBar(
        backgroundColor: Colors.black12,
        leading: const Icon(Icons.menu),
        title: const Text(
        'Let\'s Go!',
        textAlign: TextAlign.center,
        ),
    ),
    body: Stack(
        children: <Widget>[
        Container(
            padding: const EdgeInsets.all(5),
            child: CustomPaint(
            painter: ShapingPainter(),
            child: Container(
                height: size.height / 1,
            ),
            ),
        ),
        ListView(
            children: const [
            FirstContainer(),
            SecondContainer(),
            ThirdContainer()
            ],
        ),
        ],
    ),
    );
}
}
```

Therefore, we need to take a look at the file where we've kept all containers. Each Containers will have three rows that again will have three Column widgets each.

```
import 'package:flutter/material.dart';

import 'all_columns.dart';

class FirstContainer extends StatelessWidget {
const FirstContainer({
    Key? key,
}) : super(key: key);

@override
Widget build(BuildContext context) {
    return Container(
    margin: const EdgeInsets.all(10),
    child: Padding(
        padding: const EdgeInsets.all(10),
        child: Row(
        children: const [
            FirstColumn(),
            SecondColumn(),
            ThirdColumn(),
        ],
        ),
    ),
    );
}
}

class SecondContainer extends StatelessWidget {
const SecondContainer({
    Key? key,
}) : super(key: key);

@override
Widget build(BuildContext context) {
    return Container(
    margin: const EdgeInsets.all(10),
    child: Padding(
        padding: const EdgeInsets.all(10),
        child: Row(
        children: const [
            FirstColumn(),
            SecondColumn(),
            ThirdColumn(),
        ],
        ),
    ),
    );
}
}

class ThirdContainer extends StatelessWidget {
const ThirdContainer({
    Key? key,
}) : super(key: key);

@override
Widget build(BuildContext context) {
    return Container(
    margin: const EdgeInsets.all(10),
    child: Padding(
        padding: const EdgeInsets.all(10),
        child: Row(
        children: const [
            FirstColumn(),
            SecondColumn(),
            ThirdColumn(),
        ],
        ),
    ),
    );
}
}
```

We can place each Column widget in a single file. So that it looks like the following screenshot.

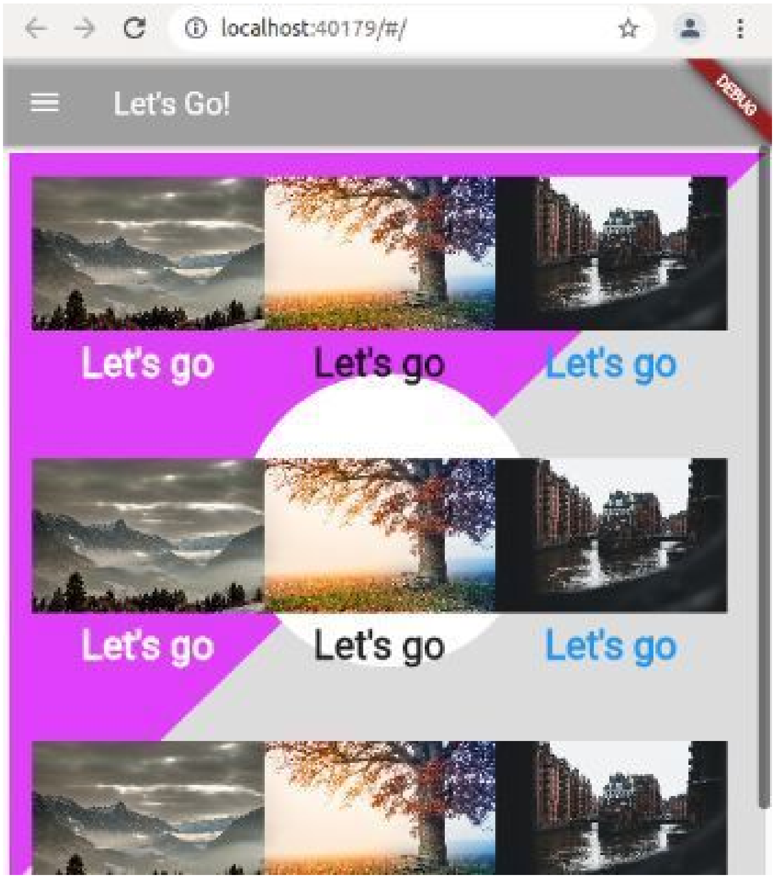


Figure 2.3 – Building layout in Flutter with ListView, the upper part

However, that's the upper portion of the screen. Before we take a look at the lower part, let's take a look at the code where we've kept the Column widgets.

```
import 'package:flutter/material.dart';

class ThirdColumn extends StatelessWidget {
const ThirdColumn({
    Key? key,
}) : super(key: key);

@override
Widget build(BuildContext context) {
    return Column(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
        Image.network(
        'https://cdn.pixabay.com/photo/2021/12/12/20/26/f\
low-6866055_960_720.jpg',
        width: 150,
        height: 100,
        ),
        Container(
        padding: const EdgeInsets.all(7),
        child: const Text(
            'Let\'s go',
            style: TextStyle(
            fontSize: 25,
            fontWeight: FontWeight.bold,
            color: Colors.blue,
            ),
        ),
        ),
    ],
    );
}
}

class SecondColumn extends StatelessWidget {
```

```
const SecondColumn({
    Key? key,
}) : super(key: key);

@override
Widget build(BuildContext context) {
    return Column(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
        Image.network(
        'https://cdn.pixabay.com/photo/2021/11/13/23/06/t\
ree-6792528_960_720.jpg',
        width: 150,
        height: 100,
        ),
        Container(
        padding: const EdgeInsets.all(7),
        child: const Text(
            'Let\'s go',
            style: TextStyle(
            fontSize: 25,
            fontWeight: FontWeight.bold,
            ),
        ),
        ),
    ],
    );
}
}

class FirstColumn extends StatelessWidget {
const FirstColumn({
    Key? key,
}) : super(key: key);

```

```
@override
Widget build(BuildContext context) {
    return Column(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
        Image.network(
        'https://cdn.pixabay.com/photo/2021/12/05/10/28/n\
ature-6847175_960_720.jpg',
        width: 150,
        height: 100,
        ),
        Container(
        padding: const EdgeInsets.all(7),
        child: const Text(
            'Let\'s go',
            style: TextStyle(
            fontSize: 25,
            fontWeight: FontWeight.bold,
            color: Colors.white,
            ),
        ),
        ),
    ],
    );
}
}
```

As we can see, each Column widget places two child widgets in its main axis. One is Image and the other is the Text widget.

Finally, we've successfully built the layout model based on the principle that we can break the main widget tree in several small widget trees that will again house several different type of Widgets. Of these widgets, some are visible and some of them invisible.

Now, we can scroll down and take a look at the lower portion of the screen.

Our next challenge will be to make a more complex tree of widgets and layout based on the same principle. For this code snippet, please visit the respective GitHub repository.

Till then, stay tuned and keep reading on. Happy reading and coding Flutter.

What is an AppBar? How do you use AppBar in Flutter?

Do you have web development experience? In that case, you must have heard about the header section.

In a mobile app, built in Flutter, AppBar plays almost the same role. Although not similar in every sense, yet they have similarities.

We use AppBar in Scaffold.appbar property. At the top of the screen, app bar places itself as a fixed bar widget. However, we can control its size and other functionalities.

Moreover, you may think app bar as a continuation of material design component.

How do you use AppBar in Flutter?

An app bar consists of a toolbar and other widgets that we're going to see in a minute. In addition, an app bar exposes many kind of actions with icon buttons.

To get an idea, we can think of an app where the front page app bar takes us to the second page. For second page, flutter by default makes a system to go back to the previous page.

At the top of the screen, we can see the app bar. On the left side, there is leading property. In between the it places the title. On the right side, there are icon buttons.

Flutter manages the app bar's padding, so it can handle the system UI's intrusion. For that reason the body part never intrudes into the app bar's section.

In this app, we've followed the same custom theme. In addition, we've not changed the material design components.

```dart
import 'package:flutter/material.dart';
import '../controllers/custom_theme.dart';

class HowAppbarWorks extends StatelessWidget {
const HowAppbarWorks({Key? key}) : super(key: key);

static const String _title = 'How Appbar Works';

@override
Widget build(BuildContext context) {
return MaterialApp(
title: _title,
home: AppbarFirstHome(),
theme: customTheme(),
debugShowCheckedModeBanner: false,
);
}
}
```

Navigate to second page through AppBar As the home property indicates the home page, we've defined it too as the following. And in that page, we've defined the app bar properties.

```
class AppbarFirstHome extends StatelessWidget {
const AppbarFirstHome({Key? key}) : super(key: key);

@override
Widget build(BuildContext context) {
return Scaffold(
appBar: AppBar(
leading: Builder(
builder: (BuildContext context) {
return IconButton(
icon: const Icon(Icons.menu),
onPressed: () {
Scaffold.of(context).openDrawer();
},
tooltip: MaterialLocalizations.of(context).openAppDrawerT\
ooltip,
);
},
),
title: const Text('AppBar Home Page'),
actions: <Widget>[
IconButton(
icon: const Icon(Icons.add_alert),
tooltip: 'Show Snackbar',
onPressed: () {
ScaffoldMessenger.of(context).showSnackBar(
const SnackBar(
content: Text('This is a snackbar'),
),
);
},
),
IconButton(
icon: const Icon(Icons.navigate_next),
tooltip: 'Go to the second page',
onPressed: () {
Navigator.push(context, MaterialPageRoute<void>(
builder: (BuildContext context) {
return AppbarSecondHome();
},
));
},
),
],
),
body: AppbarHomePage(),
);
}
}
```

Because of the defined app bar's property, we can click the bell icon and it opens the snack bar widget at the bottom.

Not only showing the home page, but also we can move to the second page through the app bar.

However, to do that, we need to use the following lines of code inside app bar:

```
IconButton(
icon: const Icon(Icons.navigate_next),
tooltip: 'Go to the second page',
onPressed: () {
Navigator.push(context, MaterialPageRoute<void>(
builder: (BuildContext context) {
return AppbarSecondHome();
},
));
},
),
```

The second page shows the back button at its app bar. Consequently, by clicking the back button, we can easily go back to the first page. Moreover, we can again go to other pages, as per the design allows.

For full snippet of code regarding the material design components, and other essential widgets, you may visit the GitHub repository.

AppBar Flutter: How to use AppBar right way?

AppBar in Flutter plays a key role to give users a rich experience. Here is a complete guide.

On Android toolbars represent AppBar. Certainly, it's another Material design widget that we've discussed earlier.

To use AppBar we firstly need another widget – Scaffold.

As we've just learned, AppBar consists of many toolbars.

Therefore, we can use our AppBar in many ways. We can navigate to another page, we can use search icon to search our app, and many more.

We'll see that in a minute.

Before that, to see more Material design widgets in one place, let's organize our code. Let's create a folder "controllers" and "views".

In "controllers" we create our AppBar widget. In addition, we can visit the full code snippet in this GitHub repository.

```dart
import 'package:flutter/material.dart';
import 'package:flutter_artisan/views/app_bar_home.dart';

class AppBarWidget extends StatelessWidget {
const AppBarWidget({Key? key}) : super(key: key);

@override
Widget build(BuildContext context) {
    return const AppBarHome();
}
}
```

And inside we have an AppBar home page, which will display all the AppBar tools.

```dart
import 'package:flutter/material.dart';
import 'package:flutter_artisan/views/app_bar_next.dart';

class AppBarHome extends StatelessWidget {
const AppBarHome({Key? key}) : super(key: key);

@override
Widget build(BuildContext context) {
    return Scaffold(
    appBar: AppBar(
        title: const Text('AppBar Example'),
        actions: <Widget>[
        IconButton(
            icon: const Icon(Icons.add_alert),
            tooltip: 'Show Snackbar',
            onPressed: () {
            ScaffoldMessenger.of(context).showSnackBar(
                const SnackBar(
                content: Text('A SnackBar'),
                ),
            );
            },
        ),
        IconButton(
            icon: const Icon(Icons.search_outlined),
            tooltip: 'Search',
            onPressed: () {
            // our code
            },
        ),
        IconButton(
            icon: const Icon(Icons.navigate_next),
            tooltip: 'Next page',
            onPressed: () {
            Navigator.push(
                context,
                MaterialPageRoute<void>(
                builder: (BuildContext context) {
                    return const AppBarNext();
                },
                ),
            );
            },
        ),
        ],
    ),
    body: const Center(
        child: Text(
        'This is the AppBar Example home page',
        style: TextStyle(fontSize: 30),
        textAlign: TextAlign.center,
        ),
    ),
    );
}
```

```
}
```

Before explaining the code, let's take a look at how our AppBar home page looks like.

As we've learned earlier, an app bar, resting on the top of our flutter app, consists of a toolbar and potentially other widgets.

That may include TabBar and a FlexibleSpaceBar, which we'll discuss later.

The most important role of AppBar is it expose one or more common actions with IconButtons.

In our case, we can see one bell icon, search icon and navigate icon.

They all have one common named parameter – onTap method. With the help of that method, we can even change the state of the flutter app.

That's why, the nearest ancestor of AppBar widget is StatefulWidget.

Flutter places AppBar as a fixed height widget at the top of the screen. It happens because App bars are typically used in the Scaffold.appBar property.

We can even add scrollable AppBar. However, that's a different topic. For a scrollable app bar, we use SliverAppBar, which embeds an AppBar in a sliver for use in a CustomScrollView.

Usually, any AppBar displays the toolbar widgets, such as leading, title, and actions.

In the above code, we've used actions to display our icons.

```
appBar: AppBar(
        title: const Text('AppBar Example'),
        actions: <Widget>[
        IconButton(
            icon: const Icon(Icons.add_alert),
            tooltip: 'Show Snackbar',
            onPressed: () {
            ScaffoldMessenger.of(context).showSnackBar(
                const SnackBar(
                content: Text('A SnackBar'),
                ),
            );
            },
        ),
...
```

Where do I put AppBar in flutter?

The advantage we enjoy in Flutter is that it allows us to create menu, search,or leading button in app bar.

With a single AppBar property, we can create several functionalities in our Flutter app.

If we click the bell icon button, it opens up the snack bar.

```
IconButton(
            icon: const Icon(Icons.add_alert),
            tooltip: 'Show Snackbar',
            onPressed: () {
            ScaffoldMessenger.of(context).showSnackBar(
                const SnackBar(
                content: Text('A SnackBar'),
                ),
            );
```

```
                },
            ),
...
```

In a separate article we'll discuss the role of SnackBar widget.

As a result we can see that snack bar pops up at the bottom of our app.

Inside our SnackBar, we can place many more widgets.

Subsequently, we can navigate to another page from AppBar.

```
IconButton(
                icon: const Icon(Icons.navigate_next),
                tooltip: 'Next page',
                onPressed: () {
                Navigator.push(
                    context,
                    MaterialPageRoute<void>(
                    builder: (BuildContext context) {
                        return const AppBarNext();
                    },
                    ),
                );
                },
            ),
...
```

We have placed the custom stateless AppBarNext widget inside our views folder. Let's take a look at the code first.

```
import 'package:flutter/material.dart';
import 'package:flutter/widgets.dart';

class AppBarNext extends StatelessWidget {
const AppBarNext({Key? key}) : super(key: key);

@override
Widget build(BuildContext context) {
    return Scaffold(
    appBar: AppBar(
        title: const Text('App Bar Next Page'),
    ),
    body: const AppBarNextPage(),
    );
}
}

class AppBarNextPage extends StatelessWidget {
const AppBarNextPage({Key? key}) : super(key: key);

@override
Widget build(BuildContext context) {
    return Center(
    child: Container(
        margin: const EdgeInsets.all(
        10,
        ),
        padding: const EdgeInsets.all(
        10,
        ),
        child: const Text(
        'AppBar Next Page',
        style: TextStyle(
            fontWeight: FontWeight.bold,
            fontSize: 30,
          ),
          ),
      ),
      );
}
}
```

Most importantly, we can use another AppBar, because it's another page. And, of course, we can add other functionalities in this AppBar.

As a result, for this page we've used another AppBar.

How do I style AppBar flutter?

Remember, every page may have different AppBar in Flutter. Therefore, to style AppBar we can use different approaches.

In the above code snippets we've restricted ourselves to certain features only.

But any AppBar comes with many functionalities that include background color, text font, leading, and many more.

We can change the alignment or position of the title in our AppBar.

In the above code snippet, we've kept it on the left side. However, if we feel, we change its position and move it either to the center or right.

In actions widget, we can add more icons with different functionalities.

That includes a pop up menu.

It depends on what type of Flutter app we're going to build.

In this Material design series we'll discuss more useful widgets, so please stay tuned.

What is SliverAppBar in flutter?

SliverAppBar is a special type of AppBar in flutter that can change its appearance and collapses as we scroll up or down.

SliverAppBar is a material design component widget, which must have a MaterialApp as its ancestors.

However, the name suggests that it is a kind of AppBar.

Then what it is exactly? And how do we use a SliverAppBar?

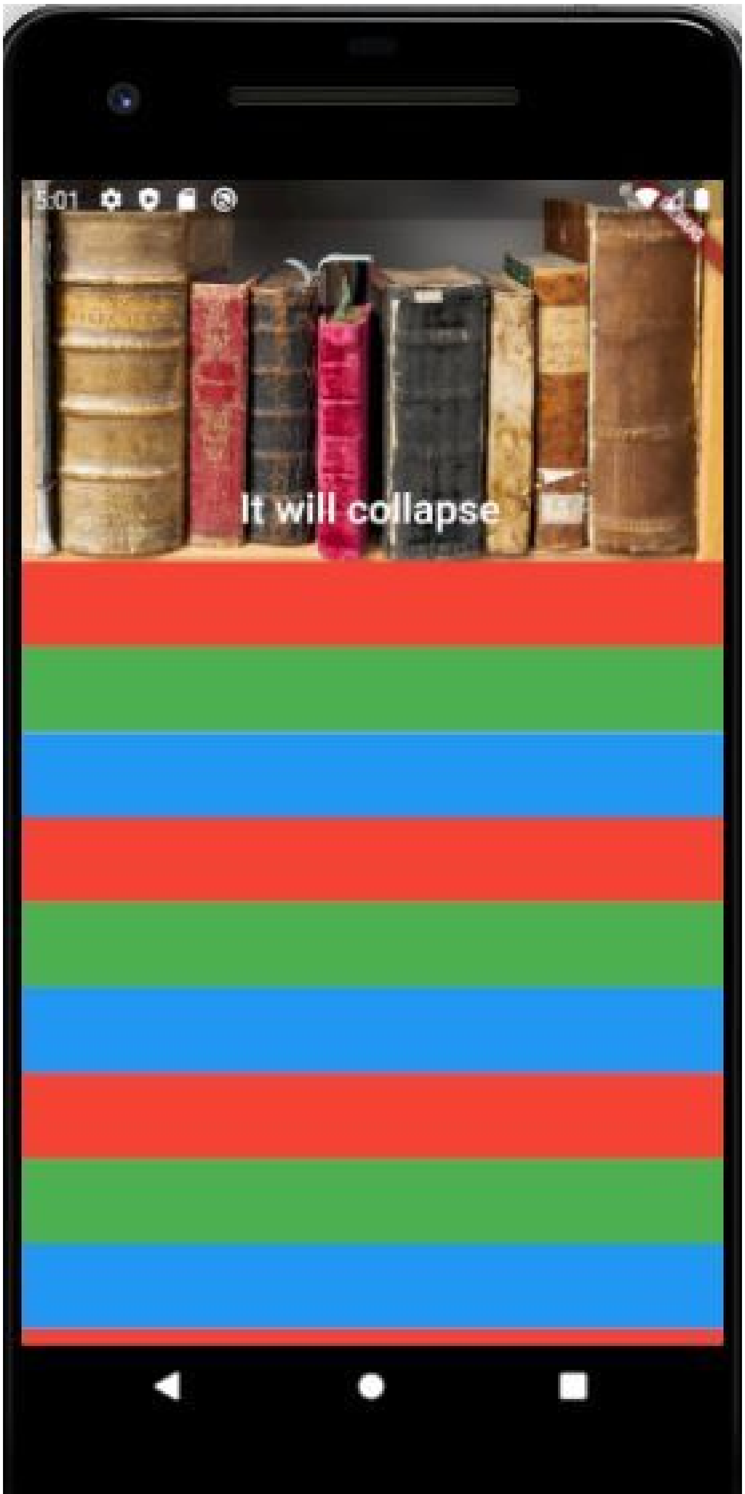
Here, the word sliver is important, because it defines a scrollable area in the AppBar, which has the ability to collapse.

As a result, we can conclude that SliverAppBar is a kind of AppBar that can change with body of our flutter app.

To make it more clear, we can add that this particular AppBar can change its appearance, if we scroll up it can shrink and gets smaller in size, even it can disappear. And, on the contrary, it can also gets to its normal size by blending with the body, as we scroll down again.

As a result, we can use an image as its background. Since it occupies the upper part, when we scroll up or down, it plays the role of a navigation bar in iOS and acts as a toolbar in Android app.

Let’s see two images one after the other, so that we can have a more clear picture of what is SliverAppBar and how it works.


5:01
It will collapse

Now, if we scroll up to see the bottom part, the image disappears and the text only remains. The SliverAppBar collapses.

To make it simple, we will take a look at the full code where we'll show how SliverAppBar integrates with a CustomScrollView.

```
import 'package:flutter/material.dart';

class SliverAppBarExample extends StatelessWidget {
const SliverAppBarExample({Key? key}) : super(key: key);

@override
Widget build(BuildContext context) {
    return MaterialApp(
    title: 'Sliver AppBar Example',
    home: SliverAppBarHome(),
    );
}
}

class SliverAppBarHome extends StatelessWidget {
const SliverAppBarHome({Key? key}) : super(key: key);

@override
Widget build(BuildContext context) {
    return Scaffold(
    body: CustomScrollView(
        slivers: [
        SliverAppBar(
            expandedHeight: 200.0,
            floating: false,
            pinned: true,
            flexibleSpace: FlexibleSpaceBar(
            centerTitle: true,
            title: Text(
                'It will collapse',
                style: TextStyle(
                color: Colors.white,
                fontSize: 16.0,
                ),
            ),
            background: Image.network(
                'https://cdn.pixabay.com/photo/2016/09/10\
/17/18/book-1659717_960_720.jpg',
                fit: BoxFit.cover,
            ),
            ),
        ),
        SliverFixedExtentList(
            itemExtent: 50,
            delegate: SliverChildListDelegate([
            Container(color: Colors.red),
            Container(color: Colors.green),
            Container(color: Colors.blue),
            Container(color: Colors.red),
            Container(color: Colors.green),
            Container(color: Colors.blue),
            Container(color: Colors.red),
            Container(color: Colors.green),
            Container(color: Colors.blue),
            Container(color: Colors.red),
            Container(color: Colors.green),
            Container(color: Colors.blue),
            ]),
        ),
        ],
    ),
    );
}
}
```

The Custom scroll view has the slivers parameter that returns a list of widgets.

```
CustomScrollView(
        slivers: [
        SliverAppBar(
            expandedHeight: 200.0,
            floating: false,
            pinned: true,
            flexibleSpace: FlexibleSpaceBar(
...
```

As a result we have added a sliver fixed extent list widget and used a combination of colors.

In our next tutorial we'll show how we this special collapsible AppBar can act as a toolbar or any other widgets, such as a TabBar and a FlexibleSpaceBar.

How to use AppBar Toolbars in flutter

An AppBar is the integral part of Material Design principles in Flutter with many features.

We've been discussing layout and designs in Flutter, and we've learned a couple of design principles so far. AppBar is an integral part of Material Design, and Flutter layout. Although we had a gentle introduction to this topic, still we feel something more we should learn regarding AppBar.

On Android, as an integral part of material design principles, toolbars represent AppBar.

To use AppBar we firstly need another widget – Scaffold.

Now once included, the AppBar widget consists of many toolbars. As a result, we can use our AppBar in many ways. We can navigate to another page, we can use search icon to search our app, and many more.

We'll see those features in a minute.

Consisting of a toolbar and other widgets as well, an AppBar always rests on the top of the Flutter application. Since, it rests on the top, we must make it look great. Moreover, we need to add as many functionalities as possible.

That may include TabBar and a FlexibleSpaceBar, which we're going to discuss in a moment.

The most important role of AppBar is it exposes one or more common actions with IconButtons.

How does it look like? Let's take a look at the screenshot below.

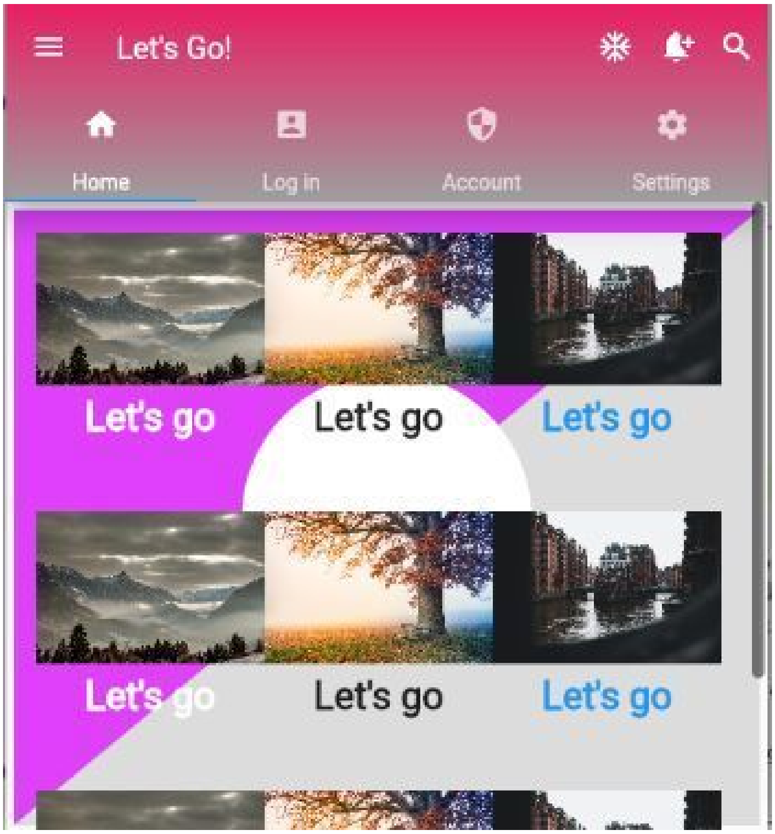


Figure 2.4 – Flutter AppBar using toolbars and other features

The AppBar, displayed above, shows us many things.

Firstly, we've used a color gradient as its background, that matches with our Application's background color combination.

Secondly, with the help of toolbars, and actions, and IconButtons we've been able to add more functionalities.

What is the use of AppBar?

Let's take a look at the code first. And after that, we can discuss the code and try to understand how an AppBar may affect our whole Flutter application's design and layout.

In our model folder we have defined a CustomPaint object that decides how our flutter application will exactly look like.

```dart
import 'package:flutter/material.dart';

class ShapingPainter extends CustomPainter {
@override
void paint(Canvas canvas, Size size) {
    final paint = Paint();

    /// setting the paint color greyish
    /// so it could cover the lower half of the screen
    ///
    paint.color = Colors.black12;

    /// Creating a rectangle with size and width same as \
the canvas
    /// It'll be going to cover the whole screen
    ///
    var rect = Rect.fromLTWH(0, 0, size.width, size.heigh\
t);

    /// Drawing the rectangle using the paint
    ///
    canvas.drawRect(rect, paint);

    /// Covering the upper half of the rectangle
    ///
    paint.color = Colors.purpleAccent;
    // Firstly, creating a path to form the shape
    var path = Path();
    path.lineTo(0, size.height);
    path.lineTo(size.width, 0);
    // Secondly, closing the path to form a bounded shape
    path.close();
    canvas.drawPath(path, paint);
    // Setting the color property of the paint
    paint.color = Colors.white;
    // Center of the canvas is (x,y) => (width/2, height/\
2)
    var center = Offset(size.width / 2, size.height / 2);
    // Finally, drawing the circle with center having rad\
ius 95.0
    canvas.drawCircle(center, 95.0, paint);
}

@override
bool shouldRepaint(CustomPainter oldDelegate) => false;
}
```

Next, comes the root app.

```
import 'package:flutter/material.dart';

import 'view/my_app.dart';

void main() => runApp(const MyApp());
The My App widget calls a Dash Board Home page.

import 'package:flutter/material.dart';
import 'dash_board_home.dart';

class MyApp extends StatelessWidget {
const MyApp({Key? key}) : super(key: key);

@override
Widget build(BuildContext context) {
    return MaterialApp(
    debugShowCheckedModeBanner: false,
    title: 'A Custom Home Page',

    /// ignore: todo
    ///TODO: we'll make a custom global theme later
    ///
    theme: ThemeData(
        primarySwatch: Colors.blue,
    ),
    home: DashBoardHome(),
    );
}
}
```

In the Dash Board Home screen we've defined our AppBar with all functionalities.

```
import 'package:flutter/material.dart';
import '/model/shaping_painter.dart';

import 'all_containers.dart';

class DashBoardHome extends StatelessWidget {
const DashBoardHome({
    Key? key,
}) : super(key: key);

@override
Widget build(BuildContext context) {
    var size = MediaQuery.of(context).size;
    return DefaultTabController(
    length: 4,
    child: Scaffold(
        appBar: AppBar(
        //backgroundColor: Colors.grey[400],
        flexibleSpace: Container(
            decoration: const BoxDecoration(
            gradient: LinearGradient(
                colors: [
                Colors.pink,
                Colors.grey,
                ],
                begin: Alignment.topRight,
                end: Alignment.bottomRight,
            ),
            ),
        ),
        elevation: 20,
        leading: const Icon(Icons.menu),
        title: const Text(
            'Let\'s Go!',
            textAlign: TextAlign.center,
        ),
        actions: [
            IconButton(
            onPressed: () {},
            icon: const Icon(Icons.ac_unit),
            ),
            IconButton(
            onPressed: () {},
            icon: const Icon(
                Icons.notification_add,
            ),
            ),
            IconButton(
            onPressed: () {},
            icon: const Icon(
                Icons.search,
            ),
            ),
        ],
        bottom: TabBar(
            tabs: [
            Tab(
                icon: IconButton(
                onPressed: () {},
                icon: const Icon(
                    Icons.home,
                ),
                ),
                text: 'Home',
            ),
            Tab(
                icon: IconButton(
                onPressed: () {},
                icon: const Icon(
                    Icons.account_box,
                ),
                ),
                text: 'Log in',
            ),
            Tab(
                icon: IconButton(
                onPressed: () {},
                icon: const Icon(
                    Icons.security,
                ),
                ),
                text: 'Account',
            ),
            Tab(
                icon: IconButton(
                    onPressed: () {},
                    icon: const Icon(
                        Icons.settings,
                    ),
                    ),
                    text: 'Settings',
                ),
                ],
            ),
            ),
            body: Stack(
            children: <Widget>[
                Container(
                padding: const EdgeInsets.all(5),
                child: CustomPaint(
                    painter: ShapingPainter(),
                    child: Container(
                    height: size.height / 1,
                    ),
                ),
                ),
                ListView(
                children: const [
                    FirstContainer(),
                    SecondContainer(),
                    ThirdContainer()
                ],
                ),
            ],
            ),
        ),
        );
}
}
```

Before we've implied these functionalities in our AppBar, let us see the old AppBar first. So that we can understand how did these changes affect our AppBar as a whole.

Before the change, it looked like the above screenshot.

Now, how did we apply the change?

How do I add icons to AppBar?

It all started with Default Tab Controller widget that we've used to wrap our Scaffold whose child is AppBar.

```
return DefaultTabController(
    length: 4,
    child: Scaffold(
        appBar: AppBar(
...
```

Next we've defined the length of the list of the tabs that we're going to use.

It is 4, so we can use four tabs at the bottom.

```
bottom: TabBar(
            tabs: [
            Tab(
                icon: IconButton(
                onPressed: () {},
                icon: const Icon(
                    Icons.home,
                ),
                ),
                text: 'Home',
            ),
            Tab(
                icon: IconButton(
                onPressed: () {},
                icon: const Icon(
                    Icons.account_box,
                ),
                ),
                text: 'Log in',
            ),
            Tab(
                icon: IconButton(
                onPressed: () {},
                icon: const Icon(
                    Icons.security,
                ),
                ),
                text: 'Account',
            ),
            Tab(
                icon: IconButton(
                onPressed: () {},
                icon: const Icon(
                    Icons.settings,
                ),
                ),
                text: 'Settings',
            ),
            ],
        ),
        ),
...
```

However, any AppBar displays the toolbar widgets, such as leading, title, and actions. Therefore, we’ve used them as well.

```
elevation: 20,
leading: const Icon(Icons.menu),
actions: [
            IconButton(
            onPressed: () {},
            icon: const Icon(Icons.ac_unit),
            ),
...
```

Elevation parameter refers to the shadow at the bottom of our AppBar. We could have made it thinner or thicker.

The leading property decides which Widget will be placed at the very beginning of the AppBar. We've used a menu Icon Widget. We usually keep this place to use for a Drawer widget. For full code please visit the related GitHub repository.

We can even add scrollable AppBar. However, that's a different topic.

In the coming section we'll discuss that. We'll also discuss a special AppBar for scrolling widgets.

For a scrollable app bar, we use SliverAppBar, which embeds an AppBar in a sliver for use in a CustomScrollView.

How to make tab bar view in flutter

The Tab Bar View widget works in conjunction with default tab controller making great designs.

We use TabBarView widget to display the Widget which corresponds to the currently selected tab. It basically follows the principle of PageView and this widget is typically used in conjunction with a TabBar.

In our previous section we've learned how to use various toolbars in an AppBar, so that we can navigate to other pages.

However, we've not taken any concrete steps to make it happen. In this section, we'll learn how we can do that.

In the last section, we've build an AppBar like the following screenshot.

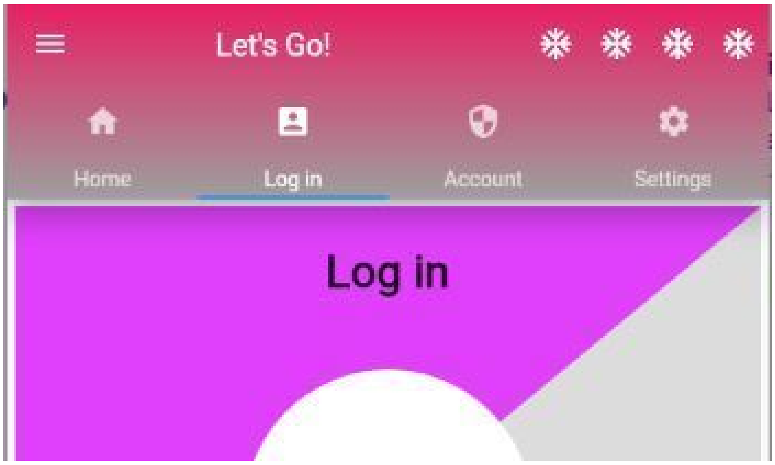


Figure 2.5 – Flutter tab bar view first page

Although we've successfully designed the above AppBar, we couldn't move to any other pages, except the home page. By default, it opens up.

Now, we want to navigate to other pages shown in the tabs, such as Log in, Accounts, or Settings.

Firstly, we've wrapped our main Dash Board Home page with a DefaultTabController.

It is customary that if a TabController is not provided, then there must be a DefaultTabController ancestor.

Why so?

Because the tab controller's TabController.length must equal the length of the children list of the Widgets. Moreover, it should also match the length of the TabBar.tabs list that houses the pages.

Let us see the code, so that we can understand this feature in a better way.

```
import 'package:flutter/material.dart';
import '/model/all_tab_bars.dart';
import '/model/shaping_painter.dart';

import 'all_containers.dart';
import 'all_pages.dart';

class DashBoardHome extends StatelessWidget {
const DashBoardHome({
    Key? key,
}) : super(key: key);

@override
Widget build(BuildContext context) {
    var size = MediaQuery.of(context).size;
    return DefaultTabController(
    length: 4,
    child: Scaffold(
        appBar: AppBar(
        //backgroundColor: Colors.grey[400],
        flexibleSpace: Container(
            decoration: const BoxDecoration(
            gradient: LinearGradient(
                colors: [
                Colors.pink,
                Colors.grey,
                ],
                begin: Alignment.topRight,
                end: Alignment.bottomRight,
            ),
            ),
        ),
        elevation: 20,
        titleSpacing: 80,
        leading: const Icon(Icons.menu),
        title: const Text(
            'Let\'s Go!',
            textAlign: TextAlign.center,
        ),
        actions: [
            buildIcons(
            const Icon(Icons.ac_unit),
            ),
            buildIcons(
            const Icon(
                Icons.notification_add,
            ),
            ),
            buildIcons(
            const Icon(Icons.ac_unit),
            ),
            buildIcons(
            const Icon(Icons.search),
            ),
        ],
        bottom: allTabBars(),
        ),
        body: TabBarView(
        children: [
            FirstPage(size: size),
            SecondPage(size: size),
            ThirdPage(size: size),
            FourthPage(size: size),
        ],
        ),
    ),
    );
}
```

```

IconButton buildIcons(Icon icon) {
    return IconButton(
    onPressed: () {},
    icon: const Icon(Icons.ac_unit),
    );
}
}
```

As we can see, we've provided the DefaultTabController's length property first.

Next, according to the number of length, we return the equal number of list of Widgets in actions property.

```
actions: [
            buildIcons(
            const Icon(Icons.ac_unit),
            ),
            buildIcons(
            const Icon(
                Icons.notification_add,
            ),
            ),
            buildIcons(
            const Icon(Icons.ac_unit),
            ),
            buildIcons(
            const Icon(Icons.search),
            ),
        ],
...
```

After that, we've followed the same rule in case of Tab Bar View pages.

```
body: TabBarView(
        children: [
            FirstPage(size: size),
            SecondPage(size: size),
            ThirdPage(size: size),
            FourthPage(size: size),
        ],
        ),
...
```

We've provided just four pages.

As a result, when we click the Log in, the Log in page opens up.

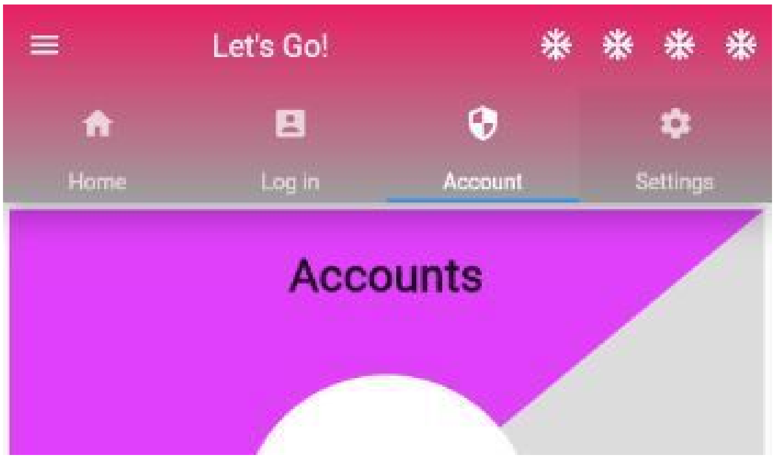


Figure 2.6 – Tab bar view shows second page

When we click the Account page, it opens up.

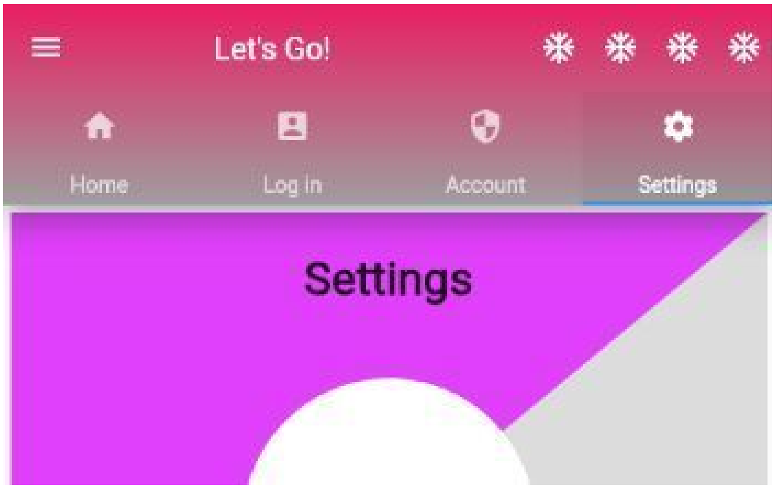


Figure 2.7 – Tab bar view shows third page

And finally, when we click the Settings page, it also opens up. Moreover, when each tab is clicked, it is highlighted by a blue shadow.

To get the full code, please visit the respective GitHub repository.

In this section, we've learned how the tab bar works in Flutter. We use the tabs mainly for mobile navigation.

The styling of tabs may vary according to the operating systems. For example, we can place the tabs at the top of the screen in android devices. On the contrary, we can place them at the bottom in iOS devices.

However, don't assume that we cannot apply the same rule in Android. We can use the tabs at the bottom in Android in a different way.

To sum up, working with tabs is a common pattern in Android and iOS apps.

And, in addition to that, we've already learned that they follow the Material Design principles. Flutter is in advantage, because it provides a convenient way to create a tab layout.

Moreover, you may think app bar as a continuation of material design component.

An app bar consists of a toolbar and other widgets that we’ve just seen. In addition, an app bar exposes many kind of actions with icon buttons.

And the story of multipurpose AppBar is not finished yet. We’ll learn how we can make AppBar transparent in our next section.

How to use TabBar in Flutter

TabBar is guided by default tab controller that keeps tabs and contents in sync.

We use TabBar inside the AppBar in Flutter. In this section, we’ll learn how to set a TabBar inside Flutter AppBar and we’ll also learn how to use TabBar.

In general, when we select a tab, it displays some content or a page. Just like the following screenshot.

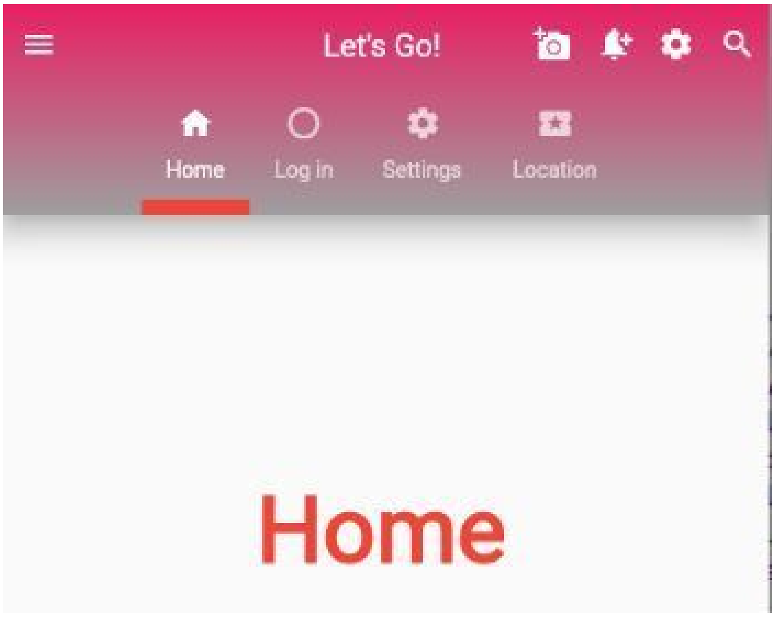


Figure 2.8 – Home tab is selected and displays the Home page

We've created tabs using the TabBar widget. As we can see in the above screenshot, we've created four tabs and each tab will display different page.

Now this page might be not as simple as we've shown in this example. Each page, or screen of any Flutter application should have a unique identity that we're also going to learn as we progress.

Let's try to understand the inside mechanism. After that, we'll see the code.

Firstly, any TabBar needs a TabBarView widget to display different pages.

Although before that, when a tab is selected, the TabBar widget always looks up and tries to find the nearest DefaultTabController.

Basically, the TabController is created by the DefaultTabController. Because of that reason, we wrap the Scaffold widget by DefaultTab-

Controller.

Sitting at the top, the DfaultTabController maintains the synchronization between tab and content. That's why, it asks how many tabs we're going to create. The length property sets the value.

As a result, when we select a tab, only related page is opened up in the body section. We've clicked the Log in tab, therefore the Log in page has popped up.

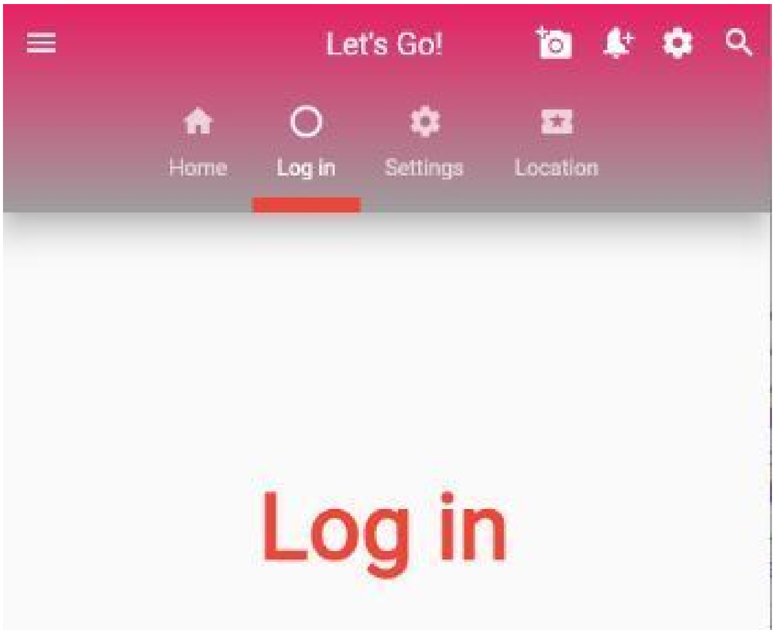


Figure 2.9 – Tab controller keeps tab and content in sync

Let's take a look at the code snippet where we've defined these properties.

```
import 'package:flutter/material.dart';

//import 'all_pages.dart';

/// adding transparent appbar
/// modifying build icons

class DashBoardHome extends StatelessWidget {
const DashBoardHome({
    Key? key,
}) : super(key: key);

@override
Widget build(BuildContext context) {
    //var size = MediaQuery.of(context).size;
    return DefaultTabController(
    length: 4,
    child: Scaffold(
        appBar: AppBar(
        centerTitle: true,

        //backgroundColor: Colors.grey[400],
        flexibleSpace: Container(
            decoration: const BoxDecoration(
            gradient: LinearGradient(
                colors: [
                Colors.pink,
                Colors.grey,
                ],
                begin: Alignment.topRight,
                end: Alignment.bottomRight,
            ),
            ),
        ),
        elevation: 20,
        titleSpacing: 80,
        leading: const Icon(Icons.menu),
        title: const Text(
            'Let\'s Go!',
            textAlign: TextAlign.center,
        ),
        actions: [
            buildIcons(
            const Icon(Icons.add_a_photo),
            ),
            buildIcons(
            const Icon(
                Icons.notification_add,
            ),
            ),
            buildIcons(
            const Icon(
                Icons.settings,
            ),
            ),
            buildIcons(
            const Icon(Icons.search),
            ),
        ],
        bottom: const TabBar(
            isScrollable: true,
            indicatorColor: Colors.red,
            indicatorWeight: 10,
            tabs: [
            Tab(
                icon: Icon(
                Icons.home,
                ),
                text: 'Home',
            ),
            Tab(
                icon: Icon(
                Icons.panorama_fish_eye,
                ),
                text: 'Log in',
            ),
            Tab(
                icon: Icon(
                Icons.settings,
                ),
                text: 'Settings',
            ),
            Tab(
                icon: Icon(
                Icons.local_activity,
                ),
                text: 'Location',
            ),
            ],
        ),
        ),
        body: TabBarView(
        children: [
            newPage('Home'),
            newPage('Log in'),
            newPage('Settings'),
            newPage('Location'),
        ],
        ),
    ),
    );
}

IconButton buildIcons(Icon icon) {
    return IconButton(
    onPressed: () {},
    icon: icon,
    );
}

Widget newPage(String text) => Center(
        child: Text(
        text,
        style: const TextStyle(
            fontSize: 60,
            fontWeight: FontWeight.bold,
            color: Colors.red,
        ),
        ),
    );
}
```

For the full code, please visit the concerned GitHub repository.

The above code is quite simple. We've wrapped the Scaffold widget with default tab controller and set the length to 4. It means, we can keep four tabs inside TabBar and four pages inside TabBarView widget.

Exactly that happens in the above code.

```
bottom: const TabBar(
          isScrollable: true,
          indicatorColor: Colors.red,
          indicatorWeight: 10,
          tabs: [
          Tab(
              icon: Icon(
              Icons.home,
              ),
              text: 'Home',
            ),
            Tab(
                icon: Icon(
                Icons.panorama_fish_eye,
                ),
                text: 'Log in',
            ),
            Tab(
                icon: Icon(
                Icons.settings,
                ),
                text: 'Settings',
            ),
            Tab(
                icon: Icon(
                Icons.local_activity,
                ),
                text: 'Location',
            ),
            ],
        ),
        ),
        body: TabBarView(
        children: [
            newPage('Home'),
            newPage('Log in'),
            newPage('Settings'),
            newPage('Location'),
        ],
        ),
    ),
....
```

Here DefaultTabController has created a Tab Controller and makes it available to all descendant widgets.

As a result, we can set the other properties of TabBar as well.

```
bottom: const TabBar(
                isScrollable: true,
                indicatorColor: Colors.red,
                indicatorWeight: 10,
...
```

We can add more tabs and TabBarView pages. Because we have set the “isScrollable” property true. Moreover, indicator color, and indicator height to a certain value. Consequently, when we click the tab, it’s highlighted.

The job of DefaultTabController is to share a TabController with a TabBar or a TabBarView. No exception takes place here. It does its job, and the tab and content are in complete sync.

Last but not least. One caveat before we close down this topic.

Since TabBar works at tandem with TabBarView widget, and pages – to be shown – are maintained by TabBarView, we need not use Icon Button widget inside the TabBar. That’ll throw a render overflow error.

Why?

The reason is quite logical. The TabBar here just shows the icons. It does not allow those icons to pass a callback.

How to make the AppBar transparent

With a simple tweak in AppBar, we can make it completely or partially transparent.

To make the AppBar transparent is quite easy. All we need to do is tweak the back ground color property of the AppBar.

Now, question is what is the role of back ground color in an AppBar?

It basically fills the fill color to use for an app bar's Material.

If we don't supply any value, it uses the AppBarTheme.backgroundColor, which is blue by default.

That's the reason, when we create a fresh Flutter Application, it comes up with a blue AppBar.

To make it happen, the AppBar also uses the theme's ColorScheme.primary, if the overall theme's brightness is Brightness.light.

However, if the theme's brightness is Brightness.dark, then it uses the ColorScheme.surface.

When we want to make the AppBar scrollable, we need to overlaps the AppBar.

And to do that, we need to use the transparent property, or the withOpacity property of the color class.

Let's try the transparent property and see the effect first.

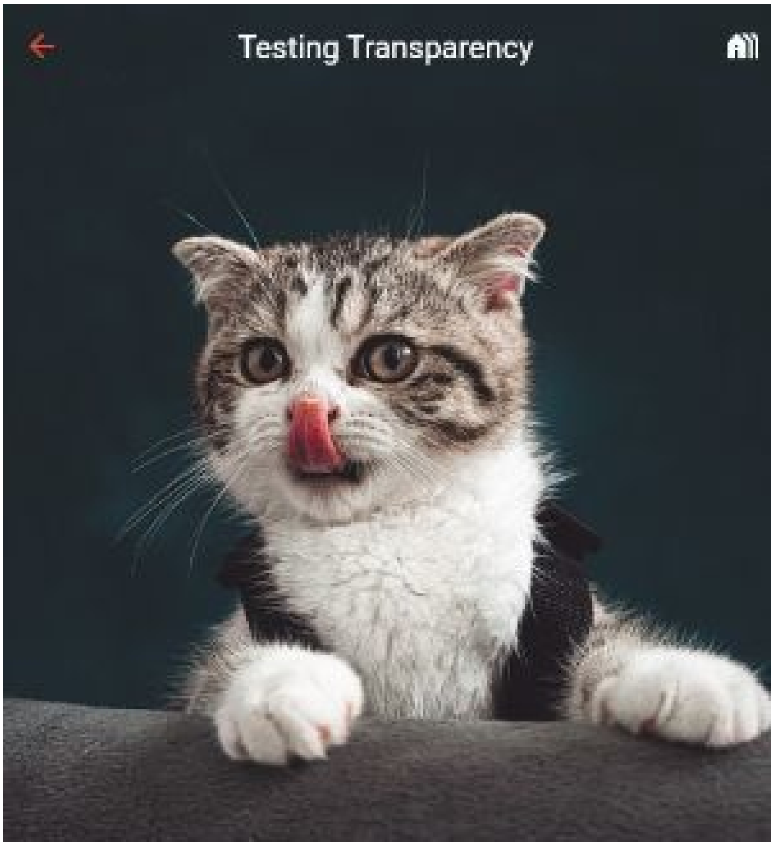


Figure 2.10 – Background color transparent

As a result, the image’s color overlaps the AppBar completely.

However, in case of opacity, we can control how much opacity we can use.

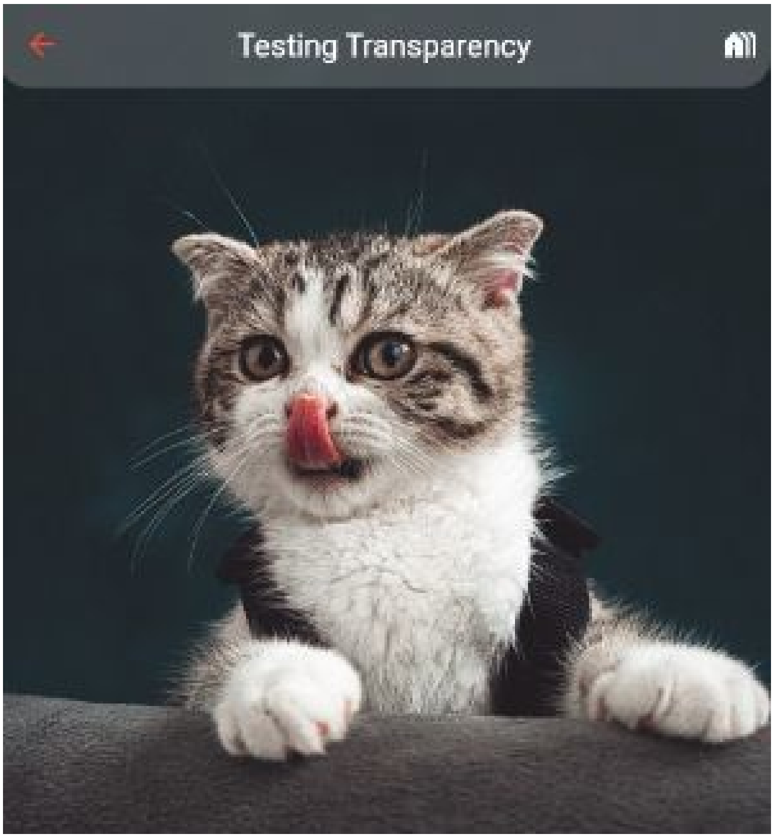


Figure 2.11 – Opacity makes AppBar partially transparent

Finally, the time has come to see the full code where we've defined such properties.

```
import 'package:flutter/material.dart';
import '/model/all_tab_bars.dart';

//import 'all_pages.dart';

/// adding transparent appbar
/// modifying build icons

class DashBoardHome extends StatelessWidget {
const DashBoardHome({
    Key? key,
}) : super(key: key);

@override
Widget build(BuildContext context) {
    //var size = MediaQuery.of(context).size;
    return DefaultTabController(
    length: 4,
    child: Scaffold(
        extendBodyBehindAppBar: true,
        appBar: AppBar(
        title: const Text('Testing Transparency'),
        centerTitle: true,
        leading: const BackButton(
            color: Colors.red,
        ),
        actions: [
            IconButton(
            onPressed: () {},
            icon: const Icon(Icons.holiday_village),
            )
        ],
        shape: const RoundedRectangleBorder(
            borderRadius: BorderRadius.vertical(
            bottom: Radius.circular(20),
            ),
        ),
        //backgroundColor: Colors.transparent,
        backgroundColor: Colors.white.withOpacity(0.19),
        elevation: 0,
        ),
        body: Image.network(
        'https://cdn.pixabay.com/photo/2021/10/19/10/56/c\
at-6723256_960_720.jpg',
        fit: BoxFit.cover,
        width: double.infinity,
        height: double.infinity,
        ),
    ),
    );
}

IconButton buildIcons(Icon icon) {
    return IconButton(
    onPressed: () {},
    icon: icon,
    );
}
}
```

In the above code, only this line helps us to get the effect.

```
//backgroundColor: Colors.transparent,
backgroundColor: Colors.white.withOpacity(0.19),
```

Use any one of them, and make your AppBar transparent.

For the full code snippet please visit the respective GitHub repository mentioned below.

- To check all related code snippets, please visit these GitHub branches[9]
- To check all related code snippets, please visit these GitHub branches[10]
- To check all related code snippets, please visit these GitHub branches[11]
- To check all related code snippets, please visit these GitHub branches[12]
- To check all related code snippets, please visit these GitHub branches[13]

[9]https://github.com/sanjibsinha/flutter_artisan/blob/basic-layout-part-one/lib/main.dart
[10]https://github.com/sanjibsinha/flutter_artisan/tree/layout-part-one
[11]https://github.com/sanjibsinha/flutter_artisan/tree/layout-part-two
[12]https://github.com/sanjibsinha/flutter_artisan/tree/appbar-layout
[13]https://github.com/sanjibsinha/flutter_artisan/tree/appbar-layout-part-one

3. What are responsive and adaptive Flutter Applications

Let us see which layout Widgets help us to build an adaptive and responsive Flutter Application.

How to use Stack in Flutter

Stack contains a list of widgets and positions them on top of the other children widgets.

To use Stack widget in Flutter, firstly, we need to know how it works. Secondly, we'll learn how we can use Stack widget to build and design an adaptive and responsive Flutter application that runs on web and mobile platform at the same time.

Stack widget is one of the main layout widgets that we'll need to use very often to build any kind of Flutter application.

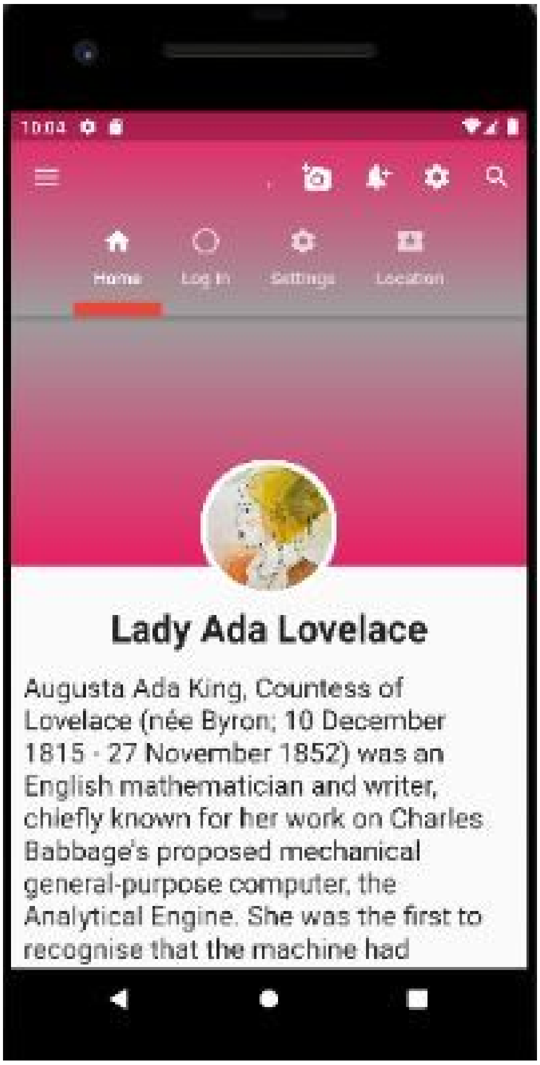
Stack contains a list of widgets and positions them on top of the other. That means, the Stack allows us to overlap multiple widgets on a single screen. In other words, the first child of Stack is the bottom-most widget. And the last child is the top-most widget.

The first question that comes to our mind is, according to the Stack mechanism, the top-most widget should only be visible, as it sits on top of the other widgets and completely overlaps others.

Then, how we would use Stack, so every child widget will not only be visible, but also be positioned to build a nice-looking layout.

Therefore, the Stack either wrap its children widgets in a Positioned widget, or may place them in a Non-Positioned way.

We'll see how we can use Stack to build the following design.



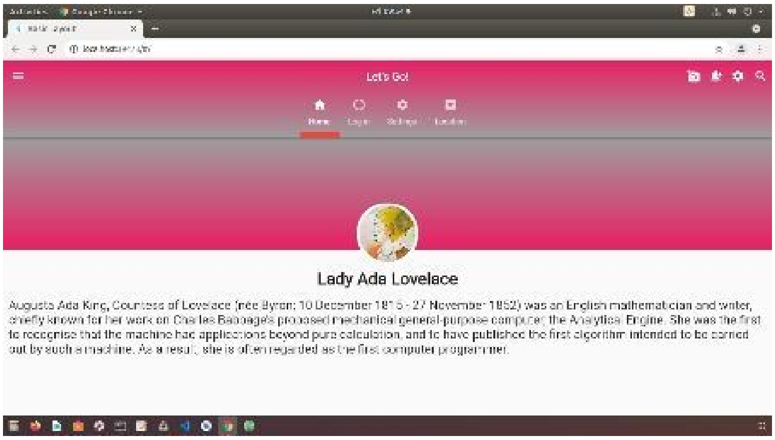
Lady Ada Lovelace

Augusta Ada King, Countess of Lovelace (née Byron; 10 December 1815 - 27 November 1852) was an English mathematician and writer, chiefly known for her work on Charles Babbage's proposed mechanical general-purpose computer, the Analytical Engine. She was the first to recognise that the machine had

Figure 3.1 – Stack widget use positioned widgets

We've built the above design using Stack widget that again positions its children widget in a way, so that the image and text are placed properly.

Not only that the same design remains adaptive and responsive when we run the same flutter application in web platform.



Lady Ada Lovelace

Augusta Ada King, Countess of Lovelace (née Byron; 10 December 1815 - 27 November 1852) was an English mathematician and writer, chiefly known for her work on Charles Babbage's proposed mechanical general-purpose computer, the Analytical Engine. She was the first to recognise that the machine had applications beyond pure calculation, and to have published the first algorithm intended to be carried out by such a machine. As a result, she is often regarded as the first computer programmer.

Figure 3.2 – Stack widget maintains adaptive and responsive nature

Let's see the first part of code.

```dart
import 'package:flutter/material.dart';

//import 'view/my_app.dart';

void main() {
runApp(const MyApp());
}

class MyApp extends StatelessWidget {
const MyApp({Key? key}) : super(key: key);

static const String title = 'Basic Layout';

@override
Widget build(BuildContext context) {
    return const MaterialApp(
    debugShowCheckedModeBanner: false,
    title: title,
    home: MyAppHome(),
    );
}
}
```

Now, we’re going to design the above MyAppHome() widget that shows the profile page as its home page.

To do that, let’s first design the AppBar first.

```
class MyAppHome extends StatelessWidget {
const MyAppHome({Key? key}) : super(key: key);

static const String userUrl =
    'https://cdn.pixabay.com/photo/2019/05/04/15/24/art-4\
178302_960_720.jpg';

@override
Widget build(BuildContext context) {
    return DefaultTabController(
    length: 4,
    child: Scaffold(
        appBar: AppBar(
            centerTitle: true,
            flexibleSpace: Container(
            decoration: const BoxDecoration(
                gradient: LinearGradient(
                colors: [
                    Colors.pink,
                    Colors.grey,
                ],
                begin: Alignment.topRight,
                end: Alignment.bottomRight,
                ),
            ),
            ),
            titleSpacing: 80,
            leading: const Icon(Icons.menu),
            title: const Text(
            'Let\'s Go!',
            textAlign: TextAlign.center,
            ),
            actions: [
            buildIcons(
                const Icon(Icons.add_a_photo),
            ),
            buildIcons(
                const Icon(
                Icons.notification_add,
                ),
            ),
            buildIcons(
                const Icon(
                Icons.settings,
                ),
            ),
            buildIcons(
                const Icon(Icons.search),
            ),
            ],
            bottom: const TabBar(
            isScrollable: true,
            indicatorColor: Colors.red,
            indicatorWeight: 10,
            tabs: [
                Tab(
                icon: Icon(
                    Icons.home,
                ),
                text: 'Home',
                    ),
                    Tab(
                    icon: Icon(
                        Icons.panorama_fish_eye,
                    ),
                    text: 'Log in',
                    ),
                    Tab(
                    icon: Icon(
                        Icons.settings,
                    ),
                    text: 'Settings',
                    ),
                    Tab(
                    icon: Icon(
                        Icons.local_activity,
                    ),
                    text: 'Location',
                    ),
                ],
                ),
            ),
...
```

We've wrapped the Scaffold widget with DefaultTabController widget that has a required parameter length which decides how many tabs we should use in TabBar widget. Moreover, according to that length mentioned we need to return the same number of pages in TabBarView widget in the body parameter.

Firstly, any TabBar needs a TabBarView widget to display different pages.

Although before that, when a tab is selected, the TabBar widget always looks up and tries to find the nearest DefaultTabController.

Basically, the TabController is created by the DefaultTabController.

Because of that reason, we wrap the Scaffold widget by DefaultTab-Controller.

Let us take a look at the body parameter.

What is Stack-positioned

```
...
body: TabBarView(children: [
            ListView(
            children: [
                Stack(
                clipBehavior: Clip.none,
                children: [
                    Container(
                    height: 200,
                    decoration: const BoxDecoration(
                        gradient: LinearGradient(
                        colors: [
                            Colors.pink,
                            Colors.grey,
                        ],
                        begin: Alignment.bottomRight,
                        end: Alignment.topRight,
                        ),
                    ),
                    ),
                    Positioned(
                    bottom: -20,
                    left: 0,
                    right: 0,
                    child: Center(
                        child: Container(
                        width: 100,
                        height: 100,
                        decoration: BoxDecoration(
                            borderRadius: BorderRadius.ci\
rcular(50),
                            boxShadow: const [
                            BoxShadow(
                                color: Colors.white,
                                spreadRadius: 4,
                            ),
                            ],
                            image: const DecorationImage(
                            fit: BoxFit.cover,
                            image: NetworkImage(userUrl),
                            ),
                        ),
                        ),
                    ),
                    ),
                    Positioned(
                    bottom: -150,
                    left: 0,
                    right: 0,
                    child: Center(
                        child: Container(
                        margin: const EdgeInsets.all(17),
                        width: 300,
                        height: 100,
                        child: const Text(
                            'Lady Ada Lovelace',
                            textAlign: TextAlign.center,
                            style: TextStyle(
                            fontSize: 30,
                            fontWeight: FontWeight.bold,
                            ),
                        ),
                        ),
                    ),
                    ),
                ],
                ),
/// coming out of Stack
///
                Container(
                margin: const EdgeInsets.all(10),
                padding: const EdgeInsets.only(top: 75),
                child: const Text(
                    'Augusta Ada King, Countess of Lovela\
ce (née Byron; 10 December 1815 - 27 November 1852) '
                    'was an English mathematician and wri\
ter, chiefly known for her work on Charles Babbage\'s '
                    'proposed mechanical general-purpose \
computer, the Analytical Engine. She was the '
                    'first to recognise that the machine \
had applications beyond pure calculation, and '
                    'to have published the first algorith\
m intended to be carried out by such a machine. '
                    'As a result, she is often regarded a\
s the first computer programmer.',
                    textAlign: TextAlign.left,
                    style: TextStyle(
                    fontSize: 22,
                    ),
                ),
                ),
            ],
            ),
            newPage('Log in'),
            newPage('Settings'),
            newPage('Location'),
        ])),
    );
```

```
}

IconButton buildIcons(Icon icon) {
    return IconButton(
    onPressed: () {},
    icon: icon,
    );
}
```

In the body part, we've wrapped the whole widget tree with ListView widget.

Because we want to scroll if necessary, here a scrolling widget might help us doing so.

Next, we have used the Stack widget, and used three Container widgets as the children of the Stack and decorate them accordingly.

The first Container is the bottom-most, and the third Container is the top-most.

Since the first Container is the bottom-most, we have made it non-positioned.

However, the second and the third Containers are Positioned.

And to distinguish them with each other, we've used a property, bottom; and, after that we've provided a negative value, so that it shifts towards the negative side of Y axis.

The Second Container belonging to the Stack shifts 20 pixel down towards the negative side of Y axis.

Stack-Positioned-bottom

```
Positioned(
                    bottom: -20,
                    left: 0,
                    right: 0,
                    child: Center(
                        child: Container(
...
```

And the Third Container belonging to the Stack shifts 150 pixel down towards the negative side of Y axis.

```
Positioned(
                    bottom: -150,
                    left: 0,
                    right: 0,
                    child: Center(
                        child: Container(
...
```

As the the child widget in a stack can be either positioned or non-positioned, we need to be careful to use them so that they are placed properly.

We always wrap the positioned items with Positioned widget and the must have a one non-null property.

To get the full code please visit the respective GitHub repository.

Stack-Alignment

We provide the value and according to which the Stack widget adjusts its size. Moreover, we can also adjust the position of non-positioned child widgets with the help of alignment which by default positions to the top-left corner in left-to-right environments and the top-right corner in right-to-left environments.

We can learn more about the stack layout algorithm, in Render-Stack.

The full code repository for this section, please check the respective branches[14]

[14]https://github.com/sanjibsinha/flutter_artisan/tree/basic-layout-part-one

4. How to build a Quiz App

List in Flutter is a collection of items. It is the most common collection where we keep ordered objects.

A common list looks quite simple.

var list = [1, 2, 3];

However, when a List collects a combination of list and map inside, it might look complicated.

Consider the following list.

var questions = [{ ‘question’: ‘Who are you?’, ‘answer’: [‘Robot’, ‘Human’, ‘Alien’,], }, { ‘question’: ‘What is your name?’, ‘answer’: [‘Robu’, ‘Honu’, ‘Alu’,], }, { ‘question’: ‘What do you eat?’, ‘answer’: [‘Electricity’, ‘Everything’, ‘Water of Mars’,], }, { ‘question’: ‘What do you want?’, ‘answer’: [‘Follow the instruction’, ‘Go to war and destroy.’, ‘Go back to Mars.’,], },];

In the above code, a list contains a map. A map is an object that associates keys and values.

However, both keys and values can be any type of object. Exactly that happens here.

The above list has four maps. Each map again has two key-value pair. The first key-value pair are both String.

But the second key-value pair is String and List<String>.

In our previous discussion, we have two separate lists of items. On the contrary, here, we want to relate question and answer in one single list.

How do you Map a list in flutter?

Now, each question might have several answers. For a test case, we will learn how we can map this list in Flutter.

Let us consider a simple code like the following one.

import ‘package:flutter/material.dart’;

import ‘question.dart’; import ‘answer.dart’;

main() { runApp(const QuizApp());

}

class QuizApp extends StatelessWidget { const QuizApp({Key? key}) : super(key: key);

```
@override
Widget build(BuildContext context) {
  return const MaterialApp(
    title: 'Playxis - Play + Lexis',
    home: QuizPage(),
  );
}
```

}

class QuizPage extends StatefulWidget { const QuizPage({Key? key}) : super(key: key);

```
@override
State<QuizPage> createState() => _QuizPageState();
```

}

class _QuizPageState extends State<QuizPage> { int index = 0; void increment() { setState(() { index = index + 1; }); if (index == questions.length) { index = 0; } }

```
var questions = [
  {
    'question': 'Who are you?',
    'answer': [
      'Robot',
      'Human',
      'Alien',
    ],
  },
  {
    'question': 'What is your name?',
    'answer': [
      'Robu',
      'Honu',
      'Alu',
    ],
  },
  {
    'question': 'What do you eat?',
    'answer': [
      'Electricity',
      'Everything',
      'Water of Mars',
    ],
  },
  {
    'question': 'What do you want?',
    'answer': [
      'Follow the instruction',
      'Go to war and destroy.',
      'Go back to Mars.',
    ],
  },
];

```

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text(
        'Playxis - Play + Lexis',
      ),
    ),
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        mainAxisSize: MainAxisSize.min,
        children: [
          Text('${questions[index]['question']}'),
          const SizedBox(
            height: 10.0,
          ),
          Container(
            width: double.infinity,
            margin: const EdgeInsets.all(10.0),
            child: ElevatedButton(
              onPressed: increment,
              child: const Text('Answer 1'),
            ),
          ),
          Container(
            width: double.infinity,
            margin: const EdgeInsets.all(10.0),
            child: ElevatedButton(
              onPressed: increment,
              child: const Text('Answer 2'),
            ),
          ),
          Container(
            width: double.infinity,
            margin: const EdgeInsets.all(10.0),
            child: ElevatedButton(
              onPressed: increment,
              child: const Text('Answer 3'),
            ),
          ),
        ],
      ),
    ),
  );
}
```

}

If we run this code, it looks very simple.

We have hard coded the question and answer. As a result, if we run the App, we see the following output.

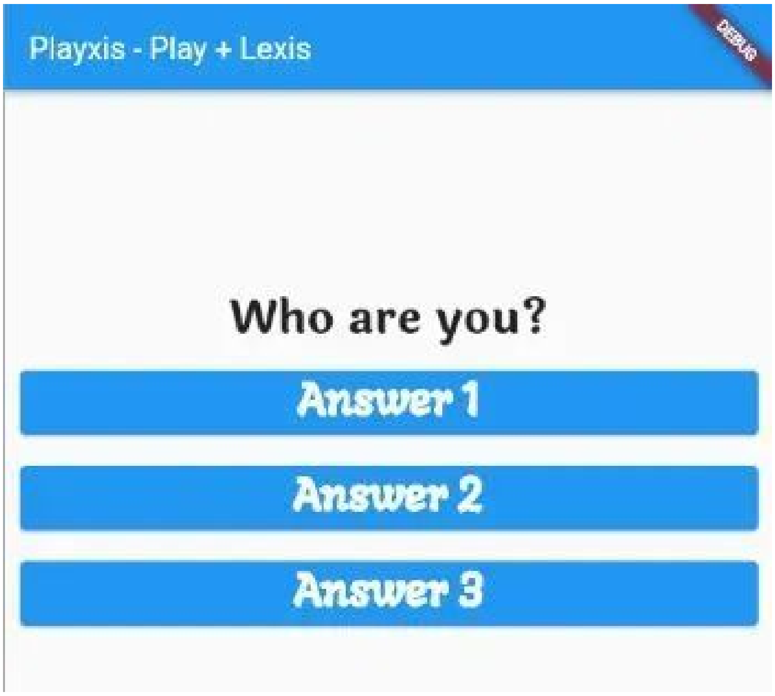


Figure 4.1 – Hard coding a List in Flutter

At the same time, taking a close look at the code tells us to refactor the code. We have unnecessarily repeated ourselves.

Instead, we can extract two separate custom Widgets. One for the Question, and the other for the Answer.

The Question Widget looks like the following.

import 'package:flutter/material.dart'; import 'package:google_-fonts/google_fonts.dart';

class Question extends StatelessWidget {
const Question({Key? key, required this.questions, required this.index}) : super(key: key); final List<Map<String, Object>> questions; final int index;

```
@override
Widget build(BuildContext context) {
  return Text(
    '${questions[index]['question']}',
    style: GoogleFonts.laila(
      textStyle: const TextStyle(
        fontSize: 30.0,
        fontWeight: FontWeight.bold,
      ),
    ),
  );
}
```

}

We will pass two final variables. One is the List of questions which is a List<Map<String, Object>>.

Actually, the data type of List is Map. And again, the Data type of Map is String and List. However, Dart infers the List inside Map as an Object.

After all, everything in Dart is Object.

On the contrary, the Answer Widget needs a final String variable which will display the answer of the List.

And, besides, it also needs a VoidCallback function.

Why?

Because we want to press the ElevatedButton to change the question.

Subsequently, with the change of the question, the answers will also change.

import ‘package:flutter/material.dart’; import ‘package:google_-fonts/google_fonts.dart’;

class Answer extends StatelessWidget { const Answer({ Key? key, required this.answer, required this.pointToOnPress, }) : super(key: key);

```
final String answer;
final VoidCallback pointToOnPress;

@override
Widget build(BuildContext context) {
  return Container(
    width: double.infinity,
    margin: const EdgeInsets.all(10.0),
    child: ElevatedButton(
      onPressed: pointToOnPress,
      child: Text(
        answer,
        style: GoogleFonts.langar(
          textStyle: const TextStyle(
            fontSize: 30.0,
            fontWeight: FontWeight.bold,
          ),
        ),
      ),
    ),
  );
}
```

}

Now we can change the top-level main() function.

We can call the Widgets inside the Column Widgets.

import ‘package:flutter/material.dart’;

import ‘question.dart’; import ‘answer.dart’;

main() { runApp(const QuizApp());

}

class QuizApp extends StatelessWidget { const QuizApp({Key? key}) : super(key: key);

```
@override
Widget build(BuildContext context) {
  return const MaterialApp(
    title: 'Playxis - Play + Lexis',
    home: QuizPage(),
  );
}
```

}

class QuizPage extends StatefulWidget { const QuizPage({Key? key}) : super(key: key);

```
@override
State<QuizPage> createState() => _QuizPageState();
```

}

class _QuizPageState extends State<QuizPage> { int index = 0; void increment() { setState(() { index = index + 1; }); if (index == questions.length) { index = 0; } }

```
var questions = [
  {
    'question': 'Who are you?',
    'answer': [
      'Robot',
      'Human',
      'Alien',
    ],
  },
  {
    'question': 'What is your name?',
    'answer': [
      'Robu',
      'Honu',
      'Alu',
    ],
  },
  {
    'question': 'What do you eat?',
    'answer': [
      'Electricity',
      'Everything',
      'Water of Mars',
    ],
  },
  {
    'question': 'What do you want?',
    'answer': [
      'Follow the instruction',
      'Go to war and destroy.',
      'Go back to Mars.',
    ],
  },
];

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text(
        'Playxis - Play + Lexis',
      ),
    ),
    body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            mainAxisSize: MainAxisSize.min,
            children: [
              Question(questions: questions, index: index),
              Answer(answer: 'Answer 1', pointToOnPress: incr\
ement),
              Answer(answer: 'Answer 2', pointToOnPress: incr\
ement),
              Answer(answer: 'Answer 3', pointToOnPress: incr\
ement),
            ],
          ),
        ),
    );
}
```

}

Now, as we press the Button, each time the question will change.

It happens because the index number changes with the press of the button.

Question(questions: questions, index: index),

Inside the Question Widget we have accessed each question the following way.

'${questions[index]['question']}',

And the above code makes the sense. We can access any List this way.

However, the answers do not change. It remains the same. But we want to access the answers the same way as we have accessed the questions.

To do that, we need to map the list.

How do I get a List of Maps in Dart?

The map method of any list returns one value that we again convert to list.

Especially in Flutter, the Column Widget‘s children property returns a List of Widgets.

Consequently, when we run the code, we see that each question comes with multiple answers associated with it.

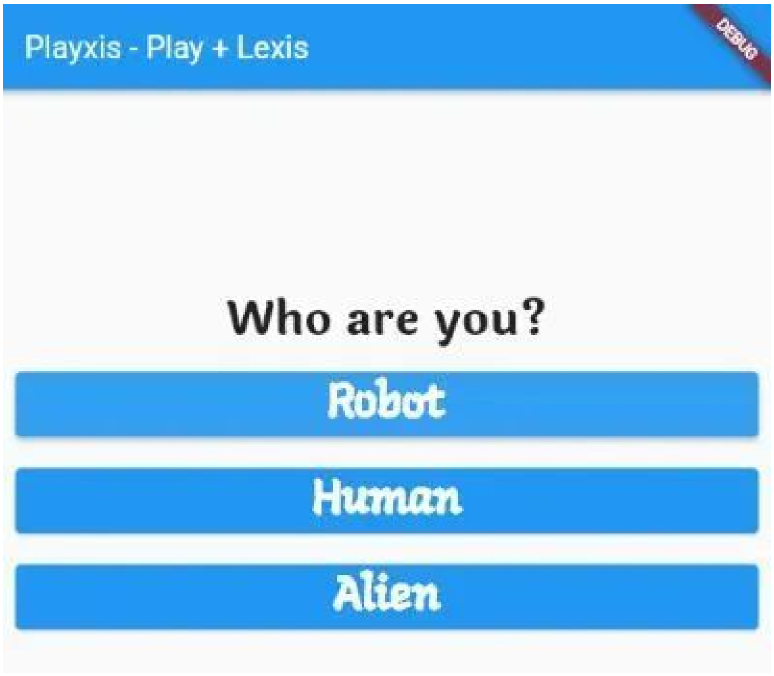


Figure 4.2 – Map a List in Flutter first example

However, we need to use the spread operators to add two lists and make them one

Because the Column Widget’s children property returns one List of Widgets. Right?

Therefore, the final code looks like the following.

import 'package:flutter/material.dart';

import 'question.dart'; import 'answer.dart';

main() { runApp(const QuizApp());

}

class QuizApp extends StatelessWidget { const QuizApp({Key? key}) : super(key: key);

```
@override
Widget build(BuildContext context) {
  return const MaterialApp(
    title: 'Playxis - Play + Lexis',
    home: QuizPage(),
  );
}
```

}

class QuizPage extends StatefulWidget { const QuizPage({Key? key}) : super(key: key);

```
@override
State<QuizPage> createState() => _QuizPageState();
```

}

class _QuizPageState extends State<QuizPage> { int index = 0; void increment() { setState(() { index = index + 1; }); if (index == questions.length) { index = 0; } }

```
var questions = [
  {
    'question': 'Who are you?',
    'answer': [
      'Robot',
      'Human',
      'Alien',
    ],
  },
  {
    'question': 'What is your name?',
    'answer': [
      'Robu',
      'Honu',
      'Alu',
    ],
  },
  {
    'question': 'What do you eat?',
    'answer': [
      'Electricity',
      'Everything',
      'Water of Mars',
    ],
  },
  {
    'question': 'What do you want?',
    'answer': [
      'Follow the instruction',
      'Go to war and destroy.',
      'Go back to Mars.',
    ],
  },
];

```

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text(
        'Playxis - Play + Lexis',
      ),
    ),
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        mainAxisSize: MainAxisSize.min,
        children: [
          Question(questions: questions, index: index),
          ...(questions[index]['answer'] as List<String>)\
.map((answer) {
            return Answer(answer: answer, pointToOnPress:\
 increment);
          }).toList(),
        ],
      ),
    ),
  );
}

}
```

The Column Widget's children property now adds two lists in one list. And the questions and answers are not separated anymore.

With the change of one question now we can get the associated answer.

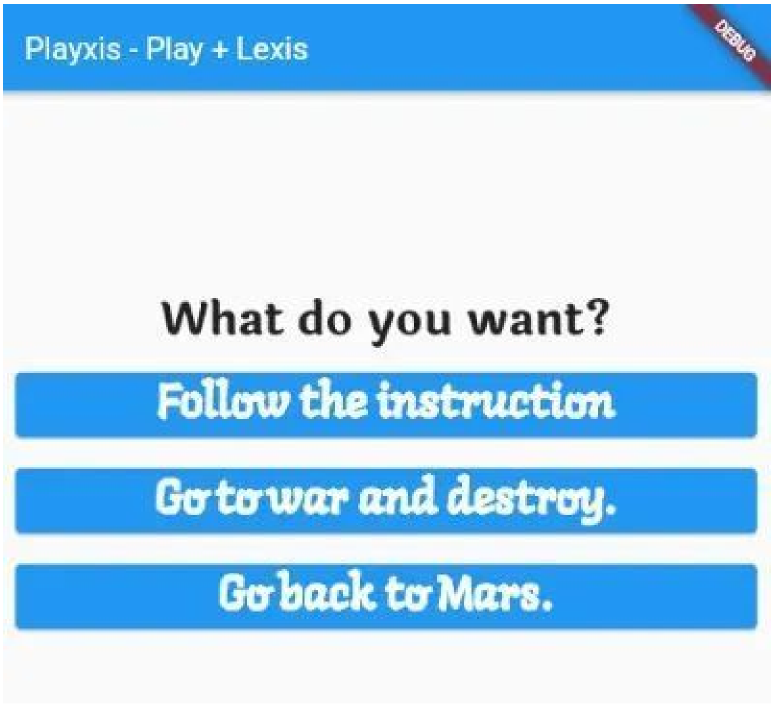


Figure 4.3 – Map a List in Flutter second example

We have solved the hardest part of the Quiz App that we are finally going to build.

In the next section we will build a "Play with Lexis" App that will help us to practice the uncommon vocabulary in English.

However, if you want to clone this preparatory App building process, please clone the related GitHub repository.

How to change theme of an App?

If we do not provide any particular theme, Flutter uses a default theme. And that default theme is used across the entire Flutter App.

What do we see when we create a new Flutter App?

flutter create my_app

By default Flutter comes up with a skeleton code where a default theme has been provided.

As a result, the initial code looks like the following.

void main() { runApp(const MyApp()); }

class MyApp extends StatelessWidget { const MyApp({Key? key}) : super(key: key);

```
// This widget is the root of your application.
@override
Widget build(BuildContext context) {
  return MaterialApp(
    title: 'Flutter Demo',
    theme: ThemeData(
      // This is the theme of your application.
      //
      // Try running your application with "flutter run".\
 You'll see the
      // application has a blue toolbar. Then, without qu\
itting the app, try
      // changing the primarySwatch below to Colors.green\
 and then invoke
      // "hot reload" (press "r" in the console where you\
 ran "flutter run",
      // or simply save your changes to "hot reload" in a\
 Flutter IDE).
      // Notice that the counter didn't reset back to zer\
o; the application
      // is not restarted.
      primarySwatch: Colors.blue,
    ),
    home: const MyHomePage(title: 'Flutter Demo Home Page\
'),
  );
}
```

}

At the same time, Flutter tells us about the default theme that runs across the entire app. Read the comment carefully. It is marked in Bold. It summarizes everything.

In the above case, Flutter provides a ThemeData to the MaterialApp Widget. We all know that it is Blue.

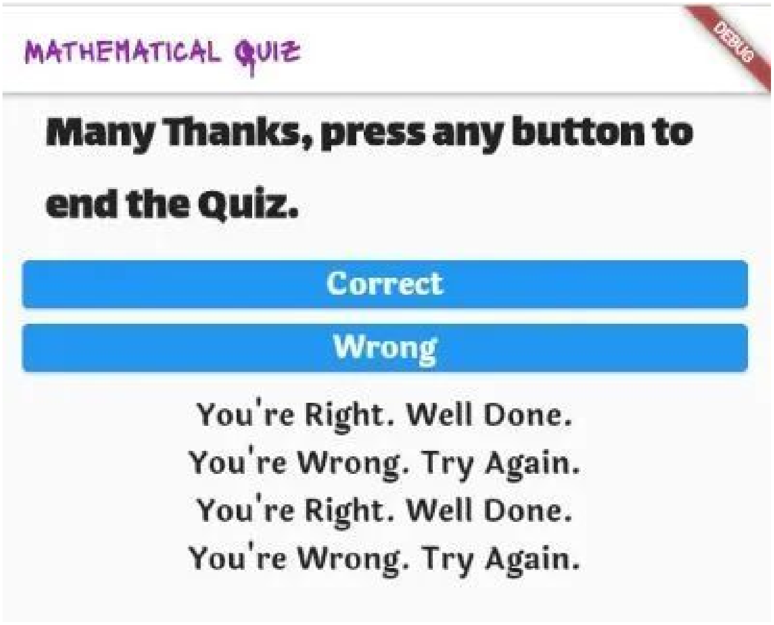
However, we can change this ThemeData.

Consider the Quiz Master App we have built in the last article.

We did not want ThemeData meddling in our affairs. Consequently, the Flutter takes the default theme color Blue as the Button Color. And the background was White.

The Google Font package although changes the Font-look.

Let us take a look at the previous Quiz Master App.



Many Thanks, press any button to end the Quiz.

You're Right. Well Done.
You're Wrong. Try Again.
You're Right. Well Done.
You're Wrong. Try Again.

Figure 4.4 – Flutter list Quiz App third example

Now we can define the configuration of the overall visual Theme for a MaterialApp.

As a result, The Widget sub-tree within the app might take a different Color.

Suppose we want our same Quiz App looks like the following.

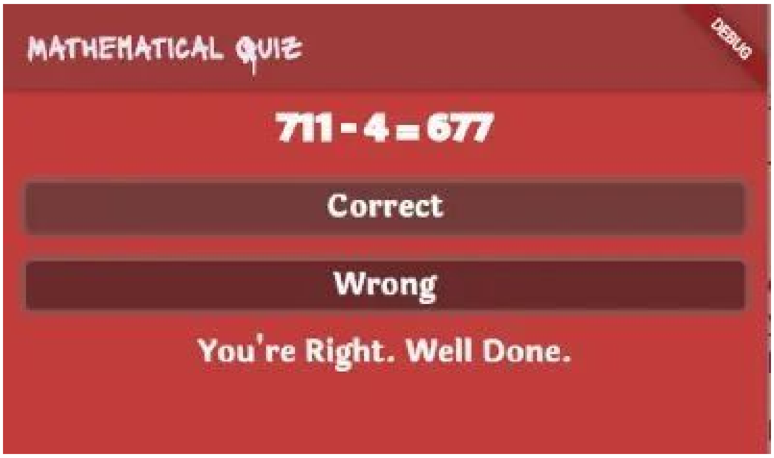


Figure 4.5 – Material Theme Color reddish

To do that, we can had used the ThemeData.dark constructor.

By default, the ThemeData.dark constructor makes the Text-color White. However, we need to provide the background color, the AppBar color, and others.

class QuizApp extends StatelessWidget { const QuizApp({Key? key}) : super(key: key);

```
@override
Widget build(BuildContext context) {
  return MaterialApp(
    theme: ThemeData.dark().copyWith(
      primaryColor: const Color(0xFF8B3817),
      scaffoldBackgroundColor: const Color(0xFFC23C3C),
    ),
    home: Scaffold(
      appBar: AppBar(
        backgroundColor: const Color(0xFF9D3A3A),
```

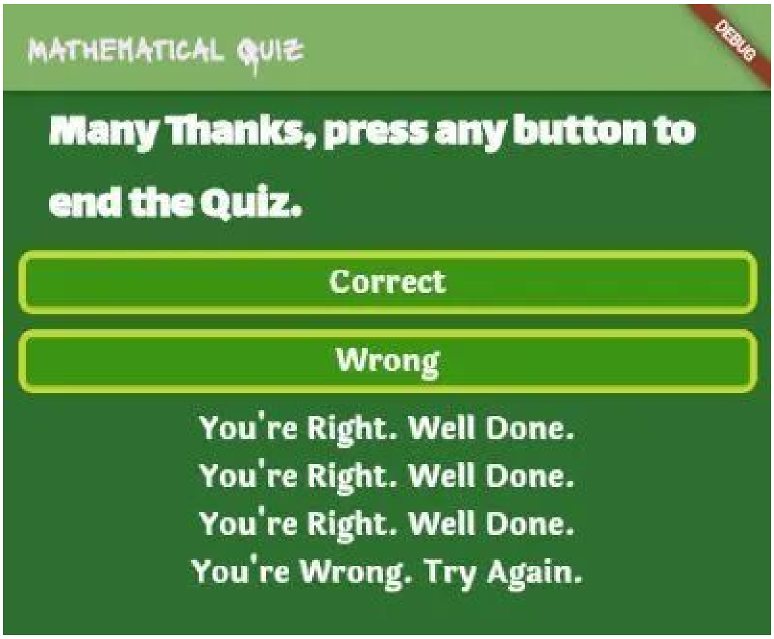
... // the code is incomplete for brevity // to clone the entire project please visit the respective GitHub repository

Besides, to synchronize Button color, we need to change that too.

Container checkingAnswer(String corerctOrWrong, bool trueOrFalse) { return Container(padding: const EdgeInsets.all(5.0), decoration: BoxDecoration(color: const Color(0xFF9B5050), // This color will change the Button color-shade borderRadius: BorderRadius.circular(10.0),), child: ElevatedButton(style: ElevatedButton.styleFrom(primary: const Color(0xFF682a2a), // This color will change the Button color), onPressed: () { checkAnswer(trueOrFalse); }, child: Text(corerctOrWrong, style: GoogleFonts.laila(textStyle: const TextStyle(fontSize: 20.0, fontWeight: FontWeight.bold,),),),),); } // to clone the entire project please visit the respective GitHub repository

As a consequence, we can use the MaterialApp theme property to configure the appearance of the entire app.

Now we can change the theme property from reddish to a greenish tone.



Many Thanks, press any button to end the Quiz.

You're Right. Well Done.
You're Right. Well Done.
You're Right. Well Done.
You're Wrong. Try Again.

Figure 4.6 – Material Theme Color Greenish

To make this happen, we have taken the same path. We have only changed the constant Color constructor.

However, this time from reddish to a greenish tone.

class QuizApp extends StatelessWidget { const QuizApp({Key? key}) : super(key: key);

```
@override
Widget build(BuildContext context) {
  return MaterialApp(
    theme: ThemeData.dark().copyWith(
      primaryColor: const Color(0xFF409B25),
      scaffoldBackgroundColor: const Color(0xFF2C6F2E),
    ),
    home: Scaffold(
      appBar: AppBar(
        backgroundColor: const Color(0xFF81B165),
```

… // the code is incomplete for brevity // to clone the entire project please visit the respective GitHub Repository

And the same way, we have changed the color of the Elevated Button providing a shadow color.

Container checkingAnswer(String corerctOrWrong, bool trueOrFalse) { return Container(padding: const EdgeInsets.all(5.0), decoration: BoxDecoration(color: const Color(0xFFC5DA28), // This changes from reddish to greenish tone borderRadius: BorderRadius.circular(10.0),), child: ElevatedButton(style: ElevatedButton.styleFrom(primary: const Color(0xFF3C9415), // This changes from reddish to greenish tone), onPressed: () { checkAnswer(trueOrFalse); }, child: Text(corerctOrWrong, style: GoogleFonts.laila(textStyle: const TextStyle(fontSize: 20.0, fontWeight: FontWeight.bold,),),),),); } // to clone the entire project please visit the respective GitHub Repository

What is hex color code

Finally, the question arises. How do we pick the hex color code?

Firstly, the hex color code values are a special code that represents color values from 0 to 255. The color Green is represented by this combination: #008000.

Secondly, to make it opaque we always add 0xFF in the place of # tag.

As a result, our Color constructor looks like the following.

primary: const Color(0xFF008000),

We can add the Color Pick eye dropper extension to your Chrome or Firefox Browser.

With the help of that we can pick up the value of any Color-shade.

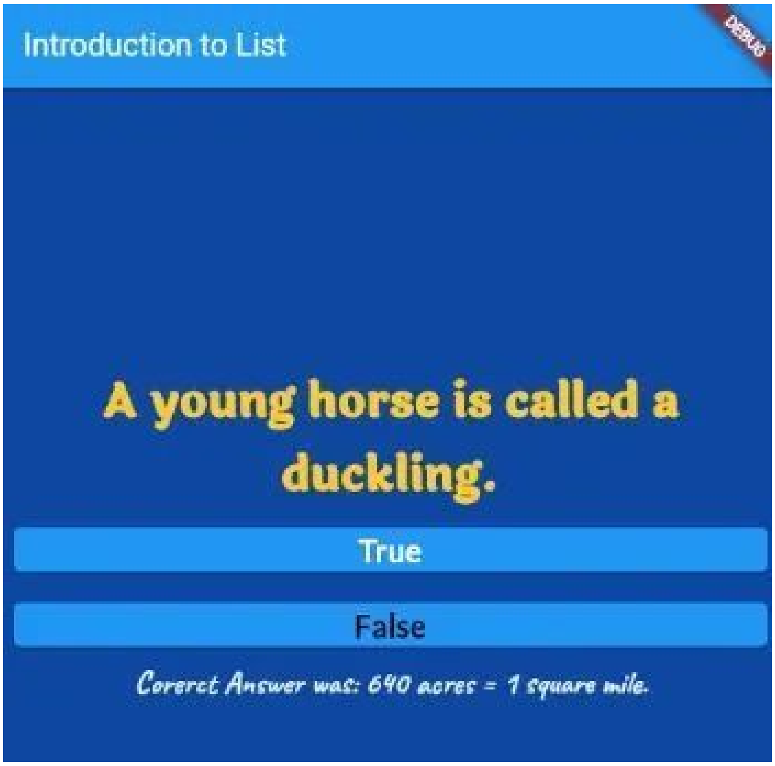
How to use Theme property in the Flutter Quiz App

We can use the Flutter theme property of the MaterialApp Widget, to control the design across the entire app. However, we can do that in various ways.

Previously, we have been building an interesting Quiz App. While building the app, we have learned a few important concepts on theme. We have used a custom theme class where we have declared many static constant Color properties.

And later, we have used those properties in our Flutter app.

Let us see how our previous Quiz App looks like.



A young horse is called a duckling.

Corerct Answer was: 640 acres = 1 square mile.

Figure 4.7 – Custom theme Flutter

As we see, the Flutter app uses a dark theme. And it was greenish.

Now, here is the most important question. Can we make this color scheme light replacing the green by pinkish color?

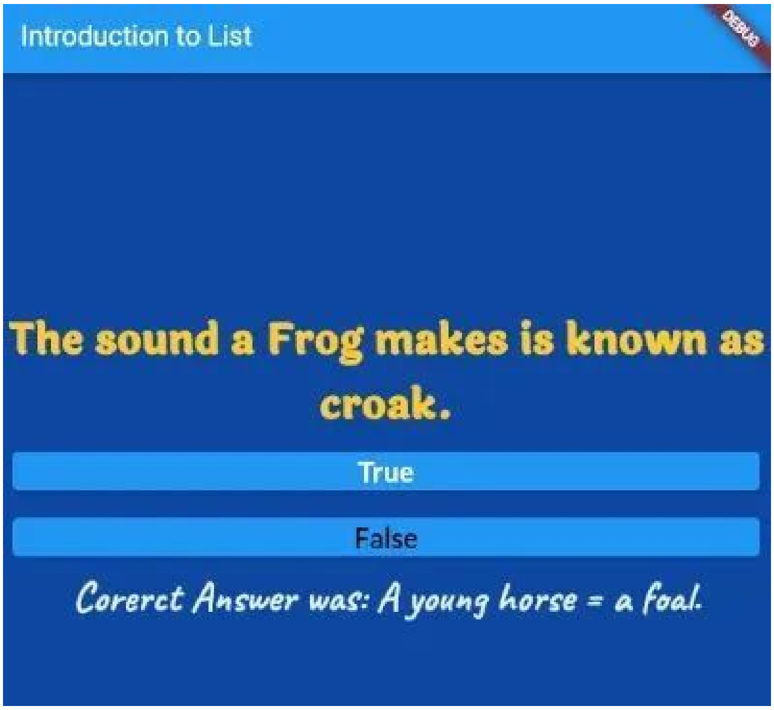
We want to do that centrally, from the same custom theme class.

The answer is, yes, we can.

Not only that, we can add many other features, which will make our Quiz App more interactive.

Let us see what we want to do first.

After that, we will learn how to do that.



The sound a Frog makes is known as croak.

Corerct Answer was: A young horse = a foal.

Figure 4.8 – Flutter theme changes entire app design

Our “Play with Lexis Quiz App” is working the same way. We have not changed the business logic part. Therefore, we are not going to discuss that part.

If you want to know how we can map a list, please read the previous articles. We have discussed how we can use List data type in Flutter to make an interactive app.

In this section, we will concentrate on the Flutter theme part.

Firstly, we will build a custom ThemeData function that will return our custom theme to the MaterialApp theme property.

import ‘package:flutter/material.dart’;

import ‘../model/quiz_theme.dart’; import ‘quiz_page.dart’;

QuizTheme myTheme = QuizTheme();

class QuizApp extends StatelessWidget { const QuizApp({Key? key}) : super(key: key);

```
@override
Widget build(BuildContext context) {
  return MaterialApp(
    title: 'Flutter Demo',
    debugShowCheckedModeBanner: false,
    home: const QuizPage(),

    /// we've started changing and building new theme
    /// from this point
    theme: myTheme.buildTheme(),
  );
}
```

}

Secondly, our newly created "myTheme" object calls the custom ThemeData function "buildTheme()". After that, it returns the value to the theme property of MaterialApp Widget.

Finally, we must take a look at the custom theme class. We have changed it a lot.

import 'package:flutter/material.dart'; import 'package:google_-fonts/google_fonts.dart';

/// In a custom theme page we have described color and fonts /// We may add more custom theme-features later ///

class QuizTheme { static const Color primaryColor = Color(0xFF409B25); static const Color scaffoldBackgroundColor = Color(0xFF2C6F2E); static const Color appBarBack-groundColor = Color(0xFF2C6F2E); static const Color boxDecorationColor = Color(0xFFC5DA28); static const

Color elevatedButtonPrimaryColor = Color(0xFF3C9415); static const Color dividerColor = Color(0xFFD9DB26); static const correctAnswerColor = Color(0xFFFACAFA); static const questionTextColor = Color(0xFFF8E1F8); static const answerColor = Color(0xFFFFFFFF);

```
static TextStyle answerStyle = GoogleFonts.langar(
  textStyle: const TextStyle(
    color: QuizTheme.answerColor,
    fontSize: 20.0,
    fontWeight: FontWeight.bold,
  ),
);

static TextStyle questionStyle = GoogleFonts.laila(
  textStyle: const TextStyle(
    color: QuizTheme.shrineBrown600,
    fontSize: 30.0,
    fontWeight: FontWeight.bold,
  ),
);

static TextStyle appbarStyle = GoogleFonts.salsa(
  textStyle: const TextStyle(
    color: QuizTheme.shrineBrown600,
    fontSize: 20.0,
    fontWeight: FontWeight.bold,
  ),
);

ThemeData _buildShrineTheme() {
  final ThemeData base = ThemeData.light();
  return base.copyWith(
    colorScheme: _shrineColorScheme,
    toggleableActiveColor: shrinePink400,
    primaryColor: shrinePink100,
    primaryColorLight: shrinePink100,
    scaffoldBackgroundColor: shrineBackgroundWhite,
    cardColor: shrineBackgroundWhite,
    textSelectionTheme:
        const TextSelectionThemeData(selectionColor: shri\
nePink100),
    errorColor: shrineErrorRed,
    buttonTheme: ButtonThemeData(
      colorScheme: _shrineColorScheme.copyWith(primary: s\
hrinePink400),
      textTheme: ButtonTextTheme.normal,
    ),
    primaryIconTheme: _customIconTheme(base.iconTheme),
    textTheme: _buildShrineTextTheme(base.textTheme),
    primaryTextTheme: _buildShrineTextTheme(base.primaryT\
extTheme),
    iconTheme: _customIconTheme(base.iconTheme),
  );
}

ThemeData buildTheme() {
  return _buildShrineTheme();
}

IconThemeData _customIconTheme(IconThemeData original) {
  return original.copyWith(color: shrineBrown900);
}

TextTheme _buildShrineTextTheme(TextTheme base) {
  return base
      .copyWith(
        caption: base.caption!.copyWith(
          fontWeight: FontWeight.w400,
          fontSize: 14,
          letterSpacing: defaultLetterSpacing,
        ),
        button: base.button!.copyWith(
          fontWeight: FontWeight.w500,
          fontSize: 14,
          letterSpacing: defaultLetterSpacing,
        ),
      )
      .apply(
        fontFamily: 'Rubik',
        displayColor: shrineBrown900,
        bodyColor: shrineBrown900,
      );
}

static const ColorScheme _shrineColorScheme = ColorScheme(
  primary: shrinePink100,
  secondary: shrinePink50,
  surface: shrineSurfaceWhite,
  background: shrineBackgroundWhite,
  error: shrineErrorRed,
  onPrimary: shrineBrown900,
  onSecondary: shrineBrown900,
  onSurface: shrineBrown900,
  onBackground: shrineBrown900,
  onError: shrineSurfaceWhite,
  brightness: Brightness.light,
);

static const Color shrinePink50 = Color(0xFFFEEAE6);
static const Color shrinePink100 = Color(0xFFFEDBD0);
static const Color shrinePink300 = Color(0xFFFBB8AC);
static const Color shrinePink400 = Color(0xFFEAA4A4);

static const Color shrineBrown900 = Color(0xFF442B2D);
static const Color shrineBrown600 = Color(0xFF7D4F52);

static const Color shrineErrorRed = Color(0xFFC5032B);

static const Color shrineSurfaceWhite = Color(0xFFFFFBFA);
static const Color shrineBackgroundWhite = Colors.white;

static const defaultLetterSpacing = 0.03;
```

}

As we see, there are lots of constant Color properties. Besides, we have built custom ThemeData, TextTheme, and IconThemeData instance methods.

As a result, later, we have used them as necessary to give our Quiz App a complete new look.

Quite naturally, we have modified the code of "quiz_page.dart", as well. So that we can accommodate the bottom navigation bar.

import 'package:flutter/material.dart'; import 'package:rflutter_-alert/rflutter_alert.dart';

import '../model/questions_list.dart'; import '../model/quiz_-theme.dart'; import 'answer.dart'; import 'question.dart';

class QuizPage extends StatefulWidget { const QuizPage({Key? key}) : super(key: key);

```
@override
State<QuizPage> createState() => _QuizPageState();
```

}

class _QuizPageState extends State<QuizPage> { int _currentIndex = 0; QuizMaster quiz = QuizMaster(); String _correctAnswer = 'Choose your correct answer!'; int _index = 0; void increment() { setState(() { _index = _index + 1; }); if (_index == quiz.questionList.length + 1) { _index = 0; } if (_index == 0) {

_correctAnswer = ‘Choose your correct answer!’; } else if (_index == 1) { _correctAnswer = ‘Synonym of Mendacity was: Falsehood’; } else if (_index == 2) { _correctAnswer = ‘Synonym of Culpable was: Guilty’; } else { _index = 0; _correctAnswer = ‘Choose your correct answer!’; Alert(context: context, title: ‘Quiz Completed.’, desc: ‘We’ve reached the end. Thanks for taking part. Meet you again.’,).show(); } }

```
@override
Widget build(BuildContext context) {
  final colorScheme = Theme.of(context).colorScheme;
  final textTheme = Theme.of(context).textTheme;
  return Scaffold(
    appBar: AppBar(
      title: Text(
        'Playxis - Play + Lexis',
        style: QuizTheme.appbarStyle,
      ),
      backgroundColor: QuizTheme.shrinePink300,
    ),
    body: SafeArea(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        mainAxisSize: MainAxisSize.min,
        children: [
          Question(questions: quiz.questionList, index: _\
index),
          ...(quiz.questionList[_index]['answer'] as List\
<String>)
              .map((answer) {
            return Answer(answer: answer, pointToOnPress:\
 increment);
          }).toList(),
          Container(
            padding: const EdgeInsets.all(5.0),
            width: 250.0,
            child: const Divider(
              thickness: 5.0,
              color: QuizTheme.shrinePink400,
            ),
          ),
          Container(
            width: double.infinity,
            alignment: Alignment.topCenter,
            margin: const EdgeInsets.only(
              left: 10.0,
              right: 10.0,
            ),
            child: Text(
              _correctAnswer,
              style: QuizTheme.questionStyle,
            ),
          ),
        ],
      ),
    ),
    bottomNavigationBar: BottomNavigationBar(
      type: BottomNavigationBarType.fixed,
      currentIndex: _currentIndex,
      backgroundColor: colorScheme.surface,
      selectedItemColor: colorScheme.onSurface,
      unselectedItemColor: colorScheme.onSurface.withOpac\
ity(.60),
      selectedLabelStyle: textTheme.caption,
      unselectedLabelStyle: textTheme.caption,
      onTap: (value) {
        // Respond to item press.
        setState(
          () => _currentIndex = value,
        );
      },
      items: const [
        BottomNavigationBarItem(
          label: 'Favorites',
          icon: Icon(Icons.favorite),
        ),
        BottomNavigationBarItem(
          label: 'Music',
          icon: Icon(Icons.music_note),
        ),
        BottomNavigationBarItem(
          label: 'Places',
          icon: Icon(Icons.location_on),
        ),
        BottomNavigationBarItem(
          label: 'News',
          icon: Icon(Icons.library_books),
        ),
      ],
    ),
  );
}
```

}

In some place we have also used the modified custom theme class.

In some place, we have used the constant Color variables that are static in nature. As a result we can access those constant Color properties through class name.

In the code above, we have seen a few examples like the following.

color: QuizTheme.shrinePink400, … backgroundColor: QuizTheme.shrinePink300,

After changing the custom theme from dark to light, we can also add some more pages. Aside from this, we will learn how to navigate to another page.

In the next section, we learn them. Before that, if you want to clone the newly modified Quiz App, please use this GitHub repository.

So stay tuned and happy Fluttering.

The code repositories for this section[15]

How we use List in the Quiz App?

One of the most common collection in every programming language is List. It is also known as array in other programming language. The list in Flutter is an ordered group of objects. We commonly call them lists.

The list plays a very important role in any Flutter App. You may ask why?

The answer is quite simple.

Any small to medium, or a big Flutter App needs to place data in order. Right?

Take a look at this Flutter App.

We have two separate lists of questions and answers. As a built in data type, list in Flutter allows any data "Type". That means, if we keep our list objects in a Class, we can call the list by mentioning that class data Type. We will see a lot of examples later.

Anyway, it is always easy to find any particular element when they are placed in ordered list.

Because an ordered list has an index that starts from 0, we can easily track that index number and find out the element.

In addition, Flutter and Dart comes with plenty of methods that we can use to handle the List.

[15]https://github.com/sanjibsinha/quiz_master/tree/material-design

Before we jump in to learn how we have built this simple "English Test App" let us some list examples in Dart console.

Watch the output, that will also clarify how list in Flutter and Dart works.

void main() {

```
IntroToList question = IntroToList();

print(question.questions.length); // 4

print(question.questions.reversed);

/**
* (The synonym of scorn is despise.,
* The sound a Frog makes is known as croak.,
* A young horse is called a duckling.,
* 640 acres equal 1 square mile.)*
*
* */

print(question.questions.firstWhere((i) => i.length > 1))\
;
// 640 acres equal 1 square mile.

print(question.questions.first);
// 640 acres equal 1 square mile.

print(question.questions.every((element) => element.start\
sWith('a')));
// false
// because every element does not start with letter 'a'



question.checkAnswer(1);
```

```
// Corerct Answer was: A young horse = a duckling.
question.checkAnswer(3);
// The synonym of scorn is despise.

for (final q in question.questions) {
  print(q);
}

/**
* <<< OUTPUT>>>
```

640 acres equal 1 square mile. A young horse is called a duckling. The sound a Frog makes is known as croak. The synonym of scorn is despise. * * * */

}

class IntroToList {

```
List<String> questions = [
  '640 acres equal 1 square mile.',
  'A young horse is called a duckling.',
  'The sound a Frog makes is known as croak.',
  'The synonym of scorn is despise.',
];

int questionIndex = 0;

List<bool> answers = [
  true,
  false,
  true,
  true,
];

String check = '';
```

```

void checkAnswer(int questionIndex) {

  if (questionIndex == 0) {
    check = '';
  } else if (questionIndex == 1) {
    check = questions[questionIndex];
  } else if (questionIndex == 2) {
    check = questions[questionIndex];
  } else if (questionIndex == 3) {
    check = questions[questionIndex];
  } else if (questionIndex > 3) {
    check = questions[questionIndex];
  }

  print(check);
}
```

}

What we see above is just a glimpse. There are plenty of other methods that Dart and Flutter List uses.

However, we do not have to use them all. In fact, a very few of the methods are used while we build a Flutter App.

In the coming sections, we will see how we can use List in Flutter to make different type of Applications.

To start with, let us build a Flutter App, that tests English knowledge. It is a simple App that displays a question. Below that question we have true and false button.

You press any button, and it will show the correct answer below. As a result, you know whether you have pressed the correct button, or not.

As we progress, we will make this Flutter App more interesting.

For the time being let us be simple in our approach so that we can understand the basic of Flutter lists.

Let us start with the top-level main() function which is our entry point.

import ‘package:flutter/material.dart’;

import ‘view/app.dart’;

void main() ⇒ runApp(const App());

Next, we create a “view” folder, in our root directory, and place the custom Widget App() there.

That will take us to the AppHomePage() Widget.

import ‘package:flutter/material.dart’;

import ‘app_home_page.dart’;

class App extends StatelessWidget { const App({Key? key}) : super(key: key);

```dart
@override
Widget build(BuildContext context) {
  return const MaterialApp(
    title: 'Introduction to List',
    home: AppHomePage(),
  );
}
```

}

In the AppHomePage() Widget, we will declare two separate lists. One list will be a collection of questions, and the other will be answers.

However, two lists will be of different data type.

Let us take a look at the lists firstly.

List<String> questions = [‘640 acres equal 1 square mile.’, ‘A young horse is called a duckling.’, ‘The sound a Frog makes is known as croak.’, ‘The synonym of scorn is despise.’,];

```
int questionIndex = 0;

List<bool> answers = [
  true,
  false,
  true,
  true,
];
```

We must always declare the data type of the list items.

As we see, the first list has a data type String. Subsequently, the second list has data type Boolean. Therefore, it consists of true and false.

In between two lists, we have an index number which is of integer data type. We need to track, or iterate both lists at the same time. Consequently, we can track which button is pressed. True or false?

As the first list of questions shows, the second statement is false. Otherwise, the rest of the lists are true.

Therefore, we have made our second list of answers matching them in accordance with the questions.

Let us see the full code snippet now.

import ‘package:flutter/material.dart’; import ‘package:google_-fonts/google_fonts.dart’;

class AppHomePage extends StatefulWidget { const AppHome-Page({Key? key}) : super(key: key);

```
@override
_AppHomePageState createState() => _AppHomePageState();
```

}

class _AppHomePageState extends State<AppHomePage> { List<String> questions = ['640 acres equal 1 square mile.', 'A young horse is called a duckling.', 'The sound a Frog makes is known as croak.', 'The synonym of scorn is despise.',];

```
int questionIndex = 0;

List<bool> answers = [
  true,
  false,
  true,
  true,
];

String check = '';

void checkAnswer() {
  if (questionIndex == 0) {
    check = '';
  } else if (questionIndex == 1) {
    check = 'Corerct Answer was: 640 acres = 1 square mil\
e.';
  } else if (questionIndex == 2) {
    check = 'Corerct Answer was: A young horse = a foal.';
  } else if (questionIndex == 3) {
    check = 'Corerct Answer was: The sound a Frog makes =\
 croak.';
  } else if (questionIndex > 3) {
    check = 'Corerct Answer was: The synonym of scorn = d\
espise.';
  }
```

```
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Colors.blue.shade900,
    appBar: AppBar(
      title: const Text('Introduction to List'),
    ),
    body: SafeArea(
      child: Center(
        child: Column(
          mainAxisSize: MainAxisSize.min,
          mainAxisAlignment: MainAxisAlignment.center,
          crossAxisAlignment: CrossAxisAlignment.stretch,
          children: [
            Text(
              questions[questionIndex],
              textAlign: TextAlign.center,
              style: GoogleFonts.laila(
                textStyle: Theme.of(context).textTheme.he\
adline6,
                fontSize: 30,
                fontWeight: FontWeight.w900,
                color: Colors.amber.shade400,
              ),
            ),
            buildButtonInsideContainer(
              'True',
              Colors.white,
            ),
            buildButtonInsideContainer(
              'False',
              Colors.black,
            ),
              Text(
                check,
                textAlign: TextAlign.center,
                style: GoogleFonts.caveat(
                  textStyle: Theme.of(context).textTheme.he\
adline6,
                  fontSize: 20,
                  fontWeight: FontWeight.w700,
                  color: Colors.blue.shade50,
                ),
              ),
            ],
          ),
        ),
      ),
    );
}

Container buildButtonInsideContainer(String answer, Color\
 color) {
  return Container(
    padding: const EdgeInsets.all(10.0),
    child: ElevatedButton(
      onPressed: () {
        setState(() {
          answers[questionIndex];
          questionIndex++;
          checkAnswer();
        });
        if (questionIndex > 3) {
          questionIndex = 0;
        }
      },
      child: Text(
        answer,
        style: GoogleFonts.lato(
          textStyle: Theme.of(context).textTheme.headline\
6,
          fontSize: 20,
          fontWeight: FontWeight.w700,
          color: color,
        ),
      ),
    ),
  );
}
```

}

Since both the Button Widgets inside a Container Widget, we have used a common function that return a Container Widget. As parameters we pass the "answer" and the "color".

Why?

Because one answer might be "True", or "False".

However, we track the index number of question list, and each press moves the question forward.

Besides, through a simple if-else conditional, we track the index number and show the correct answer.

Watch the second question below.

The answer is false. As we know that young horse is known as a foal.

Therefore, if someone presses the "True" button, without knowing the correct answer, she will be informed immediately.

Suppose a user presses any button, by knowingly, or unknwoingly.

She will be notified and the third question will pop up at the same time.

Although we have shown the full code, still you may wish to clone the project and run it in your local machine.

In that case, please visit the respective branch of the GitHub repository.

In the next section we will try to make this Flutter App more object oriented.

So stay tuned, and happy Fluttering.

The code repositories for this ection[16]

List and Map in Quiz App

In our last section, we have built a Flutter Quiz App using List. However, we tried to do in a simple way.

For one reason.

We did not want to make things complicated firstly. Secondly, we tried to give an idea how we can use list in Flutter.

Earlier we have learned that list is a collection of items. It is an ordered collection. Moreover it is sequential.

Because a list in Flutter is sequential, the index number starts with 0. And every index refers to an item.

As a result, we can easily manipulate a list. And to do that, Dart comes with a lot of list methods. In the previous section, we have also seen some examples.

This time, we will build one Quiz App in a more object oriented way.

[16]https://github.com/sanjibsinha/elementary_dart_flutter_for_beginners/tree/chap-six-1-intro-list

Why we need Object Oriented Style?

We know Dart is an object oriented programming language. As a consequence, everything in Dart is Object.

That means, behind every Object there is a Class. Which defines the Type of that object.

We can relate this concept to Flutter. Because, in Flutter, everything is Widget, which is actually a Class.

There are four pillars in Object Oriented Programming.

Abstraction Encapsulation Inheritance Polymorphism

When we implement these concepts, we understand them better. While building the Quiz App, in an object oriented way, we will implement first two concepts. Abstraction and Encapsulation.

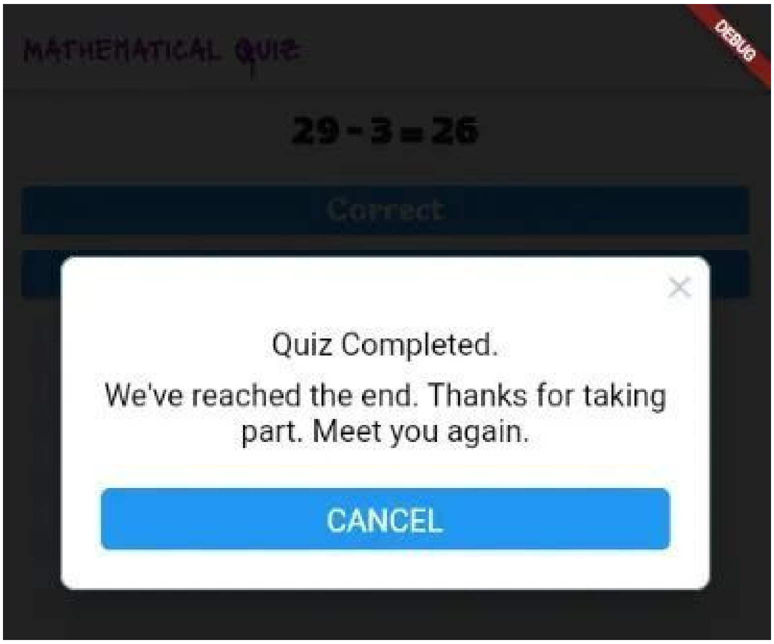
Firstly, we have used two Flutter packages to give our App a unique look and feel.

Therefore, let us first add the dependencies in our “pubspec.yaml” file.

dependencies: flutter: sdk: flutter

```
google_fonts: ^2.3.1
rflutter_alert: ^2.0.4
```

We have used Google Fonts package before. However, the alert dialog package is new. Basically when our quiz ends, it will alert the user with a dialog box.



Quiz Completed.

We've reached the end. Thanks for taking part. Meet you again.

Figure 4.9 – Flutter list Quiz App alert dialog at the end

We will develop the previous Quiz App in an Object oriented way.

To implement the first principle of Abstraction, we will create a Question class in our model folder.

class Question { String question; bool answer;

```
Question(
  this.question,
  this.answer,
);
```

}

Now, inside another class QuizMaster we can add the list items by using the Question Class constructors.

As a result, we can establish a relationship between question and answer in one object.

We have also declared an integer as the list index and set the value to 0.

import ‘question.dart’;

class QuizMaster { int _indexNumber = 0;

```
final List<Question> _quiz = [
  Question('29 - 3 = 26', true),
  Question('711 - 4 = 677', false),
  Question('455 * 3 = 1365', true),
  Question('76 / 8 = 9.5', true),
  Question('Many Thanks, press any button to end the Quiz\
.', true),
];

void nextQuestion() {
  if (_indexNumber <= _quiz.length) {
    _indexNumber++;
  }
}

String getQuestion() {
  return _quiz[_indexNumber].question;
}

bool getAnswer() {
  return _quiz[_indexNumber].answer;
}

bool isFinished() {
  if (_indexNumber >= _quiz.length - 1) {
    return true;
  } else {
    return false;
  }
}

void reset() {
  _indexNumber = 0;
}
```

}

As we see in the above code, now we are in more control to move forward our questions.

Not only that, we can get the question, answer and move to the next question.

Encapsulation and Private property

Apart from that, we have marked each property of the class as "private" by adding an underscore "_" before the name of the property.

class QuizMaster { int _indexNumber = 0;

```
final List<Question> _quiz = [
```

…

It implements another important principle of object oriented programming. Encapsulation.

As we have made each property "private", no one can access these properties from outside this class.

Now we can create a QuizMaster object in our top-level main() function. And, consequently, we can create our whole quiz app.

import 'package:flutter/material.dart'; import 'package:google_-fonts/google_fonts.dart';

import 'package:rflutter_alert/rflutter_alert.dart'; import 'model/quiz_master.dart';

QuizMaster quizMaster = QuizMaster();

void main() ⇒ runApp(const QuizApp());

class QuizApp extends StatelessWidget { const QuizApp({Key? key}) : super(key: key);

```
@override
Widget build(BuildContext context) {
  return MaterialApp(
    home: Scaffold(
      appBar: AppBar(
        backgroundColor: Colors.white,
        title: Text(
          'Mathematical Quiz',
          style: GoogleFonts.lacquer(
            textStyle: TextStyle(
              color: Colors.purple.shade600,
              fontSize: 20.0,
              fontWeight: FontWeight.bold,
            ),
          ),
        ),
      ),
      body: const SafeArea(
        child: Padding(
          padding: EdgeInsets.symmetric(horizontal: 10.0),
          child: QuizPage(),
        ),
      ),
    ),
  );
}
```

}

class QuizPage extends StatefulWidget { const QuizPage({Key? key}) : super(key: key);

```
@override
_QuizPageState createState() => _QuizPageState();
```

}

class _QuizPageState extends State<QuizPage> { List<Text> check = [];

```
void checkAnswer(bool userPickedAnswer) {
  bool correctAnswer = quizMaster.getAnswer();

  setState(() {
    if (quizMaster.isFinished() == true) {
      Alert(
        context: context,
        title: 'Quiz Completed.',
        desc:
            'We\'ve reached the end. Thanks for taking pa\
rt. Meet you again.',
      ).show();

      quizMaster.reset();

      check = [];
    } else {
      if (userPickedAnswer == correctAnswer) {
        check.add(
          Text(
            'You\'re Right. Well Done.',
            style: GoogleFonts.laila(
              textStyle: const TextStyle(
                fontSize: 20.0,
                fontWeight: FontWeight.bold,
              ),
            ),
          ),
        );
      } else {
        check.add(
          Text(
            'You\'re Wrong. Try Again.',
            style: GoogleFonts.laila(
              textStyle: const TextStyle(
                fontSize: 20.0,
                fontWeight: FontWeight.bold,
              ),
            ),
          ),
        );
      }
      quizMaster.nextQuestion();
    }
  });
}

@override
Widget build(BuildContext context) {
  return SafeArea(
    top: true,
    child: Center(
      child: ListView(
        children: <Widget>[
          Container(
      margin: const EdgeInsets.fromLTRB(20.0, 5.0, \
20.0, 5.0),
      alignment: Alignment.center,
      child: Text(
       quizMaster.getQuestion(),
       style: GoogleFonts.lalezar(
        textStyle: const TextStyle(
         fontSize: 30.0,
         fontWeight: FontWeight.bold,
        ),
       ),
      ),
     ),
     checkingAnswer('Correct', true),
     checkingAnswer('Wrong', false),
     Padding(
      padding: const EdgeInsets.all(8.0),
      child: Column(
       mainAxisSize: MainAxisSize.min,
       mainAxisAlignment: MainAxisAlignment.center,
       children: check,
      ),
     )
    ],
   ),
  ),
 );
}

Container checkingAnswer(String corerctOrWrong, bool true\
OrFalse) {
 return Container(
  padding: const EdgeInsets.all(5.0),
  child: ElevatedButton(
   onPressed: () {
          checkAnswer(trueOrFalse);
        },
        child: Text(
          corerctOrWrong,
          style: GoogleFonts.laila(
            textStyle: const TextStyle(
              fontSize: 20.0,
              fontWeight: FontWeight.bold,
            ),
          ),
        ),
      ),
    );
}
```

}

we have used common Container Widget for two Elevated Buttons.

Therefore, we can refactor this part of code. We have extracted a common method only changing the parameters.

checkingAnswer('Correct', true), checkingAnswer('Wrong', false),

How Flutter List Quiz App works

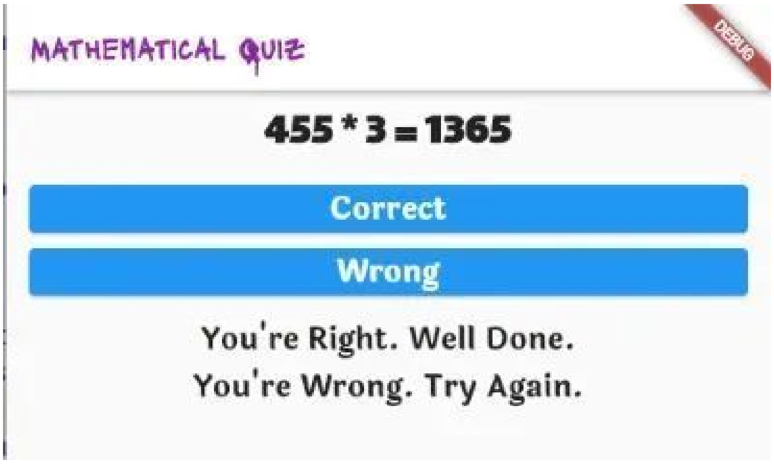
Here the setState() method plays a crucial role. As it updates the question. It also checks whether the answer is correct or wrong.

After that, it adds the answers to another list which we show inside a Column below.

The first mathematical equation is quite easy. It is correct.

When user presses the Correct button, it will rightly show the answer and displays the next equation.

Suppose this time user cannot choose the correct answer. Accordingly, it will show that user is wrong, and displays the third equation as follows.



455 * 3 = 1365

You're Right. Well Done.
You're Wrong. Try Again.

Figure 4.10 – Flutter list Quiz App second example

As we progress, one time, we reach the end point.

Now it is time to press any button to finish the quiz.

Completing this journey, will finally display the alert dialog that we have seen before.

You can just create the same Quiz App by following the above code. Or, you can clone the respective GitHub repository.

The code repositories for this ection[17]

[17]https://github.com/sanjibsinha/quiz_master/tree/mathematico

5. Let’s Build a Happiness Calculator

We are going to build a Flutter App. Name is “Happiness Calculator”. In this section, we will learn how to use Slider Widget.

So far we have been progressing step-by-step.

We have learned how to use enum. After that, we have used ternary operator to reduce the code length.

In this section we will learn what is the Slider widget. How to use it. Moreover, why we need the Slider Widget.

What is Slider Widget

Firstly, the Slider Widget requires a Material widget as its ancestor.

Secondly, it inherits from a MediaQuery widget. That means we need the MaterialApp widget at the top of the Widget tree.

Next, we need to remember another important thing.

The Slider Widget selects a range of values. It starts from the minimum and might proceed to the maximum.

As a result, the state changes. The build() method rebuilds the Widget. Therefore, we also need a Stateful Widget that will track the number.

Why?

Because as we drag the slider, the number changes.

Let us take a look. That will give us an idea how the Slider Widget works.

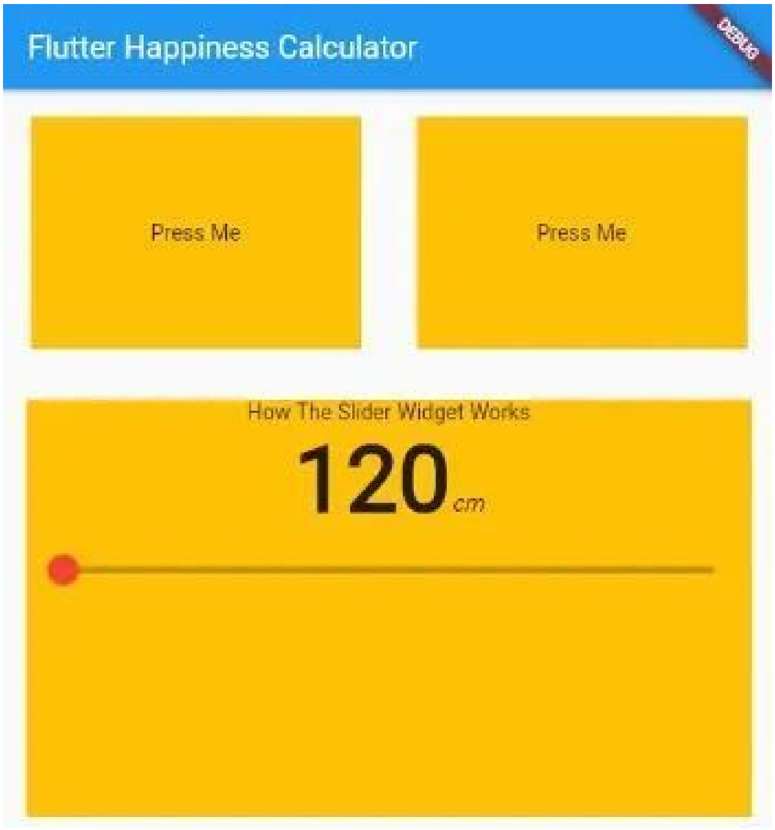


Figure 5.1 – Slider displays with the minimum value

As we drag the Slider, it proceeds to the maximum value.

During dragging the state changes. However, the Slider widget does not maintain any state.

On the contrary, to maintain state, the slider widget calls the onChanged callback.

As a result, when we drag it further, the number increases.

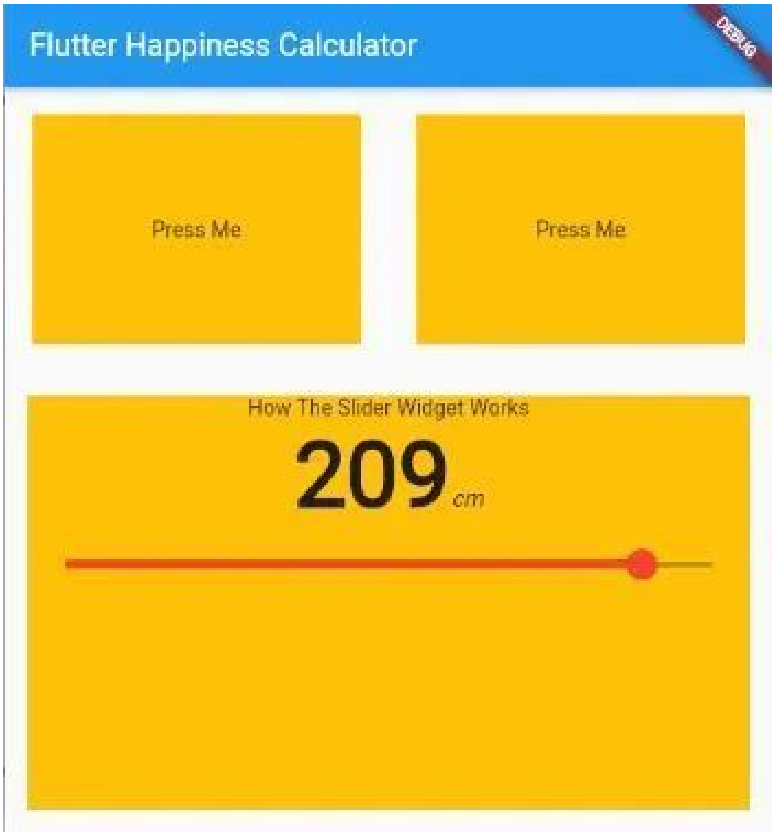


Figure 5.2 – Slider proceeds to the maximum value

Why we need and how to use Slider

Firstly, with the help of the Slider Widget we can change a value. This mechanism might help us to decide something else with respect to that value.

Since we want to calculate happiness, we can set the value by dragging the slider.

Secondly, the onChanged property expects a callback that must have the setState() method.

Not only that, it also passes a double value that takes the new value as the value changes.

Finally we see how the Slider Widget uses the abstraction.

To make it happen, we just initialise an integer value which points to the minimum value. Once the range is set, we can provide the maximum value also.

```
Slider(
                    value: height.toDouble(),
                    min: 120.0,
                    max: 220.0,
                    activeColor: activeColor,
                    inactiveColor: Colors.black26,
                    onChanged: (double newValue) {
                        setState(() {
                        height = newValue.round();
                        });
                    },
                    ),
```

We see in the above code many properties of the Slider Widget. But we can customise it more by adding more features to it.

Let us take a look at the full code, so it will clear every confusion.

```
import 'package:flutter/material.dart';

import '../model/constants.dart';
import '../model/container_color.dart';

class HappinessHomePage extends StatefulWidget {
const HappinessHomePage({
    Key? key,
    required this.title,
}) : super(key: key);

final String title;

@override
State<HappinessHomePage> createState() => _HappinessHomeP\
ageState();
}

class _HappinessHomePageState extends State<HappinessHome\
Page> {
ContainerColor? selectedContainer;
int height = 120;

@override
Widget build(BuildContext context) {
    return Scaffold(
    appBar: AppBar(
        title: Text(widget.title),
    ),
    body: Center(
        child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
            Row(
            children: [
                expandEnum(ContainerColor.first),
                expandEnum(ContainerColor.second),
            ],
            ),
            Expanded(
            child: Container(
                margin: const EdgeInsets.all(15.0),
                alignment: Alignment.center,
                color: inactiveColor,
                width: double.infinity,
                child: Column(
                children: [
                    const Text('How The Slider Widget Wor\
ks'),
                    Row(
                    mainAxisAlignment: MainAxisAlignment.\
center,
                    crossAxisAlignment: CrossAxisAlignmen\
t.baseline,
                    textBaseline: TextBaseline.alphabetic,
                    children: [
                        Text(
                        height.toString(),
                        style: const TextStyle(
                            fontSize: 60.0,
                            fontWeight: FontWeight.w900,
                        ),
                        ),
                        const Text(
                        'cm',
                        style: TextStyle(
                            fontSize: 15.0,
                            fontWeight: FontWeight.w100,
                            fontStyle: FontStyle.italic,
                        ),
                        ),
                    ],
                    ),
                    Slider(
                    value: height.toDouble(),
                    min: 120.0,
                    max: 220.0,
                    activeColor: activeColor,
                    inactiveColor: Colors.black26,
                    onChanged: (double newValue) {
                        setState(() {
                        height = newValue.round();
                        });
                    },
                    ),
                ],
                ),
            ),
            )
        ],
        ),
    ),
    );
}

Expanded expandEnum(ContainerColor? containerColor) {
    return Expanded(
    child: Padding(
        padding: const EdgeInsets.all(18.0),
        child: GestureDetector(
        onTap: () {
            setState(() {
            selectedContainer = containerColor;
            });
        },
          child: Container(
            alignment: Alignment.center,
            color: selectedContainer == containerColor
                ? activeColor
                : inactiveColor,
            width: 150.0,
            height: 150.0,
            child: const Text('Press Me'),
          ),
          ),
      ),
      );
}
}
```

In the next section we will learn to customise the Slider. For example the round thumb inside the Slider might be bigger.

Previously we have learned how to use a common theme across the entire Flutter App.

We can implement that concept here. By the way, if you want to clone this part of the project please visit the respective branch of GitHub repository.

So stay tuned.

Customizing the theme

We have been building a "Happiness Calculator App" in Flutter. However, we are building it step-by-step. In this section, we will customize the theme of Slider Widget. And, to do that we will use "SliderTheme" in Flutter.

Firstly, we have discussed enum before. Secondly, we have reduced the code size by using the ternary operator. And finally, we have discussed Slider in Flutter.

Actually, these are steps that define how we can build the App.

During building the App, the last stage was as follows.

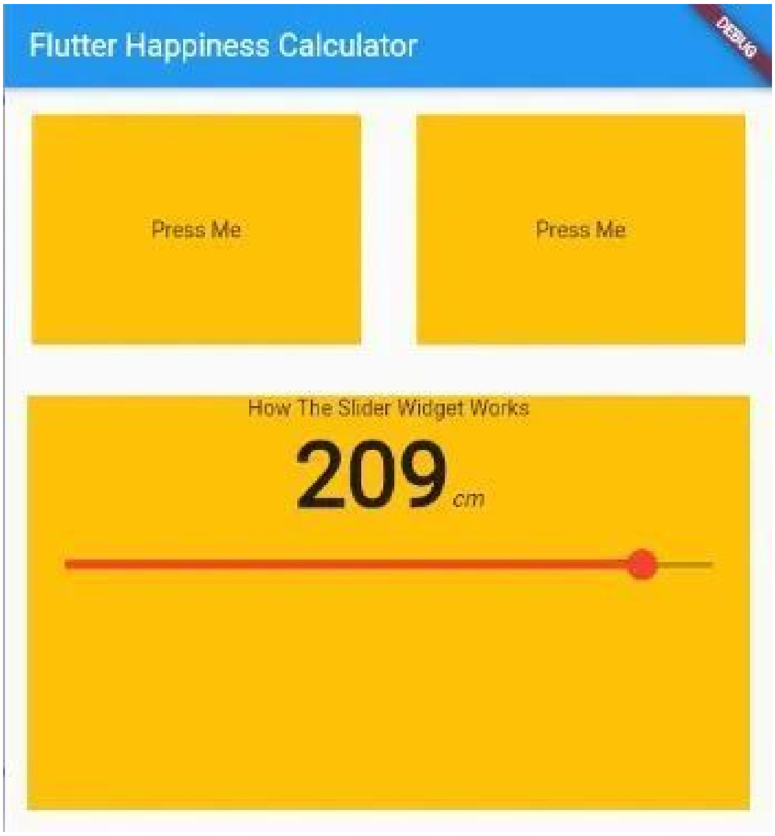


Figure 5.3 – Slider proceeds to the maximum value

However, we want to give it a professional look. And to do that, we must have a common theme that applies color and font across the entire app.

As a result, it will look as follows.

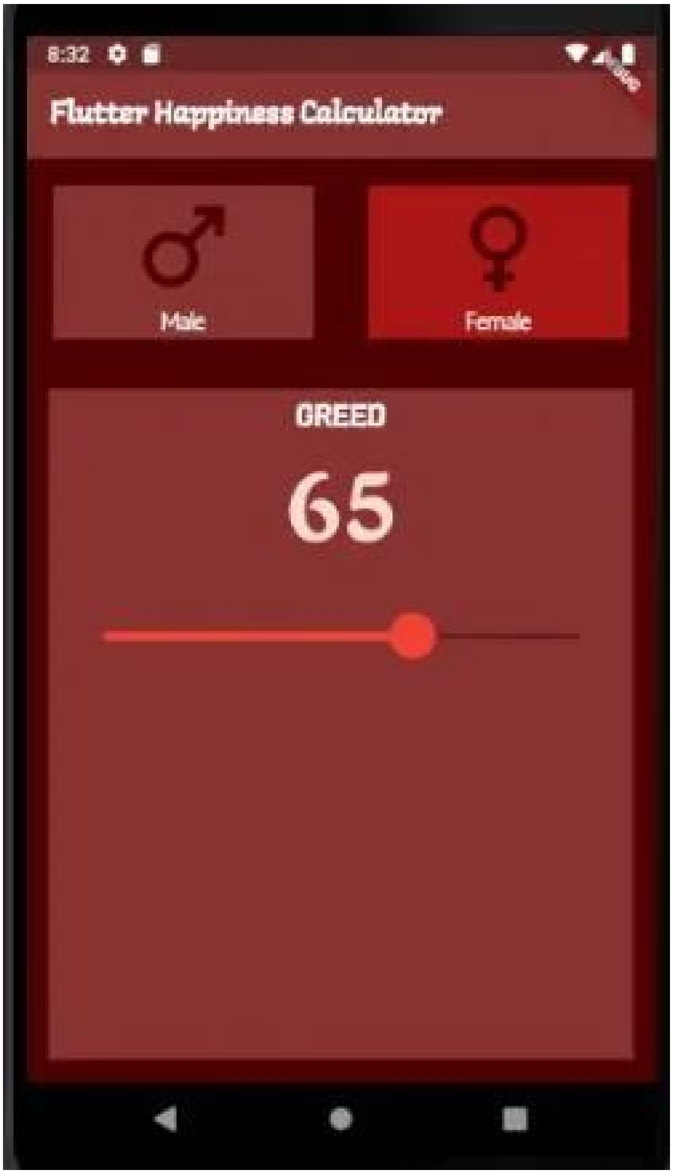


Figure 5.4 – Customized Slider Theme that runs across the entire Flutter App

Have you not read the previous sections where we have discussed how to customize color and fonts across the App? In that case, please read how to customize color, and how to customize font and design.

Firstly, when we try to calculate Happiness, we need some inputs. Say, it is greed. A greedy person suffers from eternal unhappiness.

Therefore, as we raise the Slider bar that means we raise the amount of Greed.

Secondly, the amount of happiness differs from male to female. So we have used two icons that represent gender.

As a whole, up to now, we have designed this part. And in this section, we will learn how we have achieved that.

How to use the custom theme

Although we have discussed this part before, still recapitulation will not harm.

To customize a theme that will work across the entire Flutter App, we need a custom theme class.

Let us see how this custom theme class look like.

```
import 'package:flutter/material.dart';
import 'package:google_fonts/google_fonts.dart';

/// In a custom theme page we have described color and fo\
nts
/// We may add more custom theme-features later
///

class HappyTheme {
static const Color primaryColor = Color(0xFF409B25);
static const Color scaffoldBackgroundColor = Color(0xFF2C\
6F2E);
static const Color appBarBackgroundColor = Color(0xFF2C6F\
2E);
static const Color boxDecorationColor = Color(0xFFC5DA28);
static const Color elevatedButtonPrimaryColor = Color(0xF\
F3C9415);
static const Color dividerColor = Color(0xFFD9DB26);
static const correctAnswerColor = Color(0xFFFACAFA);
static const questionTextColor = Color(0xFFF8E1F8);
static const answerColor = Color(0xFFFFFFFF);

static TextStyle answerStyle = GoogleFonts.langar(
    textStyle: const TextStyle(
    color: HappyTheme.answerColor,
    fontSize: 20.0,
    fontWeight: FontWeight.bold,
    ),
);

static TextStyle greedStyle = GoogleFonts.laila(
    textStyle: const TextStyle(
    color: HappyTheme.shrinePink100,
    fontSize: 60.0,
    fontWeight: FontWeight.bold,
    ),
);

static TextStyle genderStyle = GoogleFonts.antic(
    textStyle: const TextStyle(
    color: HappyTheme.shrineSurfaceWhite,
    fontSize: 15.0,
    fontWeight: FontWeight.bold,
    ),
);

```

```
static TextStyle questionStyle = GoogleFonts.laila(
    textStyle: const TextStyle(
    color: HappyTheme.shrineBrown600,
    fontSize: 30.0,
    fontWeight: FontWeight.bold,
    ),
);

static TextStyle appbarStyle = GoogleFonts.salsa(
    textStyle: const TextStyle(
    color: HappyTheme.shrineBrown600,
    fontSize: 20.0,
    fontWeight: FontWeight.bold,
    ),
);

ThemeData _buildShrineTheme() {
    final ThemeData base = ThemeData.light();
    return base.copyWith(
    colorScheme: _shrineColorScheme,
    toggleableActiveColor: shrinePink400,
    primaryColor: shrinePink100,
    primaryColorLight: shrinePink100,
    scaffoldBackgroundColor: shrineBackgroundWhite,
    cardColor: shrineBackgroundWhite,
    textSelectionTheme:
        const TextSelectionThemeData(selectionColor: shri\
nePink100),
    errorColor: shrineErrorRed,
    buttonTheme: ButtonThemeData(
        colorScheme: _shrineColorScheme.copyWith(primary:\
 shrinePink400),
        textTheme: ButtonTextTheme.normal,
    ),
    primaryIconTheme: _customIconTheme(base.iconTheme),
    textTheme: _buildShrineTextTheme(base.textTheme),
    primaryTextTheme: _buildShrineTextTheme(base.primaryT\
extTheme),
    iconTheme: _customIconTheme(base.iconTheme),
    );
}

ThemeData buildTheme() {
    return _buildShrineTheme();
}

IconThemeData _customIconTheme(IconThemeData original) {
    return original.copyWith(color: shrineBrown900);
}

TextTheme _buildShrineTextTheme(TextTheme base) {
    return base
        .copyWith(
        caption: base.caption!.copyWith(
            fontWeight: FontWeight.w400,
            fontSize: 14,
            letterSpacing: defaultLetterSpacing,
        ),
        button: base.button!.copyWith(
            fontWeight: FontWeight.w500,
            fontSize: 14,
            letterSpacing: defaultLetterSpacing,
        ),
        )
        .apply(
        fontFamily: 'Rubik',
        displayColor: shrineBrown900,
        bodyColor: shrineBrown900,
        );
}
```

```

static const ColorScheme _shrineColorScheme = ColorScheme(
    primary: shrinePink100,
    secondary: shrinePink50,
    surface: shrineSurfaceWhite,
    background: shrineBackgroundWhite,
    error: shrineErrorRed,
    onPrimary: shrineBrown900,
    onSecondary: shrineBrown900,
    onSurface: shrineBrown900,
    onBackground: shrineBrown900,
    onError: shrineSurfaceWhite,
    brightness: Brightness.light,
);

static const Color activeCoor = Color(0xFFaa1111);
static const Color inactiveCoor = Color(0xFF893131);

static const Color shrinePink50 = Color(0xFFFEEAE6);
static const Color shrinePink100 = Color(0xFFFEDBD0);
static const Color shrinePink300 = Color(0xFFFBB8AC);
static const Color shrinePink400 = Color(0xFFEAA4A4);

static const Color shrineBrown900 = Color(0xFF4f0808);
static const Color shrineBrown600 = Color(0xFF893131);

static const Color shrineErrorRed = Color(0xFFC5032B);

static const Color shrineSurfaceWhite = Color(0xFFFFFBFA);
static const Color shrineBackgroundWhite = Colors.white;

static const defaultLetterSpacing = 0.03;
}
```

We can give this file any name, however, the name should be meaningful.

Next, we will use this custom theme object where we need to reflect this theme design.

It could be any color or style property of the Scaffold, AppBar, or the Text Widget.

Consequently, we can take a look at the full code.

```
import 'package:flutter/material.dart';

import '../model/happ_theme.dart';
import '../model/constants.dart';
import '../model/container_color.dart';

class HappinessHomePage extends StatefulWidget {
const HappinessHomePage({
    Key? key,
    required this.title,
}) : super(key: key);

final String title;

@override
State<HappinessHomePage> createState() => _HappinessHomeP\
ageState();
}

class _HappinessHomePageState extends State<HappinessHome\
Page> {
ContainerColor? selectedContainer;
int height = 0;

@override
Widget build(BuildContext context) {
    return Scaffold(
    backgroundColor: HappyTheme.shrineBrown900,
    appBar: AppBar(
        title: Text(
        widget.title,
        style: HappyTheme.answerStyle,
        ),
        backgroundColor: HappyTheme.shrineBrown600,
    ),
    body: Center(
        child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
            Row(
            children: [
                expandEnum(
                ContainerColor.first,
                'Male',
                Icons.male,
                ),
                expandEnum(
                ContainerColor.second,
                'Female',
                Icons.female,
                ),
            ],
            ),
            Expanded(
            child: Container(
                margin: const EdgeInsets.all(15.0),
                alignment: Alignment.center,
                color: HappyTheme.shrineBrown600,
                width: double.infinity,
                child: Column(
                children: [
                    Container(
                    padding: const EdgeInsets.only(
                        top: 5.0,
                    ),
                    child: Text(
                        'GREED',
                        style: HappyTheme.answerStyle,
                    ),
                    ),
                    Row(
                    mainAxisAlignment: MainAxisAlignment.\
center,
                    crossAxisAlignment: CrossAxisAlignmen\
t.baseline,
                    textBaseline: TextBaseline.alphabetic,
                    children: [
                        Text(
                        height.toString(),
                        style: HappyTheme.greedStyle,
                        ),
                    ],
                    ),
                    SliderTheme(
                    data: SliderTheme.of(context).copyWit\
h(
                        inactiveTrackColor: HappyTheme.sh\
rinePink50,
                        activeTrackColor: HappyTheme.shri\
neBackgroundWhite,
                        thumbColor: HappyTheme.shrineErro\
rRed,
                        overlayColor: HappyTheme.shrinePi\
nk50,
                        thumbShape: const RoundSliderThum\
bShape(
                        enabledThumbRadius: 15.0,
                        ),
                        overlayShape: const RoundSliderOv\
erlayShape(
                        overlayRadius: 35.0,
                        ),
                    ),
                    child: Slider(
                        value: height.toDouble(),
                        min: 0.0,
                        max: 100.0,
                        activeColor: activeColor,
                        inactiveColor: Colors.black26,
                        onChanged: (double newValue) {
                        setState(() {
                            height = newValue.round();
                        });
                        },
                    ),
                    ),
                ],
                ),
            ),
            ),
        ],
        ),
    ),
    );
}

Expanded expandEnum(
    ContainerColor? containerColor, String gender, IconDa\
ta genderIcon) {
    return Expanded(
    child: Padding(
        padding: const EdgeInsets.all(18.0),
        child: GestureDetector(
        onTap: () {
            setState(() {
            selectedContainer = containerColor;
            });
        },
        child: Container(
            alignment: Alignment.center,
            color: selectedContainer == containerColor
                ? HappyTheme.activeCoor
                : HappyTheme.inactiveCoor,
            width: 150.0,
            height: 100.0,
            child: Column(
            children: [
                Icon(
                genderIcon,
                size: 80.0,
                ),
                Text(
                gender,
                style: HappyTheme.genderStyle,
                ),
            ],
            ),
        ),
        ),
    ),
    );
}
}
```

However, in the immediate parent Widget, we have defined the MaterialApp theme property also.

Where we have created the custom theme data object.

```
import 'package:flutter/material.dart';
import 'package:happiness_calculator/model/happ_theme.dar\
t';

import 'happiness_home_page.dart';

HappyTheme happyTheme = HappyTheme();

class HappinessApp extends StatelessWidget {
const HappinessApp({Key? key}) : super(key: key);

// This widget is the root of your application.
@override
Widget build(BuildContext context) {
    return MaterialApp(
    title: 'Flutter Demo',
    theme: happyTheme.buildTheme(),
    home: const HappinessHomePage(title: 'Flutter Happine\
ss Calculator'),
    );
}
}
```

If you want to take a closer look at how code structure works, you can clone the whole project from this GitHub repository branch.

Anyway, since we have discussed the usage of custom theme class, we are not going to chew over the same topic.

Instead, we can examine how the Slider Theme Widget works.

How SliderTheme works

The SliderTheme Widget acts as a parent class to the Slider Widget. It applies a slider theme to the descendant Slider widgets.

What is a slider theme?

It describes many properties that manage the theme, such as color, and shape. However, we need to use the SliderTheme.of method that passes the context.

And, as a result, we can access the copyWith() method. Consequently the copyWith() method has many properties that expect the slider components.

Let us take a look at the code snippet.

```
SliderTheme(
                    data: SliderTheme.of(context).copyWit\
h(
                        inactiveTrackColor: HappyTheme.sh\
rinePink50,
                        activeTrackColor: HappyTheme.shri\
neBackgroundWhite,
                        thumbColor: HappyTheme.shrineErro\
rRed,
                        overlayColor: HappyTheme.shrinePi\
nk50,
                        thumbShape: const RoundSliderThum\
bShape(
                        enabledThumbRadius: 15.0,
                        ),
                        overlayShape: const RoundSliderOv\
erlayShape(
                        overlayRadius: 35.0,
                        ),
                    ),
                    child: Slider(
                        value: height.toDouble(),
                        min: 0.0,
                        max: 100.0,
                        activeColor: activeColor,
                        inactiveColor: Colors.black26,
                        onChanged: (double newValue) {
                        setState(() {
                          height = newValue.round();
                        });
                        },
                    ),
                    ),
```

As you see, we have defined various properties that describe the theme properties in detail.

In fact, now it becomes easy to customize the Slider Widget.

In the next section we will proceed forward to complete the App.

Router API in Flutter

What is Navigation in Flutter? It is a routing mechanism. As a result, in a Flutter App, we go from one page to another page.

However, to accomplish this task, we need the Navigator Widget. in addition, we can also go from one page to another page by using the Router Widget.

Then, you may ask, what is the difference? Why not there is only one mechanism?

Because it depends on its complexity. As the Flutter App becomes more elaborate, or there are more pages to go and come back, the Navigation becomes tedious.

Then we need the Router API.

But, for simple App like the Happiness Calculator App that we have been building step by step, the Navigator API is enough.

What is navigation in Flutter Consider a simple example. Suppose there is a bread on the table.

We place a second bread on top of the first bread. Next, we add the third bread on top of the second bread.

Therefor, there are three breads one above the other.

Now, someone comes and pick up the third bread from the stack of three breads.

What will happen?

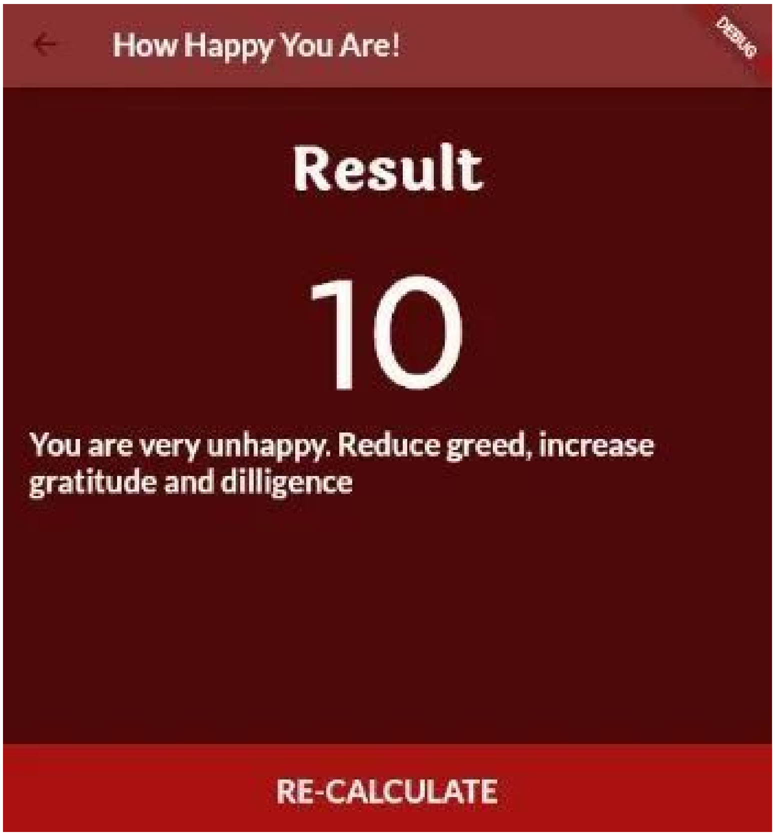
The second bread will come on the top.

If we imagine Flutter pages in place of those breads, the same thing happens. The home page from where we start our journey, is our first bread.

As we keep adding pages, they place themselves one above the other.

When we use the imperative API Navigator.push, it takes us from the home page to the second page. On the contrary, when want to come back to the home page again, we use the Navigator.pop method.

Let us see how it looks. The image will say the thousands words.



Result

10

You are very unhappy. Reduce greed, increase gratitude and dilligence

Figure 5.6 – Navigation Flutter_ Home page of Happiness App

This is the home page of our Happiness Calculator App. The App will count the index of Happiness by taking inputs from three factors.

When the user sets the values as we are seeing above, the result will show up in the next page.

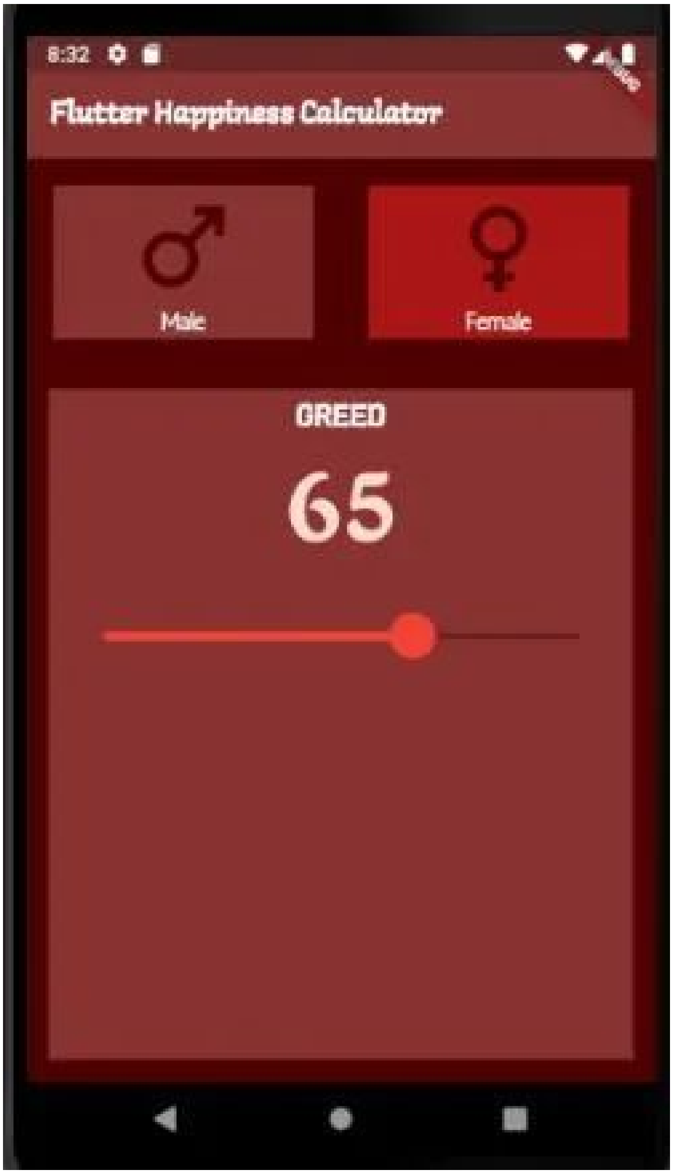


Figure 5.7 – Navigation Flutter_ result page

This Happiness Calculator App works on simple principle. Between the three factors, when greed is too high, and other two factors, gratitude and diligence are low, the happiness index is also low,

However, if a user has low level of greed and high level of gratitude and diligence, then the user becomes a happy person.

What is Navigator.push?

During building this Flutter App, we have learned a few key concepts, such as enum, and ternary operator.

After that, we have learned to use the Flutter Slider Widget and also learned how to customise it.

Similarly, in this section, we will learn the simple navigation where we will go from the home page to the result page.

Let us take a look at the code of home page firstly. Because, we had seen our App as follows before.

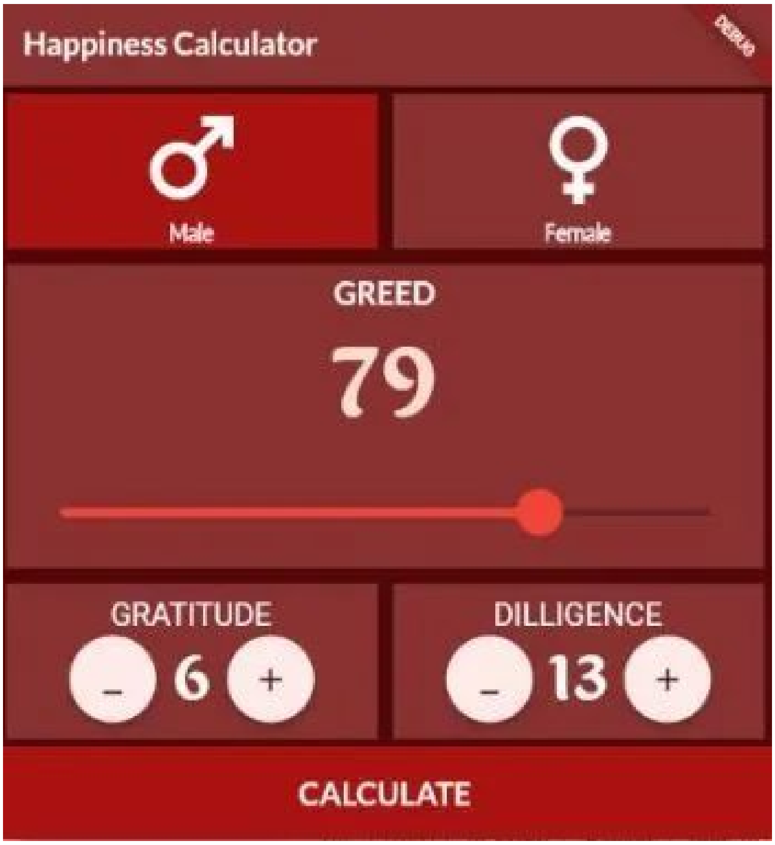
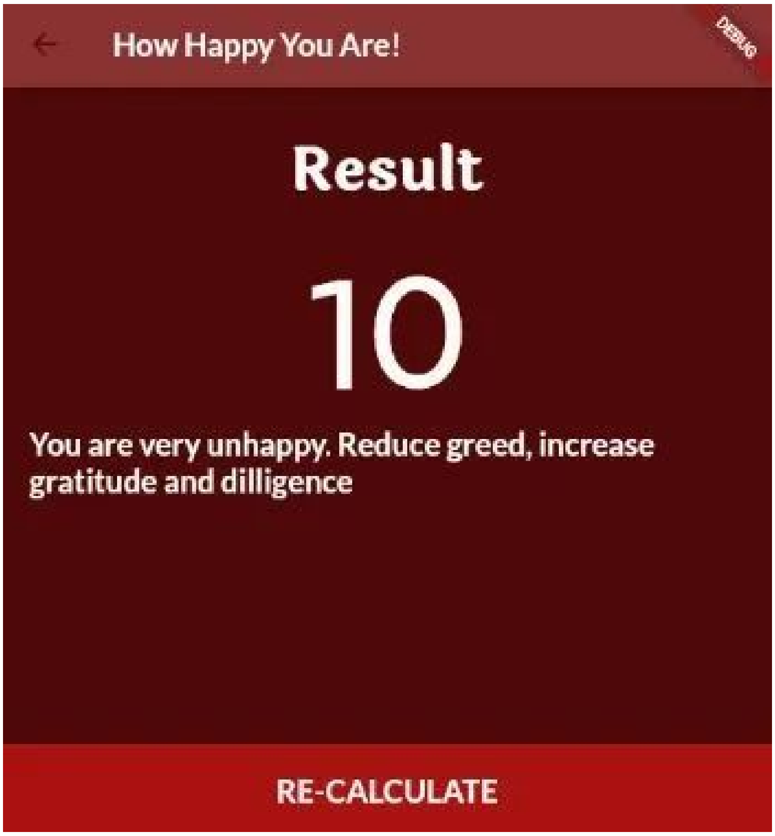


Figure 5.8 – Customized Slider Theme that runs across the entire Flutter App

We have added two more factors below the Slider Widget.

As a result, the code of Homepage has changed a lot.

In addition, the design has changed completely. And, now it looks like below.



Result

10

You are very unhappy. Reduce greed, increase gratitude and dilligence

Figure 5.9 – Navigation Flutter_ Home page of Happiness App

To clarify, it would not happen if we did not change the code of home page.

Let us see the code, first. Then we will discuss how we have added the Navigation mechanism in this page.

```
import 'package:flutter/material.dart';
import 'package:happiness_calculator/view/happiness_resul\
t.dart';

import '../model/happy_theme.dart';
import '../model/constants.dart';
import '../model/container_color.dart';

class HappinessHomePage extends StatefulWidget {
const HappinessHomePage({
    Key? key,
    required this.title,
}) : super(key: key);

final String title;

@override
State<HappinessHomePage> createState() => _HappinessHomeP\
ageState();
}

class _HappinessHomePageState extends State<HappinessHome\
Page> {
ContainerColor? selectedContainer;
int greed = 20;
int gratitude = 10;
int dilligence = 20;

@override
Widget build(BuildContext context) {
    return Scaffold(
    backgroundColor: HappyTheme.shrineBrown900,
    appBar: AppBar(
        title: Text(
        widget.title,
        style: HappyTheme.appbarStyle,
        ),
        backgroundColor: HappyTheme.shrineBrown600,
    ),
    body: Center(
        child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
            Row(
            mainAxisAlignment: MainAxisAlignment.center,
            mainAxisSize: MainAxisSize.min,
            children: [
                expandGender(
                ContainerColor.first,
                'Male',
                Icons.male,
                ),
                expandGender(
                ContainerColor.second,
                'Female',
                Icons.female,
                ),
            ],
            ),
            Expanded(
            child: Container(
                margin: const EdgeInsets.all(5.0),
                alignment: Alignment.center,
                color: HappyTheme.shrineBrown600,
                width: double.infinity,
                child: Column(
                mainAxisAlignment: MainAxisAlignment.cent\
er,
                mainAxisSize: MainAxisSize.min,
                children: [
                    Container(
                    padding: const EdgeInsets.only(
                        top: 5.0,
                        bottom: 2.0,
                    ),
                    child: Text(
                        'GREED',
                        style: HappyTheme.appbarStyle,
                    ),
                    ),
                    Row(
                    mainAxisAlignment: MainAxisAlignment.\
center,
                    crossAxisAlignment: CrossAxisAlignmen\
t.baseline,
                    textBaseline: TextBaseline.alphabetic,
                    children: [
                        Text(
                        greed.toString(),
                        style: HappyTheme.greedStyle,
                        ),
                    ],
                    ),
                    SliderTheme(
                    data: SliderTheme.of(context).copyWit\
h(
                        inactiveTrackColor: HappyTheme.sh\
rinePink50,
                        activeTrackColor: HappyTheme.shri\
neBackgroundWhite,
                        thumbColor: HappyTheme.shrineErro\
rRed,
                        overlayColor: HappyTheme.shrinePi\
nk50,
                        thumbShape: const RoundSliderThum\
bShape(
                        enabledThumbRadius: 15.0,
                        ),
                        overlayShape: const RoundSliderOv\
erlayShape(
                        overlayRadius: 35.0,
                        ),
                    ),
                    child: Slider(
                        value: greed.toDouble(),
                        min: 20.0,
                        max: 100.0,
                        activeColor: activeColor,
                        inactiveColor: Colors.black26,
                        onChanged: (double newValue) {
                        setState(() {
                            greed = newValue.round();
                        });
                        },
                    ),
                    ),
                ],
                ),
            ),
            ),
            Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
                expandGratitude(),
                expandDilligence(),
            ],
            ),
        ],
        ),
    ),
    bottomNavigationBar: Container(
        width: double.infinity,
        height: 60.0,
        color: HappyTheme.activeCoor,
        child: TextButton(
        onPressed: () {
            Navigator.push(
            context,
            MaterialPageRoute(
                builder: (context) => const HappinessResu\
lt(),
            ),
            );
        },
        child: Text(
            'CALCULATE',
            style: HappyTheme.appbarStyle,
        ),
        ),
    ),
    );
}

Expanded expandDilligence() {
    return Expanded(
    child: Padding(
        padding: const EdgeInsets.all(5.0),
        child: Container(
        alignment: Alignment.center,
        width: double.infinity,
        height: 100.0,
        color: HappyTheme.inactiveCoor,
        child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            mainAxisSize: MainAxisSize.min,
            children: [
            const Text(
                'DILLIGENCE',
                style: TextStyle(
                fontSize: 20.0,
                color: Colors.white,
                ),
            ),
            Row(
                mainAxisAlignment: MainAxisAlignment.spac\
eAround,
                mainAxisSize: MainAxisSize.min,
                children: [
                FloatingActionButton(
                    heroTag: 'btn3',
                    onPressed: () {
                    setState(() {
                        dilligence--;
                    });
                    },
                    child: const Icon(
                    Icons.minimize,
                    ),
                ),
                Container(
                    padding: const EdgeInsets.only(
                    left: 10.0,
                    right: 10.0,
                    ),
                    child: Text(
                    dilligence.toString(),
                    style: HappyTheme.dilligenceStyle,
                    ),
                ),
                FloatingActionButton(
                    heroTag: 'btn4',
                    onPressed: () {
                    setState(() {
                        dilligence++;
                    });
                    },
                    child: const Icon(
                    Icons.add,
                    ),
                ),
                ],
            ),
            ],
        ),
        ),
    ),
    );
}

Expanded expandGratitude() {
    return Expanded(
    child: Padding(
        padding: const EdgeInsets.all(5.0),
        child: Container(
        alignment: Alignment.center,
        width: double.infinity,
        height: 100.0,
        color: HappyTheme.inactiveCoor,
        child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            mainAxisSize: MainAxisSize.min,
            children: [
            const Text(
                'GRATITUDE',
                style: TextStyle(
                fontSize: 20.0,
                color: Colors.white,
                ),
            ),
            Row(
                mainAxisAlignment: MainAxisAlignment.spac\
eAround,
                mainAxisSize: MainAxisSize.min,
                children: [
                FloatingActionButton(
                    heroTag: 'btn1',
                    onPressed: () {
                    setState(() {
                        gratitude--;
                    });
                    },
                    child: const Icon(
                    Icons.minimize,
                    ),
                ),
                Container(
                    padding: const EdgeInsets.only(
                    left: 10.0,
                    right: 10.0,
                    ),
                    child: Text(
                    gratitude.toString(),
                    style: HappyTheme.dilligenceStyle,
                    ),
                ),
                FloatingActionButton(
                    heroTag: 'btn2',
                    onPressed: () {
                    setState(() {
                        gratitude++;
                    });
                    },
                    child: const Icon(
                    Icons.add,
                    ),
                ),
                ],
            ),
            ],
        ),
        ),
    ),
    );
}

Expanded expandGender(
    ContainerColor? containerColor, String gender, IconDa\
ta genderIcon) {
    return Expanded(
    child: Padding(
        padding: const EdgeInsets.all(5.0),
        child: GestureDetector(
        onTap: () {
            setState(() {
            selectedContainer = containerColor;
            });
        },
        child: Container(
            alignment: Alignment.center,
            color: selectedContainer == containerColor
                ? HappyTheme.activeCoor
                : HappyTheme.inactiveCoor,
            width: double.infinity,
            height: 100.0,
            child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            mainAxisSize: MainAxisSize.min,
            children: [
                Icon(
                genderIcon,
                size: 80.0,
                color: Colors.white,
                ),
                Text(
                gender,
                style: HappyTheme.genderStyle,
                ),
            ],
            ),
        ),
        ),
    ),
    );
}
}
```

Now, we can either make any value lower, or make it higher. Meanwhile, we can press the "CALCULATE" button, and it takes us to the result page.

In the bottom navigation bar property, in a Text Button we have used the Navigator.push method.

```
bottomNavigationBar: Container(
        width: double.infinity,
        height: 60.0,
        color: HappyTheme.activeCoor,
        child: TextButton(
        onPressed: () {
            Navigator.push(
            context,
            MaterialPageRoute(
                builder: (context) => const HappinessResu\
lt(),
            ),
            );
        },
        child: Text(
            'CALCULATE',
            style: HappyTheme.appbarStyle,
        ),
        ),
    ),
```

Certainly, the Happiness result page displays a dummy result. Because we have only finished the design part. Besides, we make an easy navigation defining the route.

As we see in the above code, Navigator.push method passes two parameters.

The first parameter is the context. And the second parameter is the MaterialPageRoute class constructor.

In this constructor the builder property expects that the context will return the second page.

What is Navigtor.pop?

Let us go back to the previous examples. The Navigator.push method basically puts one page over the other.

However, the Navigator.pop method does the opposite. It takes out the page from the top.

As a result, the page below will again emerge.

This is the most basic routing mechanism where we go to one page, and come back to the home page.

Here is the result page code that will explain how this routing mechanism works.

```
import 'package:flutter/material.dart';
import 'package:happiness_calculator/model/happy_theme.da\
rt';

class HappinessResult extends StatelessWidget {
const HappinessResult({Key? key}) : super(key: key);

@override
Widget build(BuildContext context) {
    return Scaffold(
    backgroundColor: HappyTheme.shrineBrown900,
    appBar: AppBar(
        backgroundColor: HappyTheme.shrineBrown600,
        title: Text(
        'How Happy You Are!',
        style: HappyTheme.appbarStyle,
        ),
    ),
    body: Padding(
        padding: const EdgeInsets.all(18.0),
        child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        mainAxisSize: MainAxisSize.min,
        children: [
            Container(
            margin: const EdgeInsets.all(5.0),
            child: Text(
                'Result',
                style: HappyTheme.resultStyle,
            ),
            ),
            Container(
            width: double.infinity,
            padding: const EdgeInsets.only(
                top: 10.0,
            ),
            child: Column(
                mainAxisAlignment: MainAxisAlignment.cent\
er,
                mainAxisSize: MainAxisSize.min,
                children: [
                Text(
                    '10',
                    style: HappyTheme.happinessIndexStyle,
                ),
                Text(
                    'You are very unhappy. Reduce greed, \
increase gratitude and dilligence',
                    style: HappyTheme.happinessResultStyl\
e,
                ),
                ],
            ),
            ),
        ],
        ),
    ),
    bottomNavigationBar: Container(
        width: double.infinity,
        height: 60.0,
        color: HappyTheme.activeCoor,
        child: TextButton(
        onPressed: () {
            Navigator.pop(context);
        },
        child: Text(
            'RE-CALCULATE',
            style: HappyTheme.appbarStyle,
        ),
        ),
    ),
    );
}
}
```

In the same vein, we use the bottom navigation bar, and a Text Button.

In addition, we define the routing mechanism by using Navigator.pop method.

```
onPressed: () {
            Navigator.pop(context);
        },
```

As we press the bottom navigation bar button, It picks up the result page. Therefore, the home page emerges from the below.

If you want to clone this step, please visit this branch of the GitHub repository.

Certainly, this routing mechanism is simple. Moreover, it has not sent any data from the home page to the result page.

But we can do that.

In fact, in the next section we will pass the data from home page to the result page.

And based on that data, we will calculate the Happiness Index.

Business Logic behind an Flutter App

We have reached the final stage. So we will use the Flutter Business Logic and finish our Happiness Calculator App.

While building the Flutter App, we have learned a few key concepts.

Firstly, we have learned how to use enum. Secondly, we have learned ternary operator. Thirdly, we have grasped how to customize the Slider theme.

Further, we have absorbed the basic route mechanism.

If you are a beginner, please follow the steps and clone the GitHub repository. In fact, you will find them in the step-by-step branches.

Please clone them and test in your local machine.

What is the difference between Business Logic and UI Logic?

Flutter is a cross-platform Framework or UI toolkit. What does that mean? It means the user interface plays an important role.

Each time Flutter calls the build() method the old instance of Widget destroys itself. As a result, the Widget rebuilds itself and a new instance shows up.

Flutter mostly handles the UI logic. Therefore, we do not have to bother. We just extends either Stateless, or a Stateful Widget. And, we move on.

Of course, Flutter allows us to create our own Widget. Quite naturally. Because Flutter is open source and we get the source code in GitHub.

However, we need to create our Business Logic. As it happens in our Happiness Calculator App.

In the previous section we have seen that the Result page displays some static Text output. Right?

Take a look at the last stage where we have left.

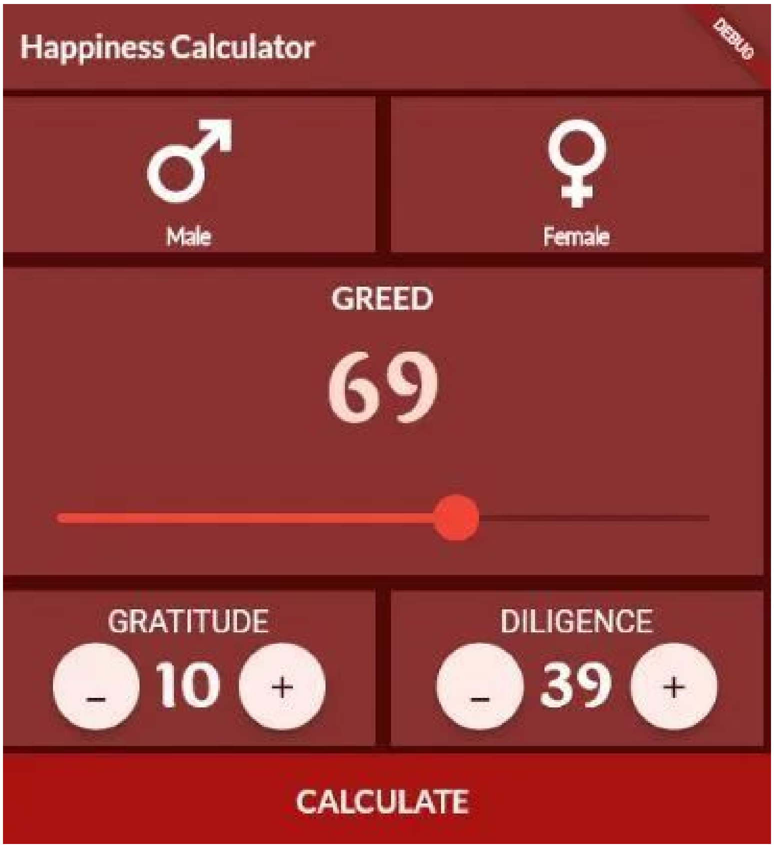


Figure 5.10 – Navigation Flutter_ result page

The above page does not reflect the dynamic properties that should have come from the home page.

A user will set the "greed" factor to a number. In respect to that she will also have to choose two other factors, such as "gratitude" and "diligence".

Subsequently, our home page will store those values somewhere and our business logic will decide whether the user is happy or unhappy.

Moreover, based on that, the Flutter Business Logic will also advise what to do. This mechanism will ensure code separation.

Why?

Because it is important for clean architecture.

How to create Flutter Business Logic?

The algorithm is simple. We have seen that greed has a minimum value 20. Because we all come to this planet with some in-built greed. The maximum value is 100.

The same is true for other two factors. The "gratitude" and "diligence" also have some minimum values. Subsequently, the user can increase or decrease the value.

Here, our business logic is simple. Firstly, we combine the values of the "gratitude" and "diligence". Secondly, we subtract the value of "greed" from it.

The result is the happiness index value.

Let us see the code.

```
class HappinessCalculator {
final int? greed;
final int? gratitude;
final int? diligence;

int? _happinessIndex;

HappinessCalculator({
    required this.greed,
    required this.gratitude,
    required this.diligence,
});

String calculateHappiness() {
    _happinessIndex = (gratitude! + diligence!) - greed!;
    return _happinessIndex.toString();
}

String getResult() {
    if (_happinessIndex! <= 50) {
    return 'Unhappy';
    } else {
    return 'Happy';
    }
}

String getAdvice() {
    if (_happinessIndex! <= 50) {
    return 'Please reduce greed and increase gratitude an\
d dligence.';
    } else {
    return 'You are a Happy person. Study old philosopher\
s, and stay cool.';
    }
}
}
```

In the above code, if happiness index is equal and lower than 50, you are unhappy. However, if the happiness index is greater than 50, then you are happy.

The real challenge is elsewhere.

We need to pass these values to the result page.

How to pass values through Navigation?

In our previous section we have finished the design part and learned how to navigate to the result page.

Therefore, we are not going to repeat the full code of the Happiness App home page.

We will take a look at the bottom navigation bar part where we have handled the business logic.

```
bottomNavigationBar: Container(
        width: double.infinity,
        height: 60.0,
        color: HappyTheme.activeCoor,
        child: TextButton(
        onPressed: () {
            HappinessCalculator happy = HappinessCalculat\
or(
            greed: greed,
            gratitude: gratitude,
            diligence: diligence,
            );
            happy.calculateHappiness();
```

```
                Navigator.push(
                context,
                MaterialPageRoute(
                    builder: (context) => HappinessResult(
                    greed: greed,
                    gratitude: gratitude,
                    diligence: diligence,
                    happinessIndex: happy.getResult(),
                    whatIsToBeDone: happy.getAdvice(),
                    ),
                ),
                );
            },
            child: Text(
                'CALCULATE',
                style: HappyTheme.appbarStyle,
            ),
            ),
        ),
```

In the above code, what do we see?

Firstly, we have created a HappinessCalculator object. After that, we call the calculateHappiness() method that returns the happiness index.

```
String calculateHappiness() {
    _happinessIndex = (gratitude! + diligence!) - greed!;
    return _happinessIndex.toString();
}
```

Now, we can pass that value to the result page through class constructor.

```
MaterialPageRoute(
                builder: (context) => HappinessResult(
                greed: greed,
                gratitude: gratitude,
                diligence: diligence,
                happinessIndex: happy.getResult(),
                whatIsToBeDone: happy.getAdvice(),
                ),
            ),
            );
```

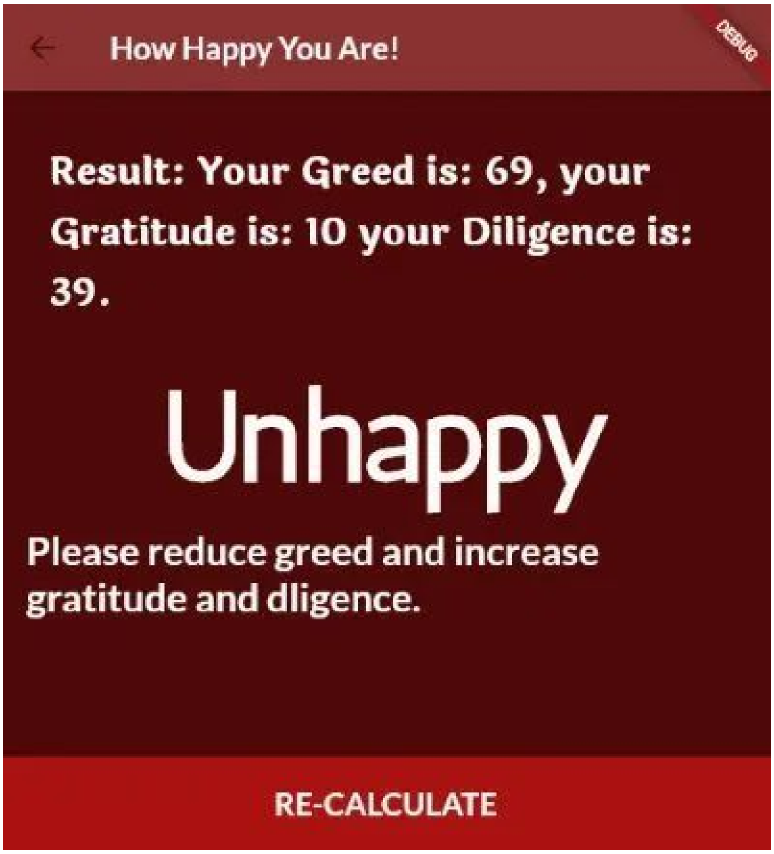
For that reason, now it becomes easier for us to display these values in the result page.

Suppose the user has more greed and less gratitude and diligence.

The home page will look as follows.

When you have more greed, it overshadows every virtue.

As a result, the user becomes unhappy. But at the same time, we can see the values on the result page.



Result: Your Greed is: 69, your Gratitude is: 10 your Diligence is: 39.

Unhappy

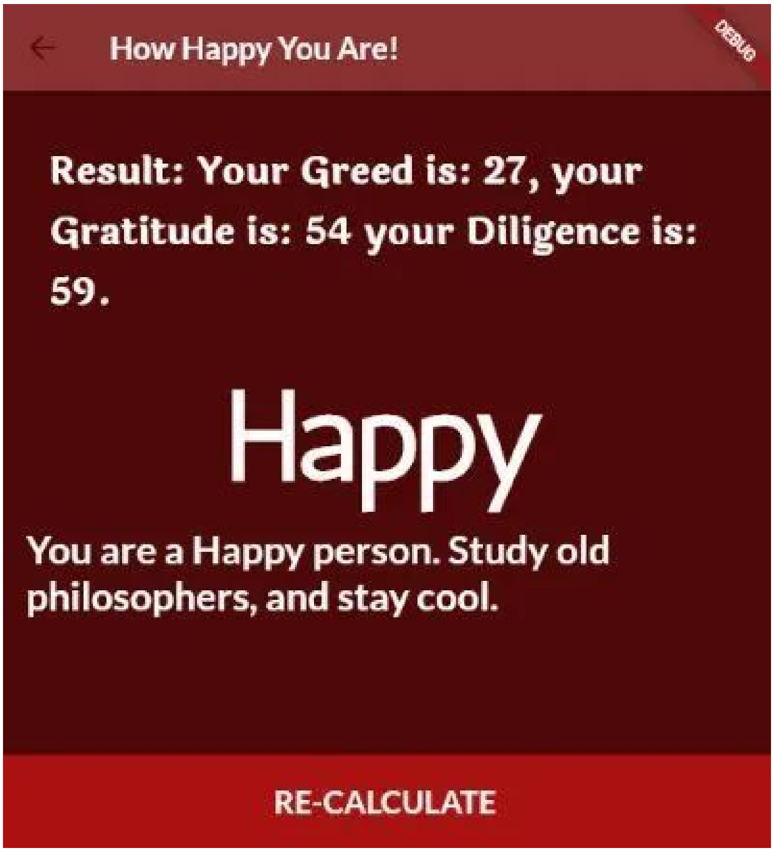
Please reduce greed and increase gratitude and dligence.

Figure 5.11 – Flutter Business Logic calculates and finds the reasons of unhappiness

The result clearly shows why the user is unhappy. The reason is simple. The user has greed 69, and on the contrary, gratitude is 10, and diligence is 39.

However, if the user has less greed but more gratitude and diligence?

The result page changes the values and displays them.



Result: Your Greed is: 27, your Gratitude is: 54 your Diligence is: 59.

Happy

You are a Happy person. Study old philosophers, and stay cool.

Figure 5.12 – Flutter Business Logic to calculates and finds out why the user is happy

How we can display data got from Business Logic?

It is not at all difficult. On the contrary, it is quite simple.

Because Flutter framework allows us to pass those data through class constructor.

Let us take a look at the result page code.

```dart
import 'package:flutter/material.dart';
import 'package:happiness_calculator/model/happy_theme.da\
rt';

class HappinessResult extends StatelessWidget {
const HappinessResult({
    Key? key,
    required this.greed,
    required this.gratitude,
    required this.diligence,
    required this.happinessIndex,
    required this.whatIsToBeDone,
}) : super(key: key);
final int greed;
final int gratitude;
final int diligence;

final String happinessIndex;
final String whatIsToBeDone;

@override
Widget build(BuildContext context) {
    return Scaffold(
    backgroundColor: HappyTheme.shrineBrown900,
    appBar: AppBar(
        backgroundColor: HappyTheme.shrineBrown600,
        title: Text(
        'How Happy You Are!',
        style: HappyTheme.appbarStyle,
        ),
    ),
    body: Padding(
        padding: const EdgeInsets.all(18.0),
        child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        mainAxisSize: MainAxisSize.min,
        children: [
            Container(
            margin: const EdgeInsets.all(15.0),
            child: Text(
                'Result: Your Greed is: $greed, '
                'your Gratitude is: $gratitude '
                'your Diligence is: $diligence.',
                style: HappyTheme.resultStyle,
            ),
            ),
            Container(
            width: double.infinity,
            padding: const EdgeInsets.only(
                top: 10.0,
            ),
            child: Column(
                mainAxisAlignment: MainAxisAlignment.cent\
er,
                mainAxisSize: MainAxisSize.min,
                children: [
                Text(
                    happinessIndex,
                    style: HappyTheme.happinessIndexStyle,
                ),
                const SizedBox(
                    height: 10.0,
                ),
                Text(
                    whatIsToBeDone,
                    style: HappyTheme.happinessResultStyl\
e,
                ),
                ],
            ),
            ),
        ],
        ),
    ),
    bottomNavigationBar: Container(
        width: double.infinity,
        height: 60.0,
        color: HappyTheme.activeCoor,
        child: TextButton(
        onPressed: () {
            Navigator.pop(context);
        },
        child: Text(
            'RE-CALCULATE',
            style: HappyTheme.appbarStyle,
        ),
        ),
    ),
    );
}
}
```

The above code shows us how the result page gets the values through class constructors.

In addition, we can display them in Text Widgets. How? Because in our business logic, we return String data type.

For brevity, we have avoided to show full code. But, if you want to clone the final project, please visit this GitHub rpository.

- For full code snippet for this app, please visit the respective GitHub Repository - [18]

[18]https://github.com/sanjibsinha/happiness_calculator

6. How we can build a Food Recipe App with List and Map

GridView is a scrollable Widget in Flutter. It places list of elements side by side. In this section we will learn how to use the GridView while we build a recipe app. We call it “CooKingKong”.

Certainly it will not be a serious cooking lesson. Therefore do not expect to be a expert cook after finishing this module.

Let us know how this module will go. What we are going to create.

Firstly, we will have a page of Categories as follows.

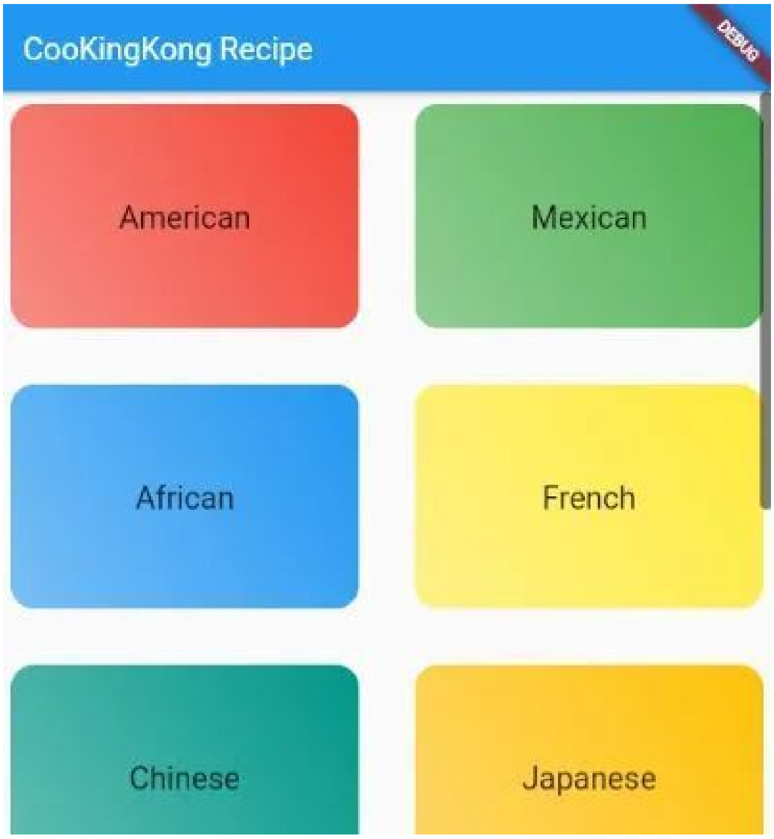


Figure 6.1 – GridView Flutter Recipe app first page

Secondly, when we click each Category, it will take us to a page where we will find many recipes belonging to the same category.

Finally, as we click any individual item there, that will open another page where we will find the detail of the recipe.

In our previous Flutter App, "Happiness Calculator", we have seen how we can use simple Navigation.

However, this time it will be a little different.

What is GridView Flutter and How it works

As we have said earlier, our categories page will display a List of categories.

Therefore, we need a Widget that will help us to accomplish that task. Right?

The GridView in Flutter does the same job.

It has a property called children. Subsequently, the children property expects a List of Widgets.

Let us take a look at how the GridView in Flutter looks like.

```dart
import 'package:flutter/material.dart';
import 'package:coo_king_kong/model/dummy_categories.dart\
';
import 'package:coo_king_kong/view/category_item.dart';

class CategoriesPage extends StatelessWidget {
const CategoriesPage({Key? key}) : super(key: key);

@override
Widget build(BuildContext context) {
    return GridView(
    children: dummyCategories
        .map(
            (e) => CategoryItem(title: e.title, color: e.\
color),
        )
        .toList(),
    gridDelegate: const SliverGridDelegateWithMaxCrossAxi\
sExtent(
        maxCrossAxisExtent: 300,
        childAspectRatio: 1.50,
        mainAxisSpacing: 20.0,
        crossAxisSpacing: 20.0,
    ),
    );
}
}
```

Our top level function main() which is also the entry point, runs the "CooKingKong" App. As if you will be cooking as a King Kong. However, there is no opponent like Godzilla.

Our Widget tree runs as follows.

```
import 'package:flutter/material.dart';
import 'view/coo_king_kong_app.dart';

void main() {
runApp(const CooKingKongApp());
}

class CooKingKongApp extends StatelessWidget {
const CooKingKongApp({Key? key}) : super(key: key);

// This widget is the root of your application.
@override
Widget build(BuildContext context) {
    return MaterialApp(
    title: 'Flutter Demo',
    theme: ThemeData(
        primarySwatch: Colors.blue,
    ),
    home: const CooKingKongHome(),
    );
}
}

```

```
class CooKingKongHome extends StatelessWidget {
const CooKingKongHome({Key? key}) : super(key: key);

@override
Widget build(BuildContext context) {
    return Scaffold(
    appBar: AppBar(
        title: const Text('CooKingKong Recipe'),
    ),
    body: const CategoriesPage(),
    );
}
}
```

In the above Widget tree, the last one is the CategoriesPage() which we have seen before.

But one Widget is still missing.

Because the build() method of the CategoriesPage() returns a GridView.

And, as a result, the children property of the GridView expects a List of Widgets.

That means we have to return all the categories through a class constructor as follows.

```
import 'package:flutter/material.dart';

class CategoryItem extends StatelessWidget {
const CategoryItem({
    Key? key,
    required this.title,
    required this.color,
}) : super(key: key);

final String title;
final Color color;

@override
Widget build(BuildContext context) {
    return Container(
    margin: const EdgeInsets.all(8.0),
    child: Center(
        child: Text(
        title,
        style: const TextStyle(
            fontSize: 20.0,
        ),
        textAlign: TextAlign.center,
        ),
    ),
    decoration: BoxDecoration(
        gradient: LinearGradient(
        colors: [
            color.withOpacity(0.6),
            color,
        ],
        begin: Alignment.bottomLeft,
        end: Alignment.topRight,
        ),
        borderRadius: BorderRadius.circular(15.0),
    ),
    );
}
}
```

The above image displays this page where we have designed how each Container will look like.

We have used the decoration property of the Container class. And, in addition,the decoration property expects a BoxDecoration class.

We have discussed how we can decorate a Container before.

For example, there are other types of GridView also. There are GridView.builder, GridView.extent, and GridView.count.

We have discussed them in detail. Have a look to realise how Flutter helps us to solve problems in many ways.

Anyway, we have built the first step of CooKIngKong App. Although we have not discussed another important aspect of this module.

How did we get the List items? How each list Item has a distinct color?

Let us discuss that part as well.

Model folder is the source of data

We are following the MVC principle. Therefore, we are trying to separate data source from our Business and UI Logic.

As a result, in the Model folder, we have a Category class that has three properties.

```
import 'package:flutter/material.dart';

// this is first-step
class Category {
final String id;
final String title;
final Color color;

const Category({
    required this.id,
    required this.title,
    this.color = Colors.deepOrange,
});
}
```

However, without a dummy category this Category class is meaningless.

For that reason, we have a List of dummy categories.

```
import 'package:flutter/material.dart';
import 'category.dart';

// this is first-step
const dummyCategories = [
Category(id: 'c1', title: 'American', color: Colors.red),
Category(id: 'c2', title: 'Mexican', color: Colors.green),
Category(id: 'c3', title: 'African', color: Colors.blue),
Category(id: 'c4', title: 'French', color: Colors.yellow),
Category(id: 'c5', title: 'Chinese', color: Colors.teal),
Category(id: 'c6', title: 'Japanese', color: Colors.amber\
),
Category(id: 'c7', title: 'Indian', color: Colors.pink),
Category(id: 'c8', title: 'Iranian', color: Colors.black1\
2),
Category(id: 'c9', title: 'German', color: Colors.purple),
Category(id: 'c10', title: 'Italian', color: Colors.red),
];
```

Remember the CategoriesPage Widget. The children property uses this dummy data through the CategoryItem Widget constructor.

```
return GridView(
    children: dummyCategories
        .map(
            (e) => CategoryItem(title: e.title, color: e.\
color),
        )
        .toList(),
    ...
```

We need to understand one principle concept here.

The data source could be local. And in our case, that happens. We are using a local data source that comes from this dummy categories.

This is the simplest form. We cannot make it simpler than that.

For instance, when data comes from the backend data base, it no longer remains so simple.

With reference to that, we have discussed SQLite database and Flutter in great detail before. You may have a look.

So far, we have successfully built the first part of our "CooK-IngKong" App.

If you wan to clone this part of Code, please visit this branch of GitHub repository.

(You can clone this branch)[https://github.com/sanjibsinha/coo_-king_kong/tree/first-step-cookingkong]

How to use Naviagtion to send data

We have been building a funny recipe app in Flutter. In the first part, we have built a categories page. But in a very simple way. Now we want to use Flutter navigation to send data.

That means when user clicks a category, it takes her to that category page.

As a result, she can press the back button, and come back.

Let us see where we had left before.

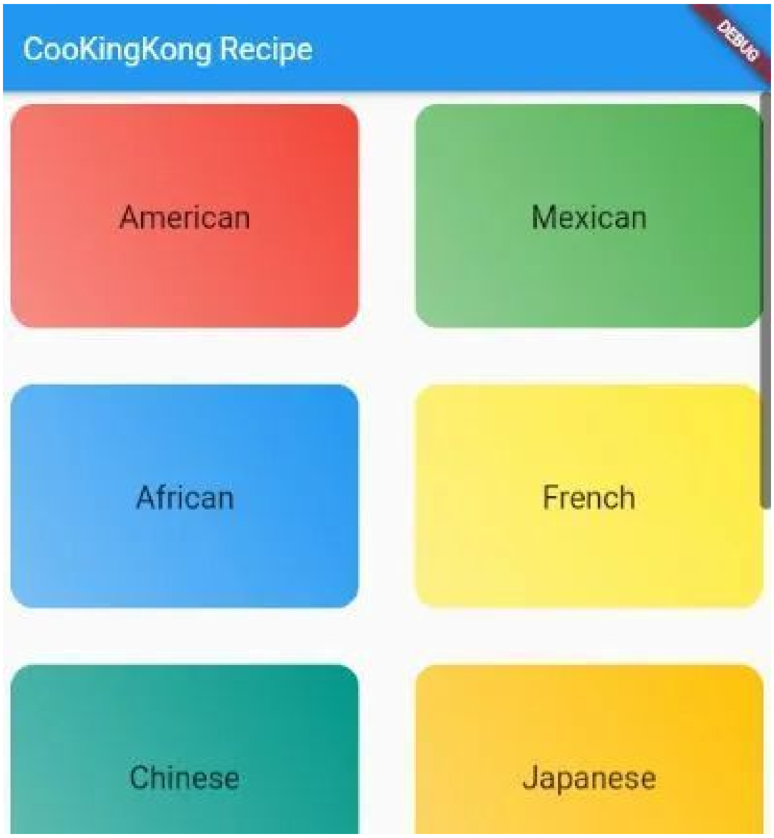


Figure 6.2 – GridView Flutter Recipe app first page

As we can see, we had not used any custom theme. Although we can use custom font. In addition, we can apply a custom theme across the app.

How to add a custom Font

Firstly, we can use Google font package for Flutter. It's a convenient way to different fonts across the app.

In fact, in a previous article, we have discussed that.

But this time we will download Google fonts. After that, we will add the fonts in pubspec.yaml file as follows.

```
fonts:
    - family: Raleway
    fonts:
        - asset: assets/fonts/Raleway-Regular.ttf
        - asset: assets/fonts/Raleway-Bold.ttf
        weight: 700
        - asset: assets/fonts/Raleway-Black.ttf
        weight: 900
    - family: RobotoCondensed
    fonts:
        - asset: assets/fonts/RobotoCondensed-Regular.ttf
        - asset: assets/fonts/RobotoCondensed-Bold.ttf
        weight: 700
        - asset: assets/fonts/RobotoCondensed-Light.ttf
        weight: 300
        - asset: assets/fonts/RobotoCondensed-Italic.ttf
        style: italic
```

Next we will customise our theme in MaterialApp Widget. Later we will take that part to a separate class.

Subsequently, we have used our fonts and changed the color across the app.

```
import 'package:flutter/material.dart';

import 'coo_king_kong_home.dart';

class CooKingKongApp extends StatelessWidget {
const CooKingKongApp({Key? key}) : super(key: key);

// This widget is the root of your application.
@override
Widget build(BuildContext context) {
    return MaterialApp(
    title: 'Flutter Demo',
    theme: ThemeData(
        canvasColor: const Color.fromRGBO(255, 254, 229, \
1),
        fontFamily: 'Raleway',
        textTheme: ThemeData.light().textTheme.copyWith(
            bodySmall: const TextStyle(
                color: Color.fromRGBO(20, 51, 51, 1),
            ),
            bodyMedium: const TextStyle(
                color: Color.fromRGBO(20, 51, 51, 1),
            ),
            titleSmall: const TextStyle(
                fontSize: 20,
                fontFamily: 'RobotoCondensed',
                fontWeight: FontWeight.bold,
            ),
            titleMedium: const TextStyle(
                fontSize: 30,
                fontFamily: 'RobotoCondensed',
                fontWeight: FontWeight.bold,
            ),
            titleLarge: const TextStyle(
                fontSize: 40,
                fontFamily: 'RobotoCondensed',
                fontWeight: FontWeight.bold,
            ),
            ),
        colorScheme: ColorScheme.fromSwatch(primarySwatch\
: Colors.pink)
            .copyWith(secondary: Colors.amber),
    ),
    home: const CooKingKongHome(),
    );
}
}
```

As a result, our CooKingKong App will have a different look.

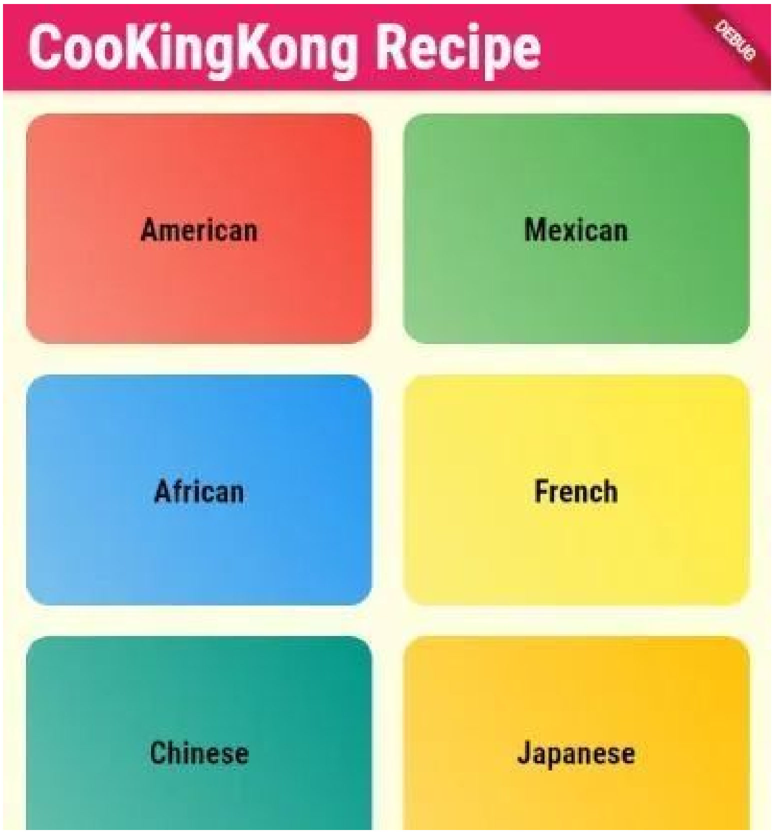


Figure 6.3 – Flutter navigation send data_ categories page

Now the previous look is no longer there. Secondly, we will press each category so that we can go to that particular page.

That is our next challenge.

Flutter navigation and sending data

How does the Flutter navigation work? We have discussed in a previous lesson.

Navigator.push method places one page on top of the other pages.

Navigator.pop method displaces that page, so that the page below can emerge. It's a simple mechanism.

But there are other better ways to handle the same problem.

Why? Because this simple mechanism is okay with simple apps where we deal with one, or two pages.

What happens when there are many pages?

Certainly, we need a better mechanism. We will discuss that topic in our next section.

This time we will only pass data from categories item page to an individual category page.

To do that, we need a Widget that has a tapping facility. So a user can tap any category to see what items it contains.

```
import 'package:flutter/material.dart';

import 'individual_category_page.dart';

/// TODO: this page displays each category by name and co\
lor
/// we will add a method that will take us to individual
/// category page

class CategoryItem extends StatelessWidget {
const CategoryItem({
    Key? key,
    required this.id,
    required this.title,
    required this.color,
}) : super(key: key);

final String id;
final String title;
final Color color;

void selectCategory(BuildContext context) {
    Navigator.of(context).push(
    MaterialPageRoute(
        builder: (context) {
        return IndividualCategoryPage(
            id: id,
            title: title,
            color: color,
        );
        },
    ),
    );
}

@override
Widget build(BuildContext context) {
    return InkWell(
    onTap: () => selectCategory(context),
    splashColor: Theme.of(context).primaryColor,
    borderRadius: BorderRadius.circular(15),
    child: Container(
        //margin: const EdgeInsets.all(8.0),
        child: Center(
        child: Text(
            title,
            style: Theme.of(context).textTheme.titleSmall,
            textAlign: TextAlign.center,
        ),
        ),
        decoration: BoxDecoration(
        gradient: LinearGradient(
            colors: [
            color.withOpacity(0.6),
            color,
            ],
            begin: Alignment.bottomLeft,
            end: Alignment.topRight,
        ),
        borderRadius: BorderRadius.circular(15.0),
        ),
    ),
    );
}
}
```

In the above code, we send data through class constructor. We have defined that mechanism in a separate method.

```
void selectCategory(BuildContext context) {
    Navigator.of(context).push(
    MaterialPageRoute(
        builder: (context) {
        return IndividualCategoryPage(
            id: id,
            title: title,
            color: color,
        );
        },
    ),
    );
}
```

Meanwhile we call that method inside Inkwell Widget.

```
return InkWell(
    onTap: () => selectCategory(context),
...
```

We could have used the GestureDetector Widget. But InkWell gives us more freedom to use the custom theme color.

```
splashColor: Theme.of(context).primaryColor,
```

How a Widget receives data

The mechanism is same. It receives data through class constructor.

Therefore, if we take a look at the individual category page, we can see that it receives data through class constructor.

```
import 'package:flutter/material.dart';

class IndividualCategoryPage extends StatelessWidget {
const IndividualCategoryPage({
    Key? key,
    required this.id,
    required this.title,
    required this.color,
}) : super(key: key);

final String id;
final String title;
final Color color;

@override
Widget build(BuildContext context) {
    return Scaffold(
    appBar: AppBar(
        title: Text(title),
    ),
    body: Center(
        child: Container(
        color: color,
        child: Text(
            title,
            style: Theme.of(context).textTheme.titleLarge,
        ),
        ),
    ),
    );
}
}
```

Now we can click any category, and see the same title in the AppBar, body and the same color.

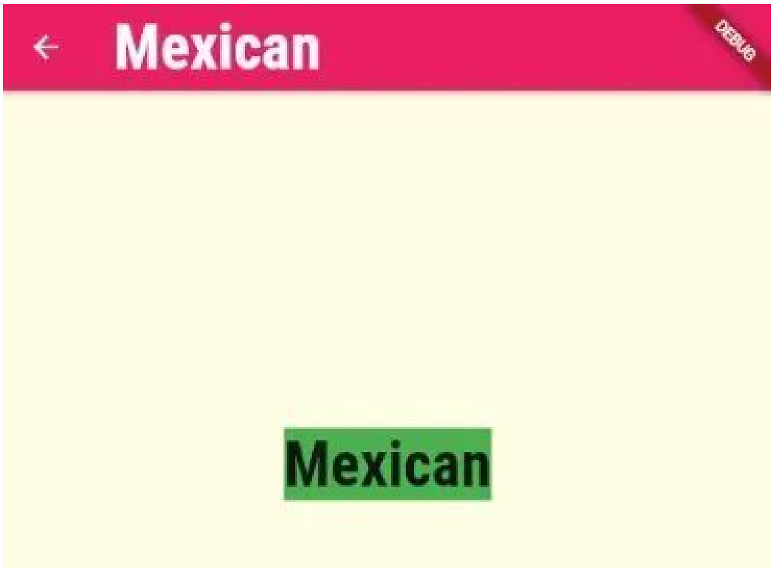


Figure 6.4 – Flutter navigation send data to a page

Finally, we have sent data to a page. However, we have not been able to show everything in this section.

If you want to clone this step please visit the respective branch of GitHub Repository.

In the next section we will discuss the named route, that will make our code cleaner and robust.

(Clone this branch and run locally)[https://github.com/sanjibsinha/coo_-king_kong/tree/second-step-cookingkong-passing-data-via-navigation]

Named Route and sending data

In the previous post we had sent data through Flutter navigation. But we could have done the same thing by Flutter named route.

We will see in a minute, how we can do this.

Firstly, we need to understand why we want Flutter named route. Secondly, we will see how it makes our CooKingKong App more dynamic.

Finally, we will compare every route mechanism in Flutter.

First thing first.

We want Flutter named route for one single reason. That is, for a big application, we can manage many pages in a better way.

The MaterialApp Widget has a “routes” property that expects a Map object.

We have discussed Map earlier. So we have seen that a Map has key and value pair.

But in our case, the value expects a #context object that returns the page where we want to navigate.

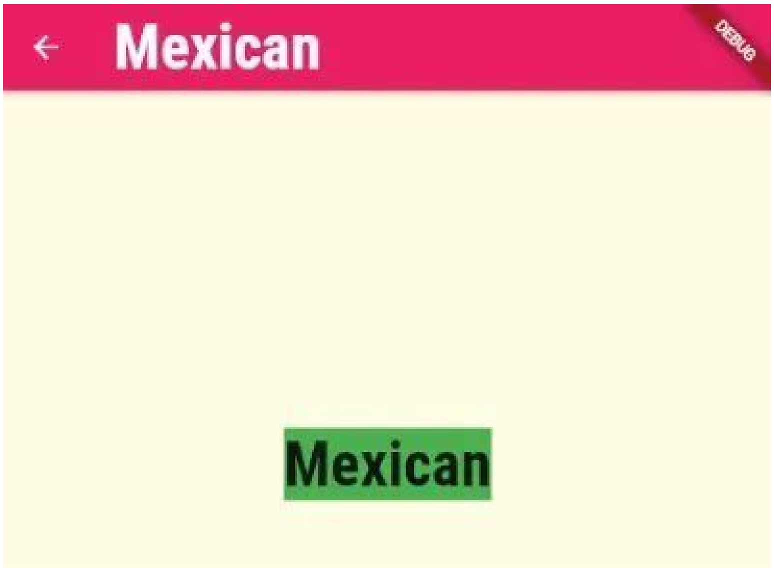


Figure 6.5 – Flutter navigation send data to a page

Simple Flutter navigation

Let us see each case separately, so we can compare them.

First, we are using the #route property where we try to pass data through class constructor.

```
import 'package:coo_king_kong/view/individual_category_pa\
ge.dart';
import 'package:flutter/material.dart';

import 'coo_king_kong_home.dart';

class CooKingKongApp extends StatelessWidget {
const CooKingKongApp({Key? key}) : super(key: key);

// This widget is the root of your application.

/// If only [routes] is given, it must include an entry f\
or the
///[Navigator.defaultRouteName] (/), since that is the ro\
ute used
///when the application is launched with an intent that s\
pecifies
///an otherwise unsupported route.

@override
Widget build(BuildContext context) {
    return MaterialApp(
    title: 'Flutter Demo',
    theme: useCustomTheme(),
    home: const CooKingKongHome(),

    /// this will throw error
    routes: {
        '/individual-category-page': ((context) =>
            IndividualCategoryPage(id: id, title: title, \
color: color))
    },
    );
}
}
// code is incomplete for brevity, to study the full code\
, please clone the entire GitHub repository
```

The #routes property which is a Map, uses the key as a String data. Just like a web URL.

Although there is no naming convention, yet the name should be meaningful. In our case, we want to navigate to "IndividualCategoryPage" Widget. Therefore we have used that name as our key.

However, the value is a #context that returns the "IndividualCategoryPage()" Widget which has many named parameters. In this page it is impossible to supply them.

Why? Because we have define the named parameters in that Widget.

```
import 'package:flutter/material.dart';

class IndividualCategoryPage extends StatelessWidget {
const IndividualCategoryPage({
    Key? key,
    required this.id,
    required this.title,
    required this.color,
}) : super(key: key);

final String id;
final String title;
final Color color;

@override
Widget build(BuildContext context) {
    return Scaffold(
    appBar: AppBar(
        title: Text(title),
    ),
    body: Center(
        child: Container(
        color: color,
        child: Text(
            title,
            style: Theme.of(context).textTheme.titleLarge,
        ),
        ),
    ),
    );
}
}
```

As a result, we must discard this route mechanism. And we must adopt the named route instead.

How to use Flutter named route

How can we do that?

We can change the #routes property as follows.

```
routes: {
        '/individual-category-page': (context) =>
            const IndividualCategoryPage(),
    },
```

But to do that, we can not use Navigator.push() method. In place of that, we should use the Navigator.of(context).pushNamed() method.

The Navigator.of(context).pushNamed() method has #arguments as its named parameter. Which is again a Map where we can associate the key and value as follows.

```
import 'package:flutter/material.dart';

import 'individual_category_page.dart';

/// TODO: this page displays each category by name and co\
lor
/// we will add a method that will take us to individual
/// category page

class CategoryItem extends StatelessWidget {
const CategoryItem({
    Key? key,
    required this.id,
    required this.title,
    required this.color,
}) : super(key: key);

final String id;
final String title;
final Color color;

void selectCategory(BuildContext context) {
    Navigator.of(context).pushNamed(
    '/individual-category-page',
    arguments: {
        'id': id,
        'title': title,
        'color': color,
    },
    );
}

@override
Widget build(BuildContext context) {
    return InkWell(
    onTap: () => selectCategory(context),
    splashColor: Theme.of(context).primaryColor,
    borderRadius: BorderRadius.circular(15),
    child: Container(
        //margin: const EdgeInsets.all(8.0),
        child: Center(
        child: Text(
            title,
            style: Theme.of(context).textTheme.titleSmall,
            textAlign: TextAlign.center,
        ),
        ),
        decoration: BoxDecoration(
        gradient: LinearGradient(
            colors: [
            color.withOpacity(0.6),
            color,
            ],
            begin: Alignment.bottomLeft,
            end: Alignment.topRight,
        ),
        borderRadius: BorderRadius.circular(15.0),
        ),
    ),
    );
}
}
```

As a result, we can receive the values in the destination page with the help of the ModalRoute class.

As a result, the entire code of the destination Widget, “Individual-CategoryPage” will change as follows.

```
import 'package:flutter/material.dart';

class IndividualCategoryPage extends StatelessWidget {
const IndividualCategoryPage({
    Key? key,
}) : super(key: key);

static const routeName = '/individual-category-page';

@override
Widget build(BuildContext context) {
    final routeArguments =
        ModalRoute.of(context)!.settings.arguments as Map\
<String, Object>;
    //final id = routeArguments['id'];
    final Object? title = routeArguments['title'];
    final Color? color = routeArguments['color'] as Color;

    return Scaffold(
    appBar: AppBar(
        title: Text(title.toString()),
    ),
    body: Center(
        child: Container(
        color: color,
        child: Text(
            title.toString(),
            style: Theme.of(context).textTheme.titleLarge,
        ),
        ),
    ),
    );
}
}
```

To avoid mistake in the #routes key value, we have defined the name in the destination page as a static constant String.

After that, we have changed the value of the #routes property in the MaterialApp.

```
routes: {
        IndividualCategoryPage.routeName: (context) =>
            const IndividualCategoryPage(),
    },
```

What is the advantage?

The biggest advantage is when we have a plenty of pages to navigate, the chance of mistake is low.

In fact, there is no chance that we do a spelling mistake and that crashes the entire app.

To sum up, Flutter named route has many features that we should take care of.

It makes our app dynamic. In addition, we can send any type of data.

Object relationship in Flutter

What is relation? When we use the word, we mean connection between two or many persons. Now, it is true for relation in Flutter also.

Why?

Because, in Flutter we use many objects. Right? For that reason, these objects may have relations.

Now, relation may be different.

Firstly, It can be one to one object. Secondly, it can be one to many objects. And, finally, it can be many to many objects.

In our App, we have only categories. And we have seen that there are many categories that we can display through GridView.

As a result we see all categories on the home page.

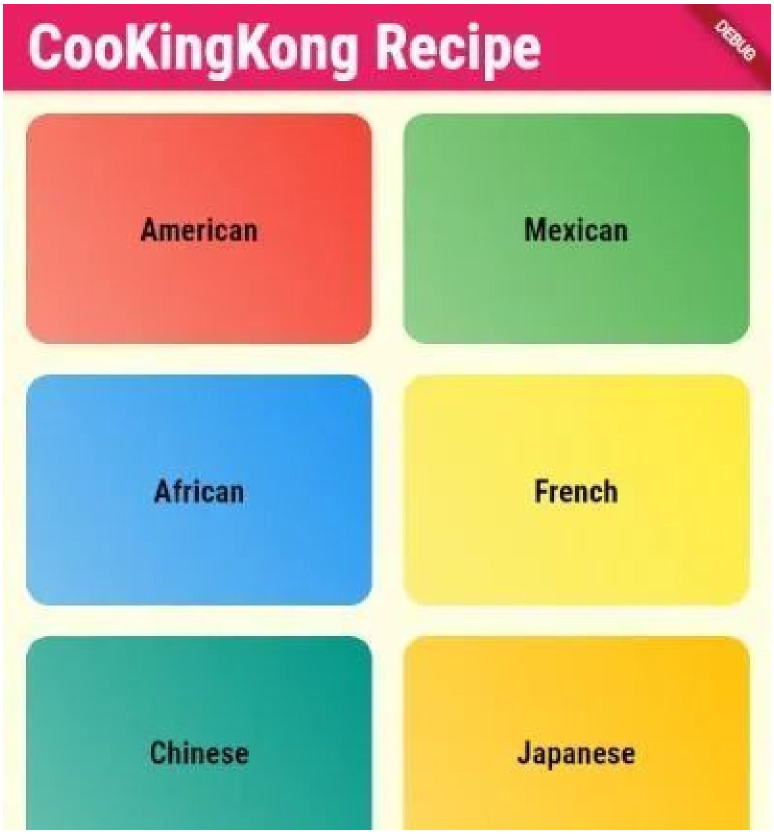


Figure 6.6 – Flutter navigation send data_ categories page

When we click any category, we see that category page. If we click Mexican, we will see the Mexican category.

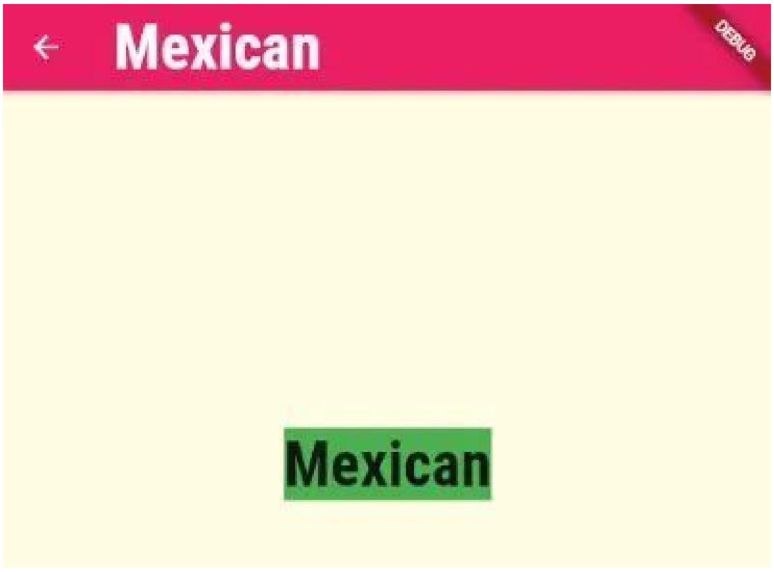


Figure 6.7 – Flutter navigation send data to a page

As we can see, the category comes with the title and color. But there should be Food items which belong to that category. Isn't it?

Because, Mexican category may have different food items. Right?

Therefore each category should display those food items on the corresponding page.

Here comes the challenge to establish the relation in Flutter. Each category has many food items. In other words, each food may have many categories.

As a result, they have a relation. So our next challenge is to establish this relation in Flutter. So that finally we can click any food item to see its funny recipe.

Why we need relation in Flutter

As we have said earlier, our category page should display various food items on the page. To make that happen, we must have a food class and dummy food items.

In addition, those food items may have different properties. We will see that in a minute.

Firstly, we try to understand one key concept in relation flutter.

What is that?

In Food class, we can have a list of categories as its property. And, in that list we can place different categories.

Therefore the vary first food may have three categories. That means the first food may belong to African, Mexican and Indian category.

There is a connection, and there is also a commonness. That is why we need to establish the relation in Flutter.

Secondly, we will look at the food class.

```
enum Complexity {
simple,
complex,
}

class LorenIpsumFood {
final String id;
final List<String> categoryID;
final String title;
final String imageUrl;
final List<String> ingredients;
final List<String> steps;
final int duration;
final Complexity complexity;
final bool isVegan;
final bool isVegetarian;

const LorenIpsumFood({
    required this.id,
    required this.categoryID,
    required this.title,
    required this.imageUrl,
    required this.ingredients,
    required this.steps,
    required this.duration,
    required this.complexity,
    required this.isVegan,
    required this.isVegetarian,
});
}
```

In the above code, we have many instance variables. Two of them of the enum type. We have discussed enum earlier. If you are a beginner, please read that section.

However, we have established relation in Flutter with one property in Food class.

```
final List<String> categoryID;
```

Now we will create a few items of food. To do that, we create many instances by passing values through class constructor.

```
import 'food.dart';

const dummyLorenIpsumLorenIpsumFood = [
LorenIpsumFood(
    id: 'f1',
    categoryID: ['c1', 'c2', 'c8'],
    title: 'Lorem ipsum dolor sit amet',
    complexity: Complexity.simple,
    imageUrl:
        'https://upload.wikimedia.org/wikipedia/commons/t\
humb/2/20/Spaghetti_Bolognese_mit_Parmesan_oder_Grana_Pad\
ano.jpg/800px-Spaghetti_Bolognese_mit_Parmesan_oder_Grana\
_Padano.jpg',
    duration: 20,
    ingredients: [
    '1 Lorem ipsum dolor sit amet, consectetur adipiscing\
 elit',
    '1 Lorem ipsum dolor sit amet',
    '1 Lorem ipsum dol',
    '250g Lorem ipsum dolor',
    'Lorem ipsum',
    'Lorem ipsum dolor'
    ],
    steps: [
    'Lorem ipsum dolor sit amet, consectetur adipiscing e\
lit.',
    'Lorem ipsum dolor sit amet, consectetur.',
    'Lorem ipsum dolo, consectetur adipiscing elit.',
    'Lorem ipsum dolor sit amet, consectetur.',
    'Lorem ipsum dolor sit amet, adipiscing elit.',
    'Lorem ipsum dolor sit amet, consectetur adipiscing e\
lit.',
    'Lorem ipsum dolor sit amet.'
    ],
    isVegan: true,
    isVegetarian: true,
),
LorenIpsumFood(
    id: 'f2',
    categoryID: ['c2', 'c5', 'c4'],
    title: 'Lorem ipsum',
    complexity: Complexity.simple,
...
// code is incomplete for brevity
```

As we see, the first food object has three category properties. In the similar vein, the second food item also belongs to three categories.

Now, all food titles can come on the individual category page.

So we need to change the code of the “IndividualCategoryPage” Widget.

How to establish relation in Flutter

Here lies our main challenge. The dummy food list contains a list of category id. Therefore, we can map that list and return each category id.

The Dart List type has a “where” method that can map through the items and return the list of items.

As a result, we can take an advantage of that List method.

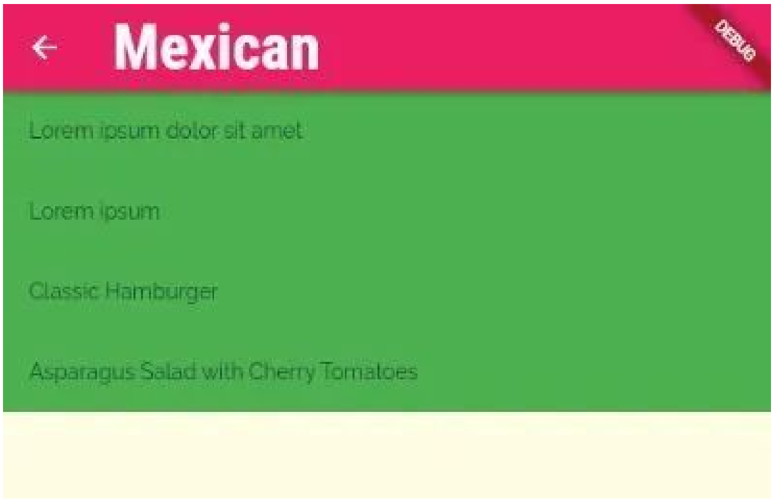
Therefore, we should change the code of the “IndividualCategory-Page” Widget.

```
import 'package:flutter/material.dart';

import '../model/dummy_foods.dart';

class IndividualCategoryPage extends StatelessWidget {
const IndividualCategoryPage({
    Key? key,
}) : super(key: key);

static const routeName = '/individual-category-page';

@override
Widget build(BuildContext context) {
    final routeArguments =
        ModalRoute.of(context)!.settings.arguments as Map\
<String, Object>;
    final id = routeArguments['id'];
    final Object? title = routeArguments['title'];
    final Color? color = routeArguments['color'] as Color;
    final categoryMeals = dummyLorenIpsumLorenIpsumFood.w\
here((food) {
    return food.categoryID.contains(id);
    }).toList();

    return Scaffold(
    appBar: AppBar(
        title: Text(title.toString()),
    ),
    body: ListView.builder(
        itemBuilder: (ctx, index) {
        return Container(
            padding: const EdgeInsets.all(18.0),
            color: color,
            child: Text(
            categoryMeals[index].title,
            style: Theme.of(context).textTheme.bodyMedium,
            ),
        );
        },
        itemCount: categoryMeals.length,
    ),
    );
}
}
```

Now we can tap any category and get all the titles of the food items.

These food items belong to that category.

First, we tap the Mexican category to see what food items are there.



Mexican

Lorem ipsum dolor sit amet

Lorem ipsum

Classic Hamburger

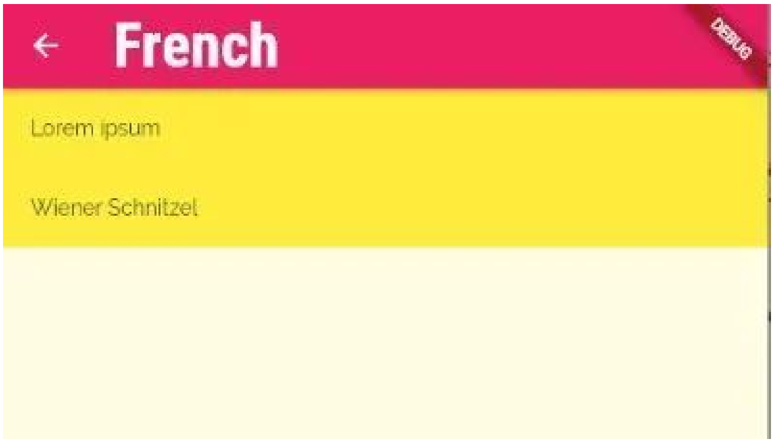
Asparagus Salad with Cherry Tomatoes

Figure 6.8 – Relation in Flutter first example

Four food titles are there.

Next, we tap the French category. It has two food items. Because when we have created Food objects, we pass this category id in two cases.

As a result, the category French has two food titles.



Lorem ipsum

Wiener Schnitzel

Figure 6.9 – Relation in Flutter second example

However, our CooKingKong App is not finished yet. We have just scratched the surface.

Now we understand that we can display the food items with title and image.

At the same time, we can also use Flutter named route to navigate to the individual Food page where we can show our funny recipe.

To do that, we need to change the design and the business logic.

In the next section we will discuss that.

So stay tuned.

Model View Controller

Model view controller architecture is a software design pattern that we can apply in Flutter. Why? Because we want clean code that will help us.

Help us to what? It will help us to understand the code. Why we are writing this part or that part. In addition, how we are connecting every part.

In this section we will learn how we can follow model view controller architecture. Moreover, how we can apply this principle to our CooKingKong food app.

By the way, model view controller architecture is also known as MVC. Therefore, further in our discussion we will use the term MVC.

What does MVC mean?

In our Flutter project we create three folders under the root folder "lib". We name them controller, model, and view. If you want to change the name view to screen, that's fine.

Or you can change the name of controller to widgets. But, that does not affect the design pattern.

The question is, how it works?

The principle says, we keep our data source at the model folder. As a result, in our CooKingKong App, we have put all category, and food classes, dummy data in Model folder.

Therefore, model supplies the data. And the view part will display that data. But, the controller plays the role of a mediator between them.

The view asks the controller, "Hey, I need this. Can you supply this?"

The controller asks model, "Do you have this data?"

If the model says yes, then controller supplies that data to the view. What is the end result? The view does not know where the data comes from. Only controller knows that.

If model says, I don't have this data, controller conveys that message to the view.

As a result, user sees a relevant message.

Enough talking, let us jump in to our code. We have progressed a little bit. But we need to take a look at how we have progressed so far.

Firstly, we have seen how to display all categories using the GridView.

So, the CooKingKong App looks like the following.

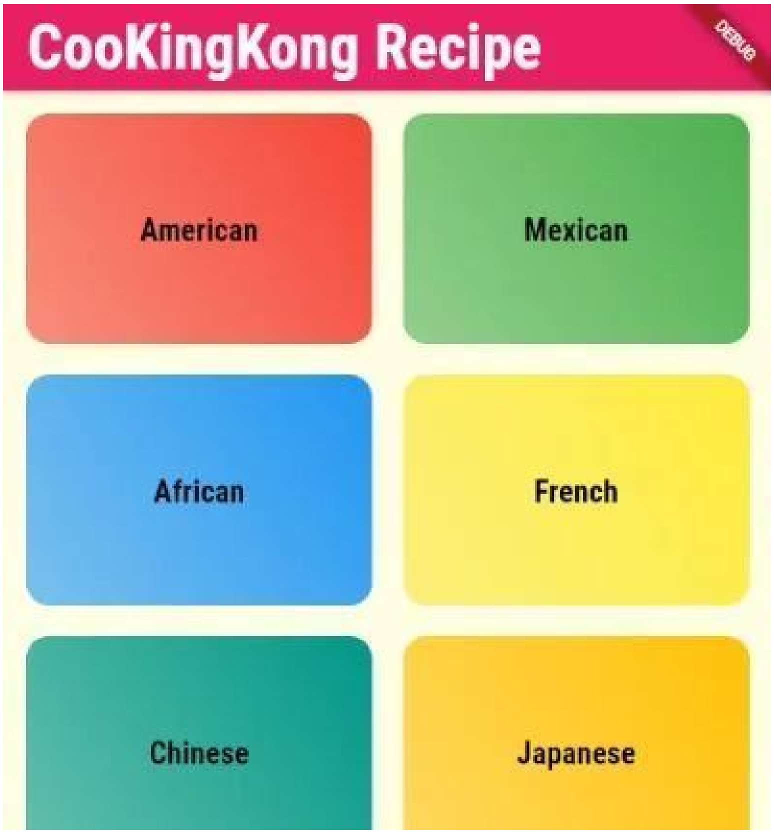


Figure 6.10 – Flutter navigation send data_ categories page

Next, we have seen how we can send data from one page to the other page.

Consequently, we tap any category and see the relevant page.

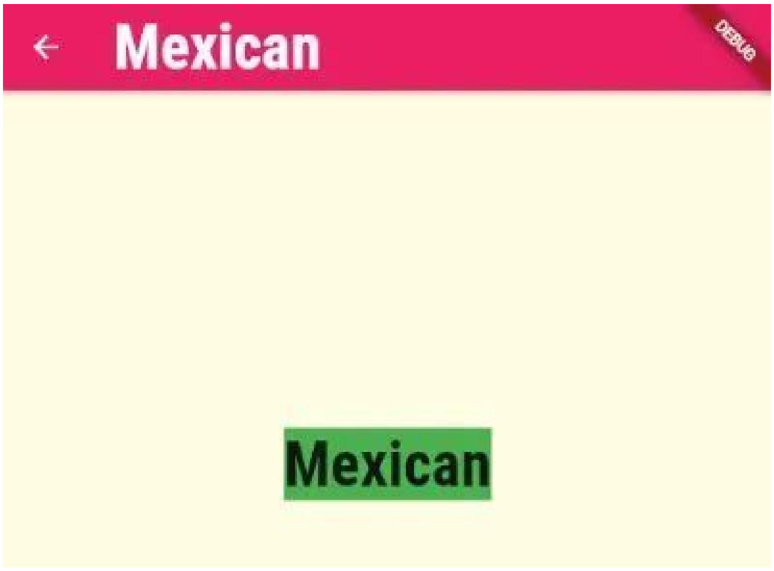


Figure 6.11 – Flutter navigation send data to a page

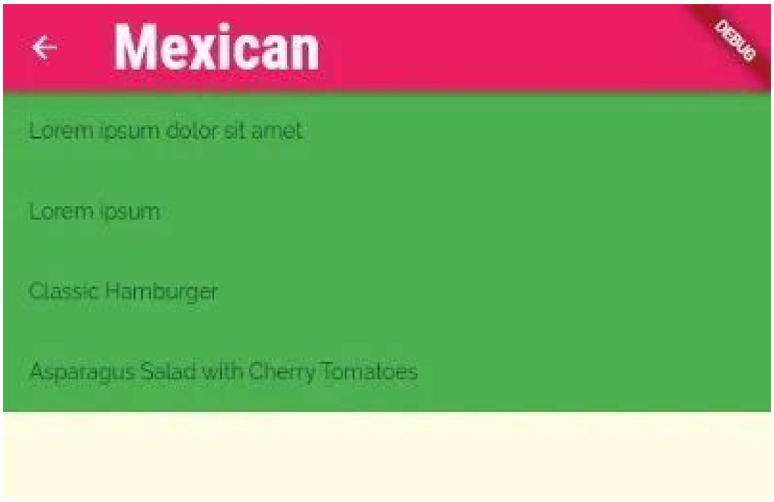
After that, we have also learned the named route to send data in a better way.

Why we need the named route to send data? The main reason is, as our app size grows, number of pages also grow. In addition, we need to send data from one page to another page.

As a result, the relation between these pages becomes complex.

How do we handle this relation?

We have used ModalRoute.of() method to get all properties. And, after that, we display some of the properties on individual food page.



Lorem ipsum dolor sit amet

Lorem ipsum

Classic Hamburger

Asparagus Salad with Cherry Tomatoes

Figure 6.12 – Relation in Flutter first example

We need to understand that each individual category should have many food items. Right?

Therefore, the individual category page code looks as follows.

```
import 'package:flutter/material.dart';

import '../model/dummy_foods.dart';

class IndividualCategoryPage extends StatelessWidget {
const IndividualCategoryPage({
    Key? key,
}) : super(key: key);

static const routeName = '/individual-category-page';

@override
Widget build(BuildContext context) {
    final routeArguments =
        ModalRoute.of(context)!.settings.arguments as Map\
<String, Object>;
    final id = routeArguments['id'];
    final Object? title = routeArguments['title'];
    final Color? color = routeArguments['color'] as Color;
    final categoryMeals = dummyLorenIpsumLorenIpsumFood.w\
here((food) {
    return food.categoryID.contains(id);
    }).toList();

    return Scaffold(
    appBar: AppBar(
        title: Text(title.toString()),
    ),
    body: ListView.builder(
        itemBuilder: (ctx, index) {
        return Container(
            padding: const EdgeInsets.all(18.0),
            color: color,
            child: Text(
            categoryMeals[index].title,
            style: Theme.of(context).textTheme.bodyMedium,
            ),
        );
        },
        itemCount: categoryMeals.length,
    ),
    );
}
}
```

But that was the fourth step where we have just displayed the titles of the food items. If you want to understand the fourth step to understand the flow of logic, please clone this branch of GitHub repository.

But it was not our intention to show only titles.

Instead we want to display each food item with respective image, and other properties.

That's why we separate the logic in separate folders applying the MVC architecture.

Now, in the model folder we have category and food classes. Further we keep dummy data there.

After that, in controller folder we keep category and food item logic that will take data from model and supply them to the view folder pages.

Role of controller in MVC

As we have said earlier the controller bridges between model and view. In other words, controller relays the message between these two components.

So we keep category item and food item Widgets in controller folder.

The category item will send the data by Navigator.of(context).pushNamed() method to the individual category page.

```
import 'package:flutter/material.dart';

class CategoryItem extends StatelessWidget {
const CategoryItem({
    Key? key,
    required this.id,
    required this.title,
    required this.color,
}) : super(key: key);

final String id;
final String title;
final Color color;

void selectCategory(BuildContext context) {
    Navigator.of(context).pushNamed(
    '/individual-category-page',
    arguments: {
        'id': id,
        'title': title,
        'color': color,
    },
    );
}

@override
Widget build(BuildContext context) {
    return InkWell(
    onTap: () => selectCategory(context),
    splashColor: Theme.of(context).primaryColor,
    borderRadius: BorderRadius.circular(15),
    child: Container(
        //margin: const EdgeInsets.all(8.0),
        child: Center(
        child: Text(
            title,
            style: Theme.of(context).textTheme.titleSmall,
            textAlign: TextAlign.center,
        ),
        ),
        decoration: BoxDecoration(
        gradient: LinearGradient(
            colors: [
            color.withOpacity(0.6),
            color,
            ],
            begin: Alignment.bottomLeft,
            end: Alignment.topRight,
        ),
        borderRadius: BorderRadius.circular(15.0),
        ),
    ),
    );
}
}
```

It gets the data from the model. Consequently, it sends data to category page.

The category page is in the view folder.

```
import 'package:coo_king_kong/model/dummy_categories.dart\
';
import 'package:coo_king_kong/controller/category_item.da\
rt';
import 'package:flutter/material.dart';

class CategoriesPage extends StatelessWidget {
const CategoriesPage({Key? key}) : super(key: key);

@override
Widget build(BuildContext context) {
    return GridView(
    padding: const EdgeInsets.all(15.0),
    children: dummyCategories
        .map(
            (e) => CategoryItem(id: e.id, title: e.title,\
 color: e.color),
        )
        .toList(),
    gridDelegate: const SliverGridDelegateWithMaxCrossAxi\
sExtent(
        maxCrossAxisExtent: 300,
        childAspectRatio: 1.50,
        mainAxisSpacing: 20.0,
        crossAxisSpacing: 20.0,
    ),
    );
}
}
```

For that reason, we see all categories on the home page.

But, as we tap any category, that takes us to the individual category page where we see all the food items belonging to that category.

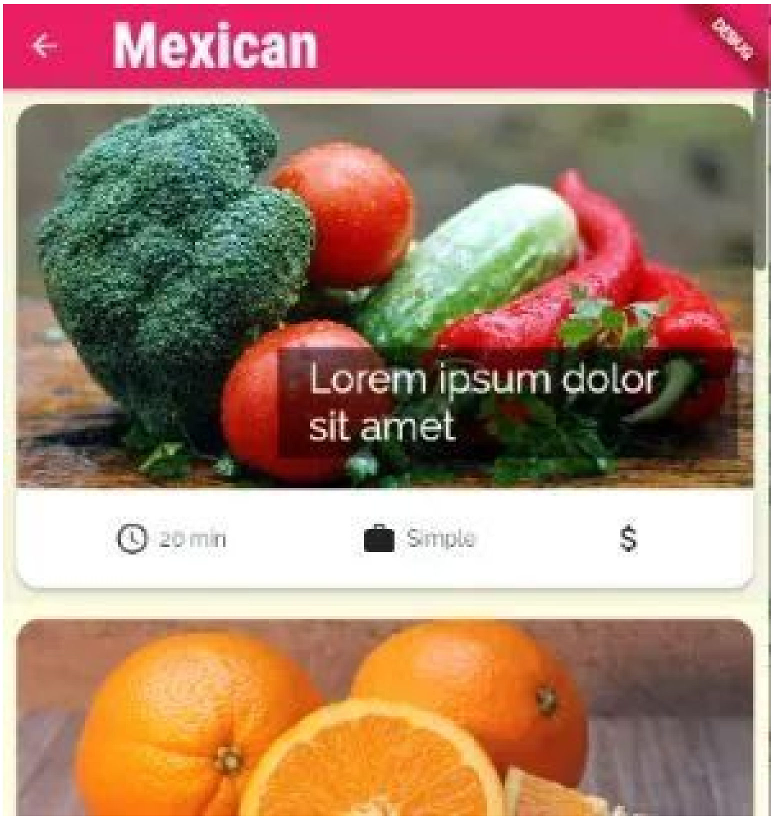


Figure 6.13 – Model view controller architecture first Example

As we see, the Mexican category has many food items in its page.

To scroll and see all the food items we have used ListView.builder() method. Not only that, in this page another controller, food item controller sends every food item.

```
import 'package:flutter/material.dart';

import '../controller/food_item.dart';
import '../model/dummy_foods.dart';

class IndividualCategoryPage extends StatelessWidget {
const IndividualCategoryPage({
    Key? key,
}) : super(key: key);

static const routeName = '/individual-category-page';

@override
Widget build(BuildContext context) {
    final routeArguments =
        ModalRoute.of(context)!.settings.arguments as Map\
<String, Object>;
    final id = routeArguments['id'];
    final Object? title = routeArguments['title'];
    final categoryMeals = dummyLorenIpsumLorenIpsumFood.w\
here((food) {
    return food.categoryID.contains(id);
    }).toList();

    return Scaffold(
    appBar: AppBar(
        title: Text(title.toString()),
    ),
    body: ListView.builder(
        itemBuilder: (ctx, index) {
        return FoodItem(
            id: categoryMeals[index].id,
            title: categoryMeals[index].title,
            imageUrl: categoryMeals[index].imageUrl,
            duration: categoryMeals[index].duration,
            complexity: categoryMeals[index].complexity,
        );
        },
        itemCount: categoryMeals.length,
    ),
    );
}
}
```

Again the food item controller acts as a supplier. Moreover, it paves the way to go to the individual food page where we will see the food item in detail.

```
import 'package:coo_king_kong/view/individual_food_page.d\
art';
import 'package:flutter/material.dart';

import '../model/food.dart';

class FoodItem extends StatelessWidget {
final String id;
final String title;
final String imageUrl;
final int duration;
final Complexity complexity;

const FoodItem({
    Key? key,
    required this.id,
    required this.title,
    required this.imageUrl,
    required this.complexity,
    required this.duration,
}) : super(key: key);

```

```
String get complexityText {
    switch (complexity) {
    case Complexity.simple:
        return 'Simple';
    case Complexity.complex:
        return 'Complex';
    default:
        return 'Unknown';
    }
}

void selectMeal(BuildContext context) {
    Navigator.of(context).pushNamed(
    IndiividualFoodPage.routeName,
    arguments: title,
    );
}

@override
Widget build(BuildContext context) {
    return InkWell(
    onTap: () => selectMeal(context),
    child: Card(
        shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(15),
        ),
        elevation: 4,
        margin: const EdgeInsets.all(10),
        child: Column(
        children: <Widget>[
            Stack(
            children: <Widget>[
                ClipRRect(
                borderRadius: const BorderRadius.only(
                    topLeft: Radius.circular(15),
                    topRight: Radius.circular(15),
                ),
                child: Image.network(
                    imageUrl,
                    height: 250,
                    width: double.infinity,
                    fit: BoxFit.cover,
                ),
                ),
                Positioned(
                bottom: 20,
                right: 10,
                child: Container(
                    width: 300,
                    color: Colors.black54,
                    padding: const EdgeInsets.symmetric(
                    vertical: 5,
                    horizontal: 20,
                    ),
                    child: Text(
                    title,
                    style: const TextStyle(
                        fontSize: 26,
                        color: Colors.white,
                    ),
                    softWrap: true,
                    overflow: TextOverflow.fade,
                    ),
                ),
                )
            ],
            ),
            Padding(
            padding: const EdgeInsets.all(20),
            child: Row(
                mainAxisAlignment: MainAxisAlignment.spac\
eAround,
                children: <Widget>[
                Row(
                    children: <Widget>[
                    const Icon(
                        Icons.schedule,
                    ),
                    const SizedBox(
                        width: 6,
                    ),
                    Text('$duration min'),
                    ],
                ),
                Row(
                    children: <Widget>[
                    const Icon(
                        Icons.work,
                    ),
                    const SizedBox(
                        width: 6,
                    ),
                    Text(complexityText),
                    ],
                ),
                Row(
                    children: const <Widget>[
                    Icon(
                        Icons.attach_money,
                    ),
                    SizedBox(
                        width: 6,
                    ),
                    ],
                ),
                        ],
                    ),
                    ),
                ],
                ),
        ),
        );
}
}
```

Watch the this piece of code.

The following code gives us the hint.

```
void selectMeal(BuildContext context) {
    Navigator.of(context).pushNamed(
    IndiividualFoodPage.routeName,
    arguments: title,
    );
}
```

Now, as a result, we can show the title of the food on the individual food page.

Later we will design the page so that it displays all food properties in detail.

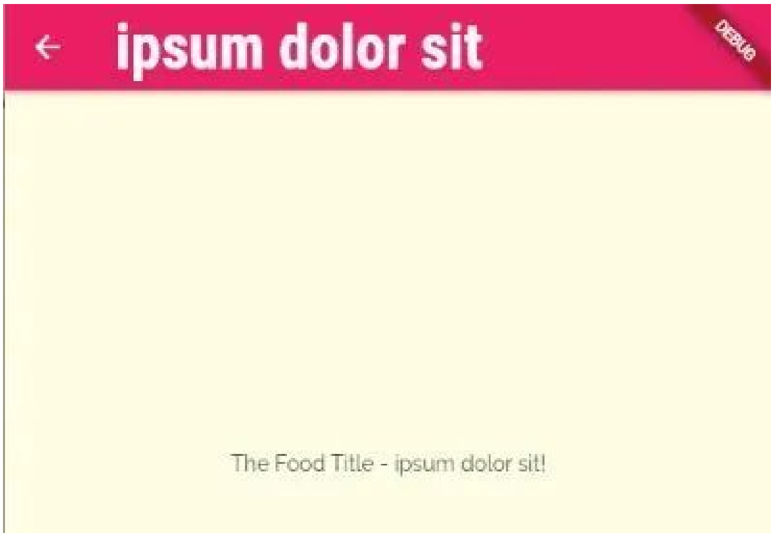


Figure 6.14 – Model view controller architecture second Example

Now we can take a look at the individual food page. At present we have a small code snippet.

```
import 'package:flutter/material.dart';

class IndiividualFoodPage extends StatelessWidget {
const IndiividualFoodPage({Key? key}) : super(key: key);
static const routeName = '/food-detail';

@override
Widget build(BuildContext context) {
    final foodTitle = ModalRoute.of(context)!.settings.ar\
guments as String;
    return Scaffold(
    appBar: AppBar(
        title: Text(foodTitle),
    ),
    body: Center(
        child: Text('The Food Title - $foodTitle!'),
    ),
    );
}
}
```

For example, we have displayed only the food title on a page that resides on the view folder. But the data comes from the model folder. In addition, the controller supplies that data to the view folder.

The Next Challenge

Our next challenge will be to display the full food items on the individual food page.

To understand the whole MVC architecture you may clone this GitHub repository. After that, you can run the code in your local machine, and see how it has worked.

(Clone this step)[https://github.com/sanjibsinha/coo_king_-kong/tree/fifth-step-mvc]

Relation between tables, list and map

As the relation between the pages becomes complex, we need to be careful. Why?Because we have been handling List and Map. When we iterate Flutter List, we must assure that we send correct data.

Otherwise, our app might crash.

So far, we have learned a few key concepts while building a recipe app.

We have seen how we can send data from one page to the other page.

In addition, we have also learned the named route to send data in a better way.

Let us take a look at the previous steps.

Firstly, we have built the categories home page. Here we can see every category of food recipe.

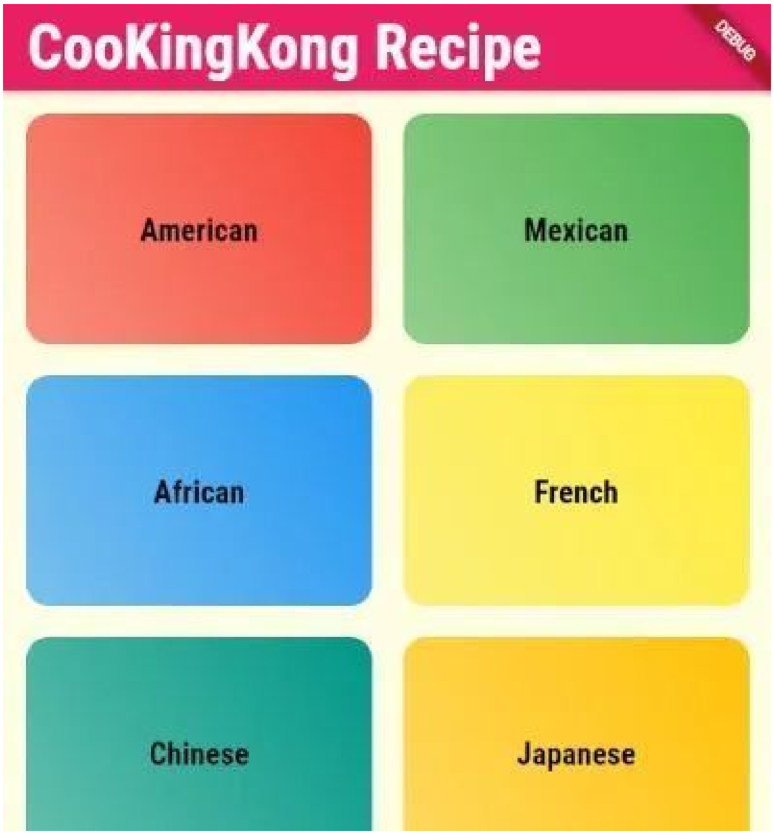


Figure 6.15 – Flutter navigation send data_ categories page

Next, we see how every category houses different type of food items.

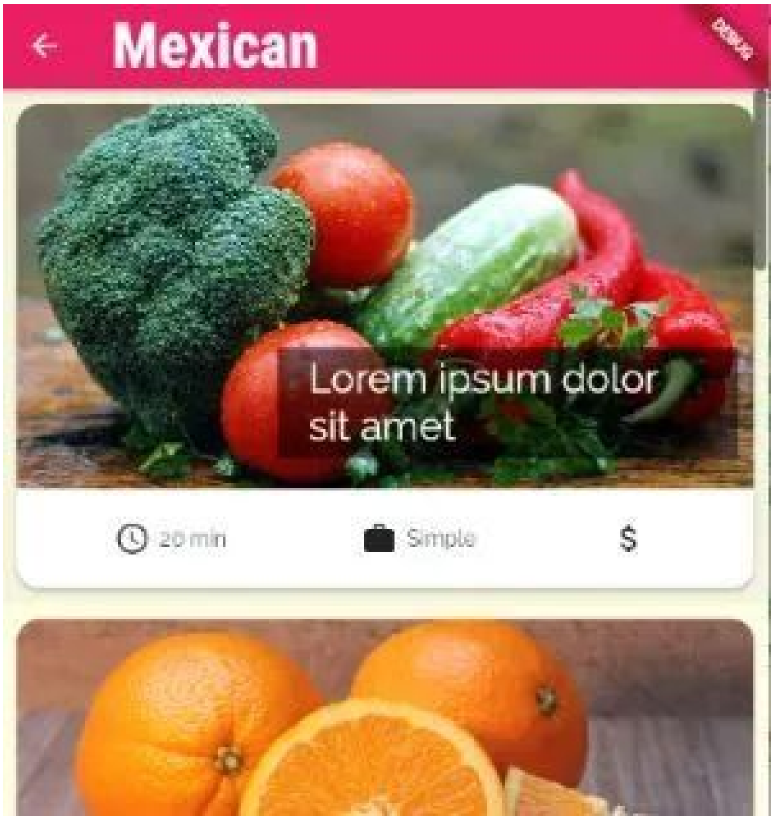
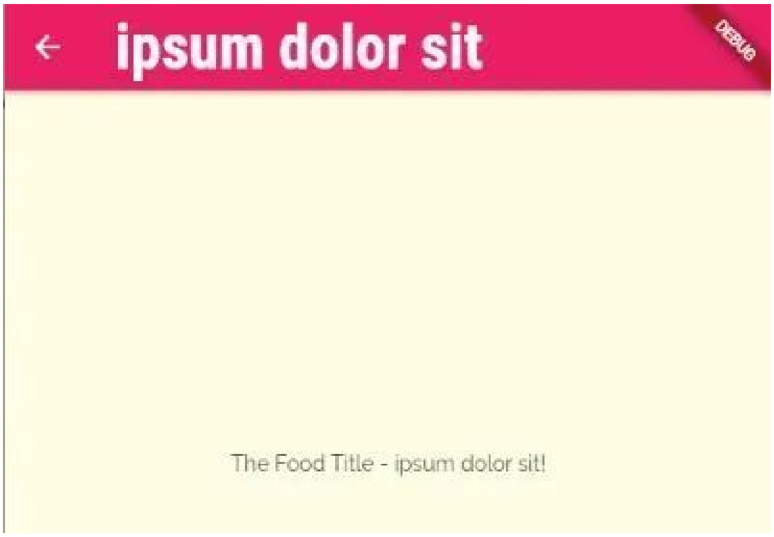


Figure 6.16 – Model view controller architecture first Example

After that, we can click any food item and see the associated title.



The Food Title - ipsum dolor sit!

Figure 6.17 – Model view controller architecture second Example

But, we want to see the full content of a food item. Why? Because, we know that dummy food class has many properties.

The question is, how we can get every content of the food item?

It is only possible if we pass the unique food id through the controller food item.

```
void selectMeal(BuildContext context) {
    Navigator.of(context).pushNamed(
    IndiividualFoodPage.routeName,
    arguments: id,
    );
}

// code is incomplete for brevity
// please clone this branch of GitHub repository
```

In our previous section, we have sent the title. As a result, whenever we had tapped any food item, we saw the title.

However, that is not our intention. We want to see all the recipe on the individual food page. Right?

Therefore, this time we have replaced the title argument to id.

As a result, we can now use a Dart list method "firstWhere".

What does this function do?

This function takes one parameter element and searches that element in the list.

In our case, we have searched the same way.

final foodId = ModalRoute.of(context)!.settings.arguments as String; final selectedMeal = dummyLorenIpsumLorenIpsumFood.firstWhere((food) => food.id == foodId);

// code is incomplete for brevity // please clone this branch of GitHub repository

Why Flutter List Iterate is important?

We will answer the above question. But, before that, we want to answer the following query.

If we take a look at any list in Dart or Flutter, what do we see?

We see a list of items. Right?

Our dummy food list is also like that.

```
import 'food.dart';

const dummyLorenIpsumLorenIpsumFood = [
LorenIpsumFood(
    id: 'f1',
    categoryID: ['c1', 'c2', 'c8'],
    title: 'Lorem ipsum dolor sit amet',
    complexity: Complexity.simple,
    imageUrl:
        'https://cdn.pixabay.com/photo/2016/08/11/08/04/v\
egetables-1584999_960_720.jpg',
    duration: 20,
    ingredients: [
    '1 DolorSan sit amet, consectetur adipiscing elit',
    '1 Lorem ipsum dolor sit amet',
    '1 Lorem ipsum dol',
    '250g Lorem ipsum dolor',
    'Lorem ipsum',
    'Lorem ipsum dolor'
    ],
    steps: [
    'Lorem ipsum dolor sit amet, consectetur adipiscing e\
lit.',
    'Lorem ipsum dolor sit amet, consectetur.',
    'Lorem ipsum dolo, consectetur adipiscing elit.',
    'Lorem ipsum dolor sit amet, consectetur.',
    'Lorem ipsum dolor sit amet, adipiscing elit.',
    'Lorem ipsum dolor sit amet, consectetur adipiscing e\
lit.',
    'Lorem ipsum dolor sit amet.'
    ],
    isVegan: true,
    isVegetarian: true,
),
LorenIpsumFood(
    id: 'f2',
    categoryID: ['c2', 'c5', 'c4'],
    title: 'Lorem ipsum',
    complexity: Complexity.simple,
...
// code is incomplete for brevity
// please clone this branch of GitHub repository
```

The above data is coming from the model folder.

We have created objects. And each food object has unique id. Now, based on that id we can search the list.

But we need to be careful that as the argument we should pass the id. Not any other property.

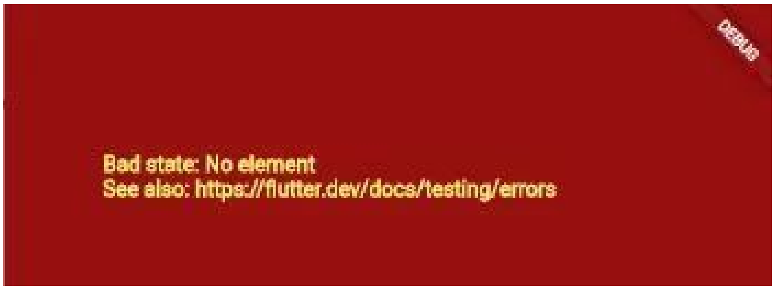
Suppose we have mistakenly sent the title instead of id.

What will happen?

```
void selectMeal(BuildContext context) {
    Navigator.of(context).pushNamed(
    IndiividualFoodPage.routeName,
    arguments: title,
    );
}
...
final foodId = ModalRoute.of(context)!.settings.arguments\
 as String;
    final selectedMeal =
        dummyLorenIpsumLorenIpsumFood.firstWhere((food) =\
> food.id == foodId);
...
```

As we see these properties don't match at all.

As a result, this will crash the app and throw an error.



```
Bad state: No element
See also: https://flutter.dev/docs/testing/errors
```

Figure 6.18 – Flutter List iterate example one

However, if send the id, and check that against the list items, it will work.

Now we can click any food item on the category page. And it takes us to the individual food page.

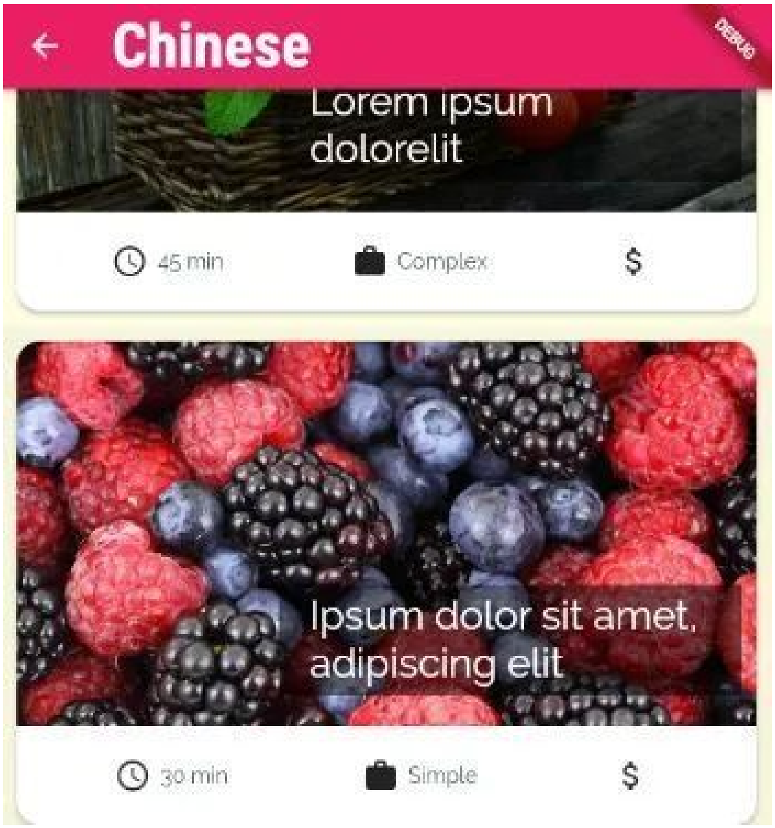


Figure 6.19 – Flutter List iterate example two

Everybody loves Chinese food. So we scroll down and tap the last item.

It will display the detail.

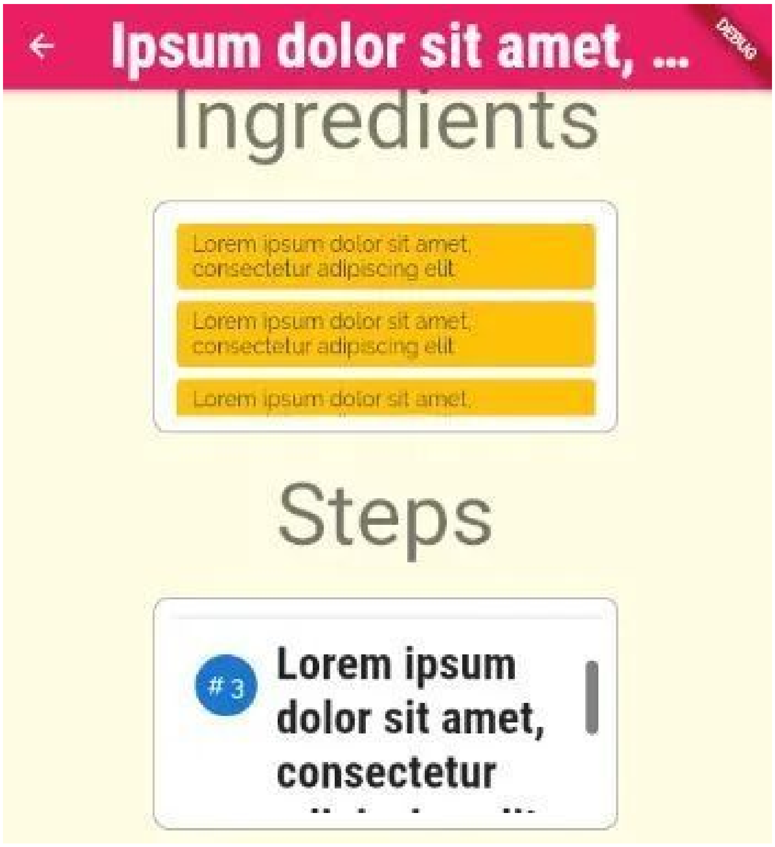

Ipsum dolor sit amet, ...
DEBUG
Ingredients
Lorem ipsum dolor sit amet, consectetur adipiscing elit
Lorem ipsum dolor sit amet, consectetur adipiscing elit
Lorem ipsum dolor sit amet,
Steps
# 3
Lorem ipsum dolor sit amet, consectetur

Figure 6.20 – Flutter List iterate example three

Now the individual food page has many features. We can scroll the page to see every property.

Not only that we can also scroll the individual properties like “steps” as we see in the above image.

Let us see the code of individual food page, so that we will realize how it works.

```dart
import 'package:flutter/material.dart';

import '../model/dummy_foods.dart';

///Displaying the individual food page
///
class IndiividualFoodPage extends StatelessWidget {
const IndiividualFoodPage({Key? key}) : super(key: key);
static const routeName = '/food-detail';

Widget displayTitle(BuildContext context, String text) {
    return Container(
    margin: const EdgeInsets.symmetric(vertical: 10),
    child: Text(
        text,
        style: Theme.of(context).textTheme.headline2,
    ),
    );
}

Widget displayContent(Widget child) {
    return Container(
    decoration: BoxDecoration(
        color: Colors.white,
        border: Border.all(color: Colors.grey),
        borderRadius: BorderRadius.circular(10),
    ),
    margin: const EdgeInsets.all(10),
    padding: const EdgeInsets.all(10),
    height: 150,
    width: 300,
    child: child,
    );
}

```

```
@override
Widget build(BuildContext context) {
    final foodId = ModalRoute.of(context)!.settings.argum\
ents as String;
    final selectedMeal =
        dummyLorenIpsumLorenIpsumFood.firstWhere((food) =\
> food.id == foodId);
    return Scaffold(
    appBar: AppBar(
        title: Text(selectedMeal.title),
    ),
    body: SingleChildScrollView(
        child: Column(
        children: <Widget>[
            Container(
            padding: const EdgeInsets.all(2.0),
            height: 300,
            width: double.infinity,
            child: Image.network(
                selectedMeal.imageUrl,
                fit: BoxFit.cover,
            ),
            ),
            displayTitle(context, 'Ingredients'),
            displayContent(
            ListView.builder(
                itemBuilder: (ctx, index) => Card(
                color: Theme.of(context).colorScheme.seco\
ndary,
                child: Padding(
                    padding: const EdgeInsets.symmetric(
                        vertical: 5,
                        horizontal: 10,
                    ),
                    child: Text(selectedMeal.ingredients[\
index])),
                    ),
                    itemCount: selectedMeal.ingredients.lengt\
h,
               ),
               ),
               displayTitle(context, 'Steps'),
               displayContent(
               ListView.builder(
                    itemBuilder: (ctx, index) => Column(
                    children: [
                         ListTile(
                         leading: CircleAvatar(
                              child: Text('# ${(index + 1)}'),
                         ),
                         title: Text(
                              selectedMeal.steps[index],
                         ),
                         ),
                         const Divider()
                    ],
                    ),
                    itemCount: selectedMeal.steps.length,
               ),
               ),
          ],
          ),
     ),
     );
}
}
```

Certainly we can move forward and make this app bigger by adding the sellers list. And as a result we can convert this recipe app to a shopping app.

Clone the repository and run locally

In the next sections, we will learn some more core features of Flutter. However, if you want to test this step in your local machine, please clone this GitHub repository.

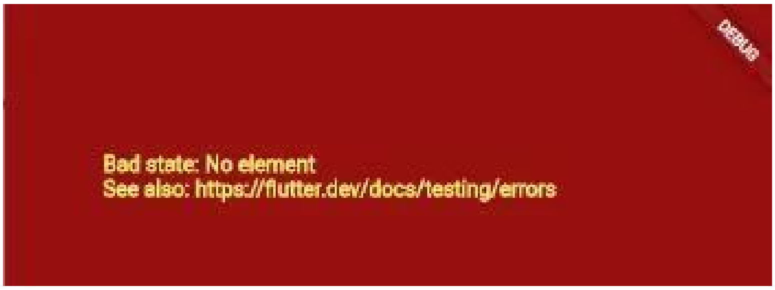
(Clone and run locally)[https://github.com/sanjibsinha/coo_king_-kong/tree/sixth-step-display-food]

What is clean navigation

What do we mean by clean navigation in Flutter? It means, in any case, our app will not crash. When does a Flutter app crash? For many reasons it may happen.

But it happens mostly when we send wrong data to a page.

In our previous section, we have seen such example.



```
Bad state: No element
See also: https://flutter.dev/docs/testing/errors
```

Figure 6.21 – Flutter List iterate example one

Why did it happen? Because we had sent wrong data to a page.

But we could have avoided that. If we had adopted the clean navigation approach, it would not happen.

How can we do that?

We need to modify our route mechanism. To do that we need to work on the page where we had set the navigation rule.

Let us take a look at the “coo_king_kong_app.dart” page code.

```
import 'package:coo_king_kong/view/individual_category_pa\
ge.dart';
import 'package:coo_king_kong/view/individual_food_page.d\
art';
import 'package:coo_king_kong/view/individual_seller_page\
.dart';
import 'package:flutter/material.dart';

import 'coo_king_kong_home.dart';

class CooKingKongApp extends StatelessWidget {
const CooKingKongApp({Key? key}) : super(key: key);

// This widget is the root of your application.

/// If only [routes] is given, it must include an entry f\
or the
///[Navigator.defaultRouteName] (/), since that is the ro\
ute used
///when the application is launched with an intent that s\
pecifies
///an otherwise unsupported route.

@override
Widget build(BuildContext context) {
    return MaterialApp(
    title: 'Flutter Demo',
    theme: useCustomTheme(),
    initialRoute: '/',
    routes: {
        '/': (context) => const CooKingKongHome(),
        IndividualCategoryPage.routeName: (context) =>
            const IndividualCategoryPage(),
        IndiividualFoodPage.routeName: (context) => const\
 IndiividualFoodPage(),
    },
    },
    );
}
// code is incomplete for brevity, please visit GitHub re\
pository for full code
```

As we see, in the above code, there is no “home” property anymore. Instead, we have used the “initialRoute” property of the MaterialApp Widget.

After that, what do we do?

We have used “routes” property which is a Map and through “routes” we have set the navigation rule.

As a result, one thing happens. Our code looks tight. Why? Because there is no flexibility.

Suppose, our app grows bigger and we keep adding many more pages. Consequently, we need to send more data to more pages.

What may happen? We may send wrong data to a page, and our whole Flutter app crash.

But we can avoid that.

How?

We can create a fallback page as we see in web pages. When we don’t get any page, it pops up a 404 page.

It reads, sorry, we don’t find this page.

We can do the same thing in Flutter also.

What is onGenerateRoute and onUnknownRoute? The MaterialApp Widget has two properties. Both the “onGenerateRoute” and “onUnknownRoute” properties are callback.

What does that mean? We have discussed callback before. Still we want recapitulate in one line.

A callback represents a function which we pass as an argument.

The MaterialApp Widget passes these two functions as the callbacks.

Flutter calls the "onUnknownRoute" property when the onGenerateRoute fails to generate a route, except for the initialRoute.

It means a lot for our app. Why? Because, now, if we fail to send the correct data to any page, our app will not crash.

As a result, this acts as a 404 page. In addition, we can specify which page will act as our 404 page.

It assures that our navigation is clean.

Let us see how we can change the build method of the above code to fit these two properties.

```
@override
Widget build(BuildContext context) {
    return MaterialApp(
    title: 'Flutter Demo',
    theme: useCustomTheme(),
    initialRoute: '/',
    routes: {
        '/': (context) => const CooKingKongHome(),
        IndividualCategoryPage.routeName: (context) =>
            const IndividualCategoryPage(),
        IndiividualFoodPage.routeName: (context) => const\
 IndiividualFoodPage(),
    },
    onGenerateRoute: (settings) {
        return MaterialPageRoute(
        builder: (ctx) => const CooKingKongHome(),
        );
    },
    onUnknownRoute: (settings) {
        return MaterialPageRoute(
        builder: (ctx) => const CooKingKongHome(),
        );
    },
    );
}
// code is incomplete for brevity, please visit GitHub re\
pository for full code
```

When our flutter app grows to be bigger, we need to assure that under any circumstance, our app will not crash.

Do you want to clone this step? The process is quite easy.

You can download the zipped folder. Or, you can clone the URL of the GitHub repository.

Hence, we can use these two properties.

(Clone this repository)[https://github.com/sanjibsinha/coo_king_-kong/tree/seventhg-step-different-routes]

Routes property and its functions

We have to be careful when we deal with the navigation in Flutter. In previous section we have discussed it. Yet, in our final step, we would like to chew over route in Flutter again.

Why?

Because, it is one of the important components in Flutter. If we do not send correct data to another page, our app will crash.

Therefore, we have to be careful when we set the "routes" property in MaterialApp Widget.

To clarify, we can use Navigator.push method with Navigator.pushNamed.

However, there is a warning.

In this section we will discuss that.

How to use GestureDetector inside InkWell To understand, we see the images first.

That will explain better than words.

The CooKingKong App opens with all categories.

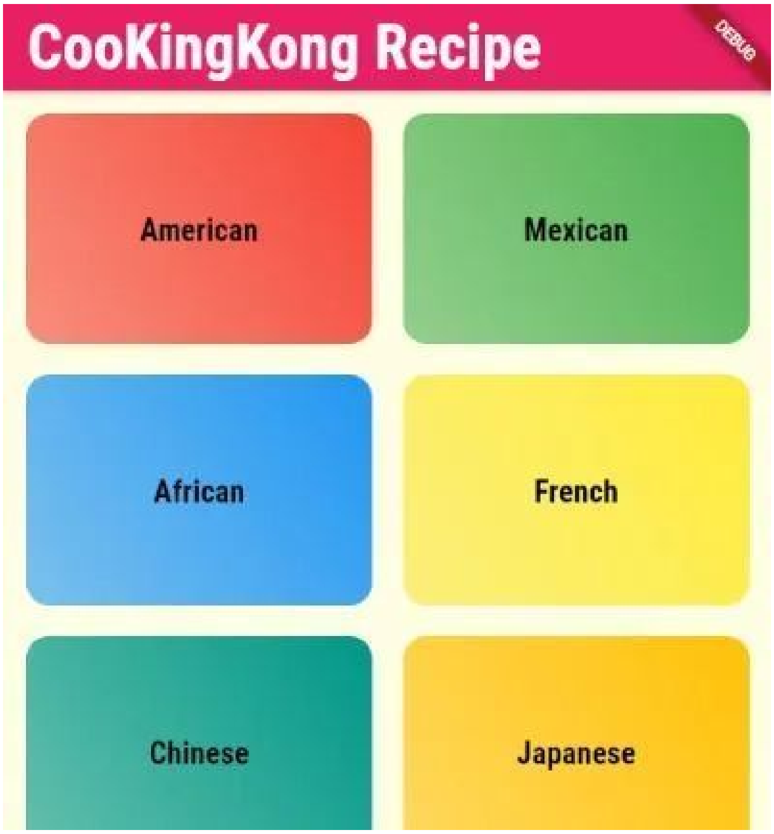


Figure 6.22 – Flutter navigation send data_ categories page

Now, we can tap any category. As a result, all the food items will show up.

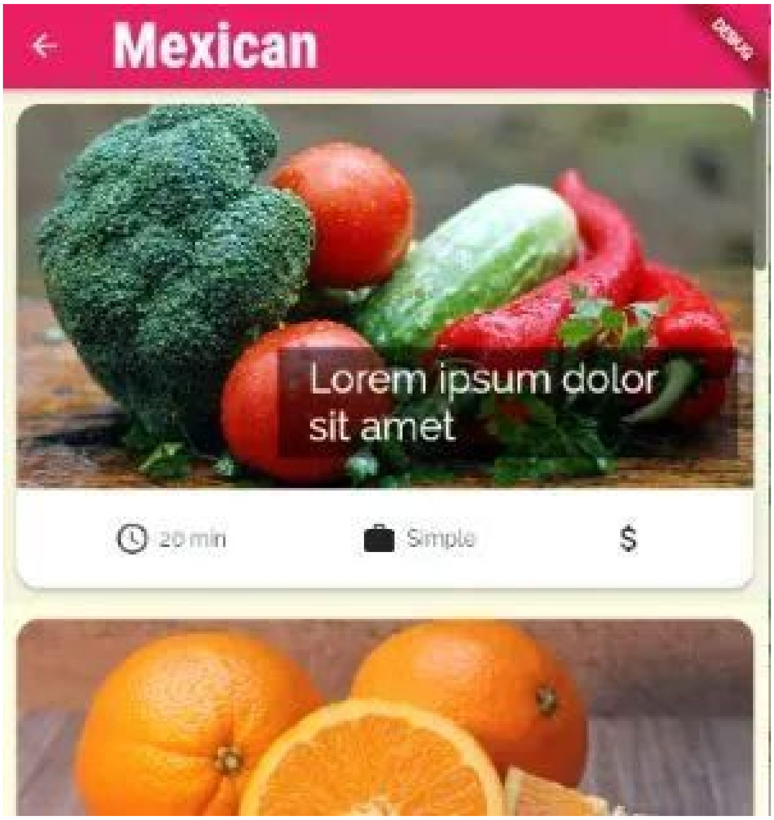


Figure 6.23 – Model view controller architecture first Example

After that, we can click any food item to see the detail of recipe. The page will display all the properties.

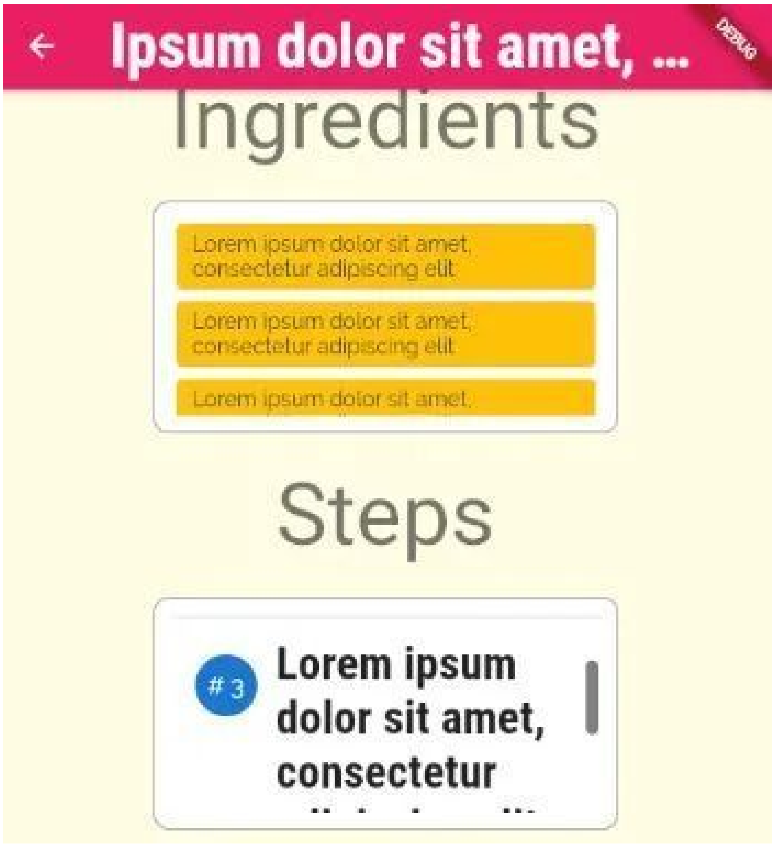

Ipsum dolor sit amet, ...
DEBUG
Ingredients
Lorem ipsum dolor sit amet, consectetur adipiscing elit
Lorem ipsum dolor sit amet, consectetur adipiscing elit
Lorem ipsum dolor sit amet,
Steps
# 3
Lorem ipsum dolor sit amet, consectetur

Figure 6.24 – Flutter List iterate example three

But we cannot either order the item, or add the item to cart. To do that, we can change the design of the food item controller where we have used the icons.

Consequently, the look of the page will change.

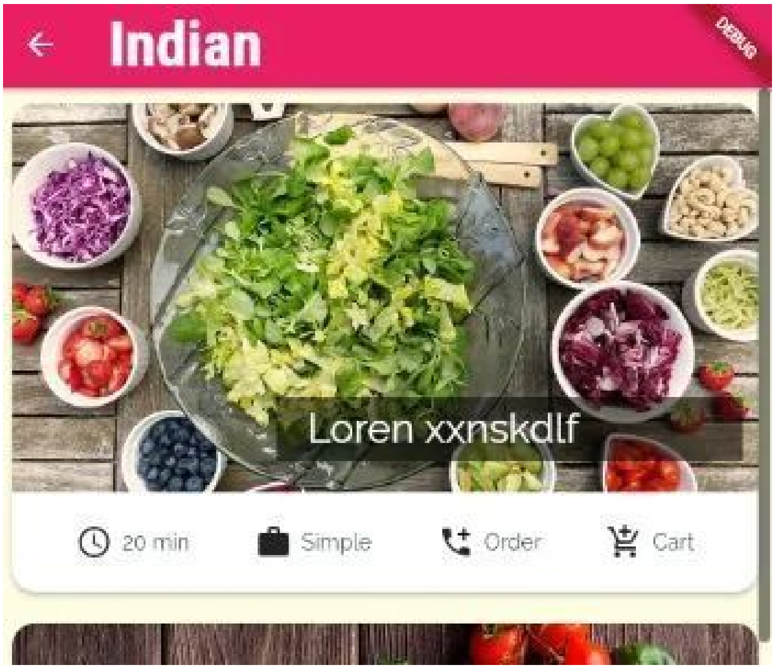


Figure 6.25 – Route Flutter final step first example

Now, from the food item page we can either order, or add the item to cart to order later.

However, to do that, we need to create separate route. So that we will reach the order page and cart page accordingly. Right?

But, at a glance, it looks difficult.

Why?

Because every food item is under the InkWell Widget which is a rectangular area of a Material that responds to touch.

As a result, if we click the icon, it will take us to the individual food page.

Let us take a look at the previous code.

```
import 'package:flutter/material.dart';

import '../model/food.dart';
import '/view/individual_food_page.dart';
import '/view/individual_seller_page.dart';

class FoodItem extends StatelessWidget {
final String id;
final String title;
final String imageUrl;
final int duration;
final Complexity complexity;

const FoodItem({
    Key? key,
    required this.id,
    required this.title,
    required this.imageUrl,
    required this.complexity,
    required this.duration,
}) : super(key: key);

String get complexityText {
    switch (complexity) {
    case Complexity.simple:
        return 'Simple';
    case Complexity.complex:
        return 'Complex';
    default:
        return 'Unknown';
    }
}

void selectMeal(BuildContext context) {
    Navigator.of(context).pushNamed(
    IndiividualFoodPage.routeName,
    arguments: id,
    );
}

@override
Widget build(BuildContext context) {
    return InkWell(
    onTap: () => selectMeal(context),
    child: Card(
        shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(15),
        ),
        elevation: 4,
        margin: const EdgeInsets.all(10),
        child: Column(
        children: <Widget>[
            Stack(
            children: <Widget>[
                ClipRRect(
                borderRadius: const BorderRadius.only(
                    topLeft: Radius.circular(15),
                    topRight: Radius.circular(15),
                ),
                child: Image.network(
                    imageUrl,
                    height: 250,
                    width: double.infinity,
                    fit: BoxFit.cover,
                ),
                ),
                Positioned(
                bottom: 20,
                right: 10,
                child: Container(
                    width: 300,
                    color: Colors.black54,
                    padding: const EdgeInsets.symmetric(
                    vertical: 5,
                    horizontal: 20,
                    ),
                    child: Text(
                    title,
                    style: const TextStyle(
                        fontSize: 26,
                        color: Colors.white,
                    ),
                    softWrap: true,
                    overflow: TextOverflow.fade,
                    ),
                ),
                )
            ],
            ),
            Padding(
            padding: const EdgeInsets.all(20),
            child: Row(
                mainAxisAlignment: MainAxisAlignment.spac\
eAround,
                children: <Widget>[
                Row(
                    children: <Widget>[
                    const Icon(
                        Icons.schedule,
                    ),
                    const SizedBox(
                        width: 6,
                    ),
                    Text('$duration min'),
                    ],
                ),
                Row(
                    children: <Widget>[
                    const Icon(
                        Icons.work,
                    ),
                    const SizedBox(
                        width: 6,
                    ),
                    Text(complexityText),
                    ],
                ),
                Row(
                    children: <Widget>[
                    GestureDetector(
                        onTap: () {
                        Navigator.push(
                            context,
                            MaterialPageRoute(
                            builder: (context) =>
                                const IndividualSellerPag\
e(),
                            ),
                        );
                        },
                        child: const Icon(
                        Icons.manage_search_rounded,
                        ),
                    ),
                    const SizedBox(
                        width: 6,
                    ),
                    ],
                ),
                ],
                ),
                ),
            ],
            ),
        ),
        );
}
}
```

The above code shows it clearly. Every icon is under the InkWell Widget. Now we can add two Row Widgets that will display two more icons.

However, that will again come under the InkWell widget that uses a navigation.

```
void selectMeal(BuildContext context) {
    Navigator.of(context).pushNamed(
    IndiividualFoodPage.routeName,
    arguments: id,
    );
}
...
@override
Widget build(BuildContext context) {
    return InkWell(
    onTap: () => selectMeal(context),
    child: Card(
...
```

How can we solve this problem?

To solve this problem, we need to wrap the icons with Gesture Detector Widget that detects gestures.

So we have to write the above code as follows.

```
import 'package:coo_king_kong/view/add_to_cart_page.dart';
import 'package:flutter/material.dart';

import '../model/food.dart';
import '/view/individual_food_page.dart';
import '/view/individual_seller_page.dart';

class FoodItem extends StatelessWidget {
final String id;
final String title;
final String imageUrl;
final int duration;
final Complexity complexity;

const FoodItem({
    Key? key,
    required this.id,
    required this.title,
    required this.imageUrl,
    required this.complexity,
    required this.duration,
}) : super(key: key);

String get complexityText {
    switch (complexity) {
    case Complexity.simple:
        return 'Simple';
    case Complexity.complex:
        return 'Complex';
    default:
        return 'Unknown';
    }
}

void selectMeal(BuildContext context) {
    Navigator.of(context).pushNamed(
    IndiividualFoodPage.routeName,
    arguments: id,
    );
}

@override
Widget build(BuildContext context) {
    return InkWell(
    onTap: () => selectMeal(context),
    child: Card(
        shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(15),
        ),
        elevation: 4,
        margin: const EdgeInsets.all(10),
        child: Column(
        children: <Widget>[
            Stack(
            children: <Widget>[
                ClipRRect(
                borderRadius: const BorderRadius.only(
                    topLeft: Radius.circular(15),
                    topRight: Radius.circular(15),
                ),
                child: Image.network(
                    imageUrl,
                    height: 250,
                    width: double.infinity,
                    fit: BoxFit.cover,
                ),
                ),
                Positioned(
                bottom: 20,
                right: 10,
                child: Container(
                    width: 300,
                    color: Colors.black54,
                    padding: const EdgeInsets.symmetric(
                    vertical: 5,
                    horizontal: 20,
                    ),
                    child: Text(
                    title,
                    style: const TextStyle(
                        fontSize: 26,
                        color: Colors.white,
                    ),
                    softWrap: true,
                    overflow: TextOverflow.fade,
                    ),
                ),
                )
            ],
            ),
            Padding(
            padding: const EdgeInsets.all(20),
            child: Row(
                mainAxisAlignment: MainAxisAlignment.spac\
eAround,
                children: <Widget>[
                Row(
                    children: <Widget>[
                    const Icon(
                        Icons.schedule,
                    ),
                    const SizedBox(
                        width: 6,
                    ),
                    Text('$duration min'),
                    ],
                ),
                Row(
                    children: <Widget>[
                    const Icon(
                        Icons.work,
                    ),
                    const SizedBox(
                        width: 6,
                    ),
                    Text(complexityText),
                    ],
                ),
                Row(
                    children: <Widget>[
                    GestureDetector(
                        onTap: () {
                        Navigator.push(
                            context,
                            MaterialPageRoute(
                            builder: (context) =>
                                const IndividualSellerPag\
e(),
                            ),
                        );
                        },
                        child: Row(
                        children: const [
                            Icon(Icons.add_call),
                            SizedBox(
                            width: 6,
                            ),
                            Text('Order'),
                        ],
                        ),
          ),
          ],
        ),
        Row(
          children: <Widget>[
          GestureDetector(
            onTap: () {
            Navigator.push(
              context,
              MaterialPageRoute(
              builder: (context) => const A\
ddToCartPage(),
              ),
            );
            },
            child: Row(
            children: const [
              Icon(
              Icons.add_shopping_cart,
              ),
              SizedBox(
              width: 6,
              ),
              Text('Cart'),
            ],
            ),
          ),
          ],
        ),
        ],
      ),
      ),
    ],
    ),
  ),
    );
}
}
```

Route Flutter tips and tricks

Therefore we have adjusted our new navigation that we have not defined earlier.

But it works.

Now, as we tap the order icon, it takes us to the order page.

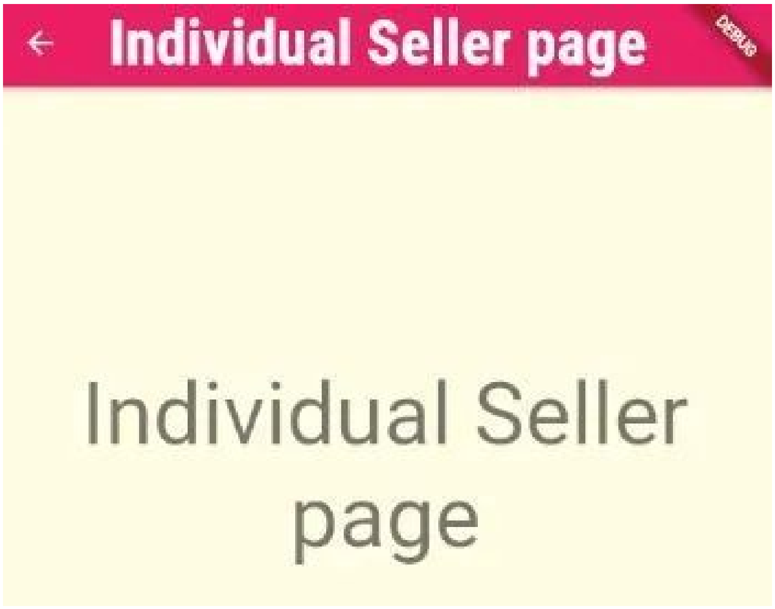


Figure 6.26 – Route Flutter final step second example

The same is true for the cart icon. For example, we can tap the cart icon and it takes us to the cart page.

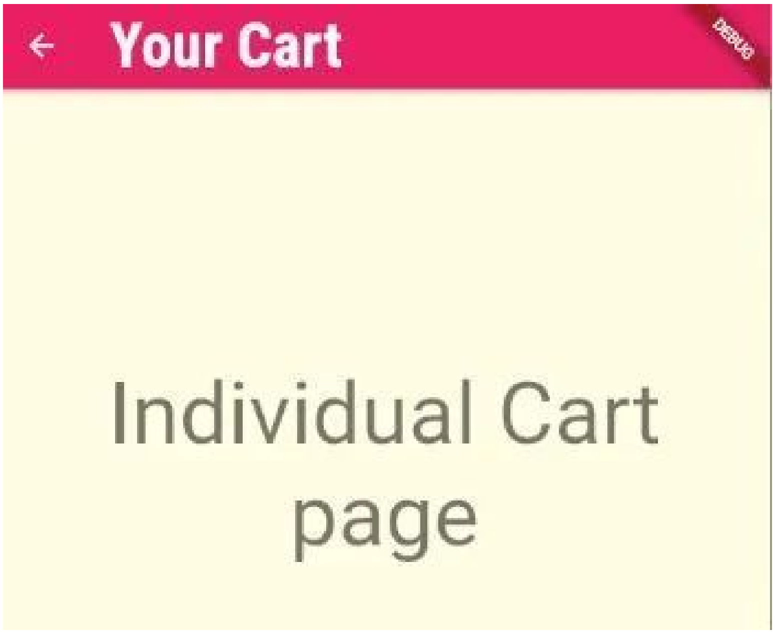


Figure 6.27 – Route Flutter final step third example

Certainly, we can convert this app to an E-Commerce App. We can add more functions.

But at present, we will stop here. You will find more Flutter App examples under one category.

If you want to clone the final step please visit this GitHub repository.

- For full code snippet please visit the respective GitHub Repository - [19]
- Read updated articles on Flutter, Dart, and Algorithm - [20]

[19]https://github.com/sanjibsinha/coo_king_kong/tree/final-step-add-to-cart
[20]https://sanjibsinha.com

7. Let’s learn how a Weather App uses API and serializes JSON data

A step by step guide to build a Current Weather Tracker App.

Future then, aync, await, API, JSON: Let’s build a Current Weather Tracker App

There are plugins like “Geolocator” and “http” that will easily connect through API and track live current Weather data.

However, we need to understand a few important concepts such as Future then, aync, await, and API, JSON in Flutter.

We will build the Current Weather Tracker App step-by-step. And, finally, we will see the output at the end.

However, you can clone each step from the respective GitHub repoitory branches.

Let’s name the App - “WithHer”.

Future Flutter: WithHer App – Step 1

We are going to build a Weather App. As before, we’ll build it step by step. And this is our first step where we will understand a key concept – Future in Flutter.

But to start with let me assure you one thing first.

It does not take much effort to build a weather app. Let's name it "WithHer".

Why it is not difficult?

Because we will build this app with a Flutter Geolocator plugin. In this first step, first we'll see how we can get the location. As a result, we will print the latitude and longitude.

After that, we will discuss the role of Future in Flutter. By the way, the Future class returns the result of an asynchronous programme.

Why we need to understand the role of Future in getting the location?

We'll see it in a minute.

However, besides, we need to understand what is asynchronous programming. In addition, we will have to understand two keywords – async and await.

How to the get the location in Flutter? Firstly, we need to add the dependency of the Geolocator plugin in our "pubspec.yaml" file.

```
dependencies:
flutter:
    sdk: flutter

cupertino_icons: ^1.0.2
geolocator: ^8.2.0
```

Secondly, in our Flutter app, we need to import the plugin where we need it.

To make it simple, we will test our weather app in the top-level main() file.

```dart
import 'package:geolocator/geolocator.dart';
```

How do we know a location?

Certainly the latitude and the longitude help us to know the exact location.

And the Geolocator plugin helps us to control how exact we want to be. However, we will keep our priority at the lowest level.

Why?

Because if we want to get the exact location of the user, it may take more time. For that reason, we will make it the lowest. So that, within a radius of 500 meters we can get the weather update.

With reference to that, we must remember one thing. The Geolocator plugin gets the data from internet.

Therefore, it may take an uncertain time. Moreover, the plugin has curtailed our job. If we want to do the low level plumbing and write the whole code we have to write long lines of code.

In addition, we need a certain level of expertise.

But with a few lines of code, the Geolocator plugin has done the same task.

```dart
void getLocation() async {
    Position position = await Geolocator.getCurrentPositi\
on(
        desiredAccuracy: LocationAccuracy.lowest);
    print(position);
}
```

Actually we're riding the Geolocator plugin's back and shoulders and curtail the heavy task. Right?

Geolocator plugin makes our life easier

Let us see the full code snippet now.

```
import 'package:flutter/material.dart';
import 'package:geolocator/geolocator.dart';

void main() {
runApp(const MyApp());
}

class MyApp extends StatelessWidget {
const MyApp({Key? key}) : super(key: key);

// This widget is the root of your application.
@override
Widget build(BuildContext context) {
return MaterialApp(
    title: 'Flutter Demo',
    theme: ThemeData(
    primarySwatch: Colors.blue,
    ),
    home: const MyHomePage(title: 'Flutter Demo Home Page\
'),
);
}
}

class MyHomePage extends StatefulWidget {
const MyHomePage({Key? key, required this.title}) : super\
(key: key);

final String title;
```

```

@override
State<MyHomePage> createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
void getLocation() async {
Position position = await Geolocator.getCurrentPosition(
    desiredAccuracy: LocationAccuracy.lowest);
print(position);
}

@override
Widget build(BuildContext context) {
return Scaffold(
    appBar: AppBar(
    title: Text(widget.title),
    ),
    body: Center(
    child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
        const Text(
            'We\'re trying to find the Lat and Lang:',
        ),
        Text(
            'Location',
            style: Theme.of(context).textTheme.headline4,
        ),
        ],
    ),
    ),
    floatingActionButton: FloatingActionButton(
    onPressed: () {
        getLocation();
```

```
      },
      tooltip: 'Get Location',
      child: const Icon(Icons.location_on),
      ), // This trailing comma makes auto-formatting nicer\
 for build methods.
  );
  }
  }
```

As we see, the getLocation() method uses two keywords "async" and "await". Later we have called that method through the onPressed property of the floating action button.

If you want to follow the steps, please clone this branch of GitHub repository.

As we see the above code we find that the two keywords "async" and "await" play an important role here.

We'll come back to that point in a minute. Before that let's run the app.

Let's run the app first.

If we click the floating action button, we will get the latitude and longitude.

However, without the user's permission we cannot get her location. Right?

Therefore, if we click the floating action button to get the location, the app will seek permission of the user.

Click allow to proceed. And in return it will print the latitude and longitude in the terminal.

```
Latitude: 22.5892652, Longitude: 88.3056119
```

Our WithHer App is working perfectly. Now, the time has come to understand what is Future in Flutter. In addition, what is the relation between the Future class and asynchronous programming.

What is asynchronous programming?

Firstly, there are two types of programming. Synchronous and asynchronous.

We need to understand why we need asynchronous programming?

Take a look at the following code which represents synchronous programming.

```
/// an example of synchronous programme
///

void main() {
callEveryTask();
}

void callEveryTask() {
doThisFirst();
doThisSecond();
doThisThird();
}

void doThisFirst() {
print('Doing it first');
}

void doThisSecond() {
print('Doing it second');
}

void doThisThird() {
print('Doing it third');
}
```

When we run the above code, the execution will take synchronously. That means it will execute the first function. Then the second, and lastly it executes the third function.

```
Output:

Doing it first
Doing it second
Doing it third
```

The code execution is sequential. But what will happen, if the second function takes much time to execute?

In that case, the third function will have to wait until the second function finishes its task. Right?

Now, consider a case, where the second function is downloading a big file, or gets an image from the internet.

In such cases, user will have to wait to get the result of the third function.

But we don't want this to happen.

Why?

It does not go with our principle. Because we want to give the user a good experience, we should allow the third function to go ahead and after that the second function may execute.

What is Future in Flutter?

We can achieve this with the help of the Future class in Flutter.

The Future class has a named constructor Future.delayed(). It runs the computation after a certain delay.

Let's use this Future constructor to delay the second function for 2 seconds.

```
/// an example of asynchronous programme
///

void main() {
callEveryTask();
}

void callEveryTask() {
doThisFirst();
doThisSecond();
doThisThird();
}

void doThisFirst() {
String result = 'First task completed.';
print('Doing it first');
}

String doThisSecond() {
String result = 'Second task completed.';
Duration duration = const Duration(seconds: 2);
Future.delayed(duration, () {
    print('Doing it second');
    result;
});
return result;
}

void doThisThird() {
String result = 'Third task completed.';
print('Doing it third');
}
```

As a result, when we run the code, the third function will execute after the first function executes.

Then, after 2 seconds the second function will execute.

```
output ##

Doing it first
Doing it third
Doing it second
```

This time, we have coded asynchronously. Although we have established a relation between the Future class and asynchronous programming, yet we need to know two keywords – "async" and "await".

What are async and await?

Just like any other TYPE in Dart, the Future is also a TYPE. As we have learned before that every class is a TYPE of an object, or instance of that class, Future is also a TYPE.

Why?

The reason is simple. It's a class. Right?

Therefore, we can change the TYPE of the second function from String to Future. But to do that, we have to use two keywords – async and await.

So our previous code changes as follows.

```dart
void main() {
callEveryTask();
}

void callEveryTask() async {
doThisFirst();
String secondTask = await doThisSecond();
doThisThird(secondTask);
}

void doThisFirst() {
String result = 'First task completed.';
print('Doing it first');
}

Future doThisSecond() async {
String result = '..second task completed..';
Duration duration = const Duration(seconds: 2);
await Future.delayed(duration, () {
    result;

    print('Please Wait ....');
});
return result.toString();
}

void doThisThird(String secondTask) {
String result = 'Third task completed.';
print('Doing it third with $secondTask');
}
```

We have tweaked the previous code a little bit. Now, we have forced the third function to take the output from the second function as its input.

As a result, we need to tell the second function that another function is waiting for your output. Therefore, if you have anything to do, do it now, However, after that, pass the output to the third function.

As a result, we get the following output.

```
output ##

Doing it first
Please Wait ....
Doing it third with ..second task completed..
```

Now, as the second function stops for 2 seconds, the user gets a message like "Please Wait ....".

Certainly it gives the user a better experience. Because the user knows that she has to wait, she will wait patiently.

This is an introduction to Future and asynchronous programming. Later while we build the weather app, we will discuss it more.

Flutter State: WithHer App – Step 2

We have been building a weather App. In the first step, the "With-Her" App has used the Geolocator plugin. In addition, we have also used a Stateful Widget.

Why we have used a Stateful Widget? We will realise as we progress.

But before that, we need to understand a few important concepts that revolve around the State class.

Firstly, the State class is an abstract class that defines the logic and internal state for a Stateful Widget.

Secondly, it does not act like a Stateless Widget.

What is the main difference? During the lifetime of a Stateless Widget, it does not change. On the contrary, it destroys the old instance and creates a new instance.

But in its lifetime, a Stateful Widget may change its information.

Let us consider a Flutter App that has two pages. We can use the "routes" property to mention the path.

```
import 'package:flutter/material.dart';
import 'first_page.dart';
import 'my_app_home.dart';
import 'second_page.dart';

class MyApp extends StatelessWidget {
const MyApp({Key? key}) : super(key: key);

// This widget is the root of your application.
@override
Widget build(BuildContext context) {
    return MaterialApp(
    title: 'Flutter Demo',
    theme: ThemeData(
        primarySwatch: Colors.blue,
    ),
    //home: const MyAppHome(title: 'Flutter Demo Home Pag\
e'),
    initialRoute: '/',
    routes: {
        '/': (context) => const FirstPage(),
        SecondPage.routeName: (context) => const SecondPa\
ge(),
    },
    );
}
}
```

Certainly the First page is a Stateless widget which has a Text Button that redirects us to the second page.

```
import 'package:flutter/material.dart';
import 'package:with_her/view/second_page.dart';

class FirstPage extends StatelessWidget {
const FirstPage({Key? key}) : super(key: key);

static const routeName = '/first-page';

@override
Widget build(BuildContext context) {
    return Scaffold(
    appBar: AppBar(
        title: const Text('First Page'),
    ),
    body: Center(
        child: Container(
        margin: const EdgeInsets.all(20.0),
        width: 400.0,
        height: 150.0,
        child: TextButton(
            onPressed: () {
            Navigator.push(
                context,
                MaterialPageRoute(
                builder: (context) => const SecondPage(),
                ),
            );
            },
            child: const Text('Go to Second Page'),
        ),
        ),
    ),
    );
}
}
```

On the other hand, the second page is a Stateful Widget with a Text Button that on pressing takes us back to the First page again.

The second page has the following code.

```
import 'package:flutter/material.dart';

class SecondPage extends StatefulWidget {
const SecondPage({Key? key}) : super(key: key);

static const routeName = '/second-page';

@override
State<SecondPage> createState() => _SecondPageState();
}

class _SecondPageState extends State<SecondPage> {
@override
void initState() {
    // TODO: implement initState
    super.initState();
    print('I am called first, after each State object has\
 been created.');
}

@override
void deactivate() {
    // TODO: implement deactivate
    super.deactivate();
    print('The state object removed from the tree');
}

@override
Widget build(BuildContext context) {
    print('I am called after initState() method.');
    return Scaffold(
    appBar: AppBar(
        title: const Text('Second Page'),
    ),
    body: Center(
        child: Container(
        margin: const EdgeInsets.all(20.0),
        width: 400.0,
        height: 150.0,
        child: TextButton(
            onPressed: () {
            Navigator.pop(context);
            },
            child: const Text('Go to Frist Page'),
        ),
        ),
    ),
    );
}
}
```

How Flutter State works

As we come to the second page, two methods fire immediately.

In our terminal we have got two outputs.

```
I am called first, after each State object has been creat\
ed.
I am called after initState() method.
```

The first output comes from the initState() method. And the second output comes from the build() method.

However, when we press the back button to go back to the first page, another output comes in.

```
The state object removed from the tree
This output comes from the deactivate() method.
```

As a result, we have a clear picture how the life cycle of the Stateful Widget starts and ends.

As we progress with our weather app, we will discuss this topic again. But now we have a basic understanding of how the Stateful widget works.

API Flutter: WithHer App – Step 3

How can we use API in Flutter? In fact, only with API we can build the weather App "WithHer". Without the API? We cannot build it.

Therefore, firstly, we need to know what is API? Secondly, how we can use the API so that we can build the weather App.

As a result, we would get the current weather of any city in the world.

Till now, we have been progressing a little bit. We have learned how we can use Future in Flutter.

After that, we have also discussed the life cycle of Stateful Widget.

Why?

Because Stateful Widget will play an important role in using the API that will help us to get the current weather.

What is an API?

An API is a short name for Application Programming Interface.

But, to clarify, an API is actually a type of software.

What kind of software is it?

API is a kind of software that acts as an intermediary between two applications. In short, an API can connect two applications.

In a wider sense, an API can connect two computers so that they can communicate.

If we go to read the Flutter documentation, it says as follows:

Welcome to the Flutter API reference documentation!

In fact, when we use Flutter framework, we use API all the time. But we do that without knowing.

For example, Flutter and Dart gives us plenty of classes, methods, libraries to use.

Why?

The reason is simple. We do not have to write everything from scratch. Instead, we can use them and build any app in a very short period of time.

The same thing happens in our weather app, we will get the data from the open weather map website.

We can go to that website anytime, and check the current weather of any place.

Our "WithHer" App will fetch the weather data from there. And, certainly, with the help of the API that they provide.

Without that API, our App cannot display the current data.

How can we use API in Flutter?

We will see in a minute how we can use API in Flutter. But, before that, we would like to see what we are going to build.

In the first screenshot, we will see two Text Widgets. They display a static text.

But below the Text Widgets, we have a Text Button. Once we press the Button, the Text Widgets display the name of the city, and the current weather there.

In our previous section, we have seen how we can get the latitude and longitude.

For that we used the Geolocator plugin. Right?

Actually, in this case, we get the city name and current weather with the help of latitude and longitude. But, with that, we need the API key.

So that, we can get the weather as follows.

By the way, you may wonder about the name of the city.

Kara-Kulja is the center of Kara-Kulja District in Osh Region of Kyrgyzstan.

In the morning, there was nice weather. Clear blue sky. I had seen in my mind-eyes.

API and Format Exception error

Firstly, we need to register at the open weather map website to get the API key, or app ID.

Secondly, we need to use the “http” plugin and add the dependency in our “puspec.yaml” file.

```
dependencies:
flutter:
    sdk: flutter

cupertino_icons: ^1.0.2
geolocator: ^8.2.0
http: ^0.13.4
```

Finally, we need to import the package in our file.

Then, firstly you register at the open weather map website, and get your own API key. After that you will find that as we request for data, the API responds and sends data in JSON format.

We will discuss JSON in our next section. Otherwise, we will not understand why many people get the format exception error.

At present, we will take a look at the full code.

```
import 'dart:convert';
import 'dart:ui';
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;

class MyAppHome extends StatefulWidget {
const MyAppHome({Key? key, required this.title}) : super(\
key: key);

final String title;

@override
State<MyAppHome> createState() => _MyAppHomeState();
}

class _MyAppHomeState extends State<MyAppHome> {
String apiKey = 'Your Key';
double lat = 40.7128;
double lon = 74.0060;

Future<String> getCityName() async {
    final httpsUri = Uri.http('api.openweathermap.org', '\
/data/2.5/weather', {
    'lat': '$lat',
    'lon': '$lon',
    'appid': apiKey,
    });

    var request = await http.get(httpsUri);
    if (request.statusCode == 200) {
    String data = request.body.toString();
    var city = jsonDecode(data)['name'];

    return city;
    } else {
    return '${request.statusCode}';
    }
}

Future<String> getWeather() async {
    final httpsUri = Uri.http('api.openweathermap.org', '\
/data/2.5/weather', {
    'lat': '$lat',
    'lon': '$lon',
    'appid': apiKey,
    });

    var request = await http.get(httpsUri);
    if (request.statusCode == 200) {
    String data = request.body.toString();
    var description = jsonDecode(data)['weather'][0]['des\
cription'];

```

```
    return description;
    } else {
    return '${request.statusCode}';
    }
}

String city = 'Your ';
String description = 'Good Weather ';

@override
Widget build(BuildContext context) {
    //var city = getData();
    return Scaffold(
    appBar: AppBar(
        title: const Text('Get City and Weather'),
    ),
    body: Center(
        child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
            Text(
            'Welcome to ${city.toString()} city!',
            style: Theme.of(context).textTheme.headline6,
            ),
            Text(
            description.toString(),
            style: Theme.of(context).textTheme.headline4,
            ),
            const SizedBox(
            height: 20.0,
            ),
            TextButton(
            onPressed: () {
                getCityName().then((String result) {
                setState(() {
                    city = result;
                });
                });
                getWeather().then((String result) {
                setState(() {
                    description = result;
                });
                });
            },
            child: const Text(
                'Get City and Current Weather',
                style: TextStyle(
                color: Colors.white,
                fontSize: 30.0,
                fontWeight: FontWeight.bold,
                ),
            ),
            style: ButtonStyle(
                backgroundColor: MaterialStateProperty.al\
l(Colors.red),
                shape: MaterialStateProperty.all(
                RoundedRectangleBorder(
                    borderRadius: BorderRadius.circular(3\
0.0),
                ),
                ),
            ),
            )
        ],
        ),
    ),
    // This trailing comma makes auto-formatting nicer fo\
r build methods.
    );
}
```

```
}
```

We have hard coded the latitude, longitude and the app ID.

As we progress, we will take the inputs and pass them dynamically.

But, till now, we have fetched the data from the open weather map website with the help of API.

Without API, we cannot use the data as follows.

```
final httpsUri = Uri.http('api.openweathermap.org', '/dat\
a/2.5/weather', {
    'lat': '$lat',
    'lon': '$lon',
    'appid': apiKey,
    });
```

We cannot request the data as follows.

```
var request = await http.get(httpsUri);
```

After that, with the help of Dart convert package, we have converted the JSON data to a human readable format.

```
import 'dart:convert';
...
String data = request.body.toString();
    var description = jsonDecode(data)['weather'][0]['des\
cription'];
```

Overall, we have progressed a lot to display the current weather data. In the next sections, we will progress more.

Stay tuned.

JSON Flutter: WithHer App – Step 4

As we progress, we find that we need to format structured data to run the weather app. In fact, in many Flutter app, at some point we need to format JSON data.

In our "WithHer" App, we have done that.

It proves that our weather app is working. As a result, we have been able to format the JSON data that comes from the open weather map website API.

In our previous section we have discussed the role of API in Flutter.

And before that, we have discussed why we need a Stateful Widget and what is Future in Flutter.

We need to understand them for the sake of our weather app. For example, without API, Stateful Widget, and Future, we cannot build this weather App.

Now, we need to know how we format the JSON data.

Firstly, we will know what does the term JSON mean? Secondly, we will learn how we have formatted the JSON data.

What is JSON in Flutter?

Firstly, let us see a small JSON data which will give us an idea about how it looks.

```
import 'dart:convert';

void main() {
var userProfile = {
"name": "Json Web",
"profession": "Spy",
"location": "Unknown"
};

String jsonString = jsonEncode(userProfile);


Map<String, dynamic> user = jsonDecode(jsonString);
print('${user['name']} is a ${user['profession']} and his\
 location'
    ' is ${user['location']}.');

}
// // Json Web is a Spy and his location is Unknown.
```

The above code shows that JSON data is a structured data that we can encode into a String and later decode.

But we can always make this code better.

Therefore, we can use a User model to avoid the compile time error.

```
import 'dart:convert';

class User {
final String? name;
final String? profession;
final String? location;


User(this.name, this.profession, this.location);

User.fromJson(Map<String, dynamic> json)
    : name = json['name'],
        profession = json['profession'],
        location = json['location'];

Map<String, dynamic> toJson() => {
        'name': name,
        'profession': profession,
        'location': location,
    };
}

void main() {
User userProfile = User('Json Web', 'Spy', 'Unknown');

String json = jsonEncode(userProfile);

Map<String, dynamic> userMap = jsonDecode(json);
var user = User.fromJson(userMap);


print('${user.name} is a ${user.profession} and his locat\
ion'
    ' is ${user.location}.');

}

// Json Web is a Spy and his location is Unknown.
```

In that case, our code will be more type safe which we want. But in our weather App, we can decode the JSON data.

The weather map JSON data and user location model class In our weather App, we have added the dependency of the "http" plugin in our "pubspec.yaml" file.

Why?

Because we will request the open weather map API to get the data in JSON format.

After that, we have used the Dart convert package to decode JSON data.

Let us see the user location model class firstly.

```
import 'dart:convert';
import 'package:http/http.dart' as http;

class UserLocation {
String apiKey = 'Secret Key';
double lat = 40.7128;
double lon = 74.0060;

Future<String> getCityName() async {
    final httpsUri = Uri.http('api.openweathermap.org', '\
/data/2.5/weather', {
    'lat': '$lat',
    'lon': '$lon',
    'appid': apiKey,
    });

    var request = await http.get(httpsUri);
    if (request.statusCode == 200) {
    String data = request.body.toString();
    var city = jsonDecode(data)['name'];

    return city;
    } else {
    return '${request.statusCode}';
    }
}
Future<String> getWeather() async {
    final httpsUri = Uri.http('api.openweathermap.org', '\
/data/2.5/weather', {
    'lat': '$lat',
    'lon': '$lon',
    'appid': apiKey,
    });

    var request = await http.get(httpsUri);
    if (request.statusCode == 200) {
    String data = request.body.toString();
    var description = jsonDecode(data)['weather'][0]['des\
cription'];

    return description;
    } else {
    return '${request.statusCode}';
    }
}
}
```

Secondly, we will take a look at how the open weather map website sends us the current weather data in JSON format.

For example, we have typed any city name, then the website will show the sample JSON data as follows.

```
{
"coord": {
    "lon": -122.08,
    "lat": 37.39
},
"weather": [
    {
    "id": 800,
    "main": "Clear",
    "description": "clear sky",
    "icon": "01d"
    }
],
"base": "stations",
"main": {
    "temp": 282.55,
    "feels_like": 281.86,
    "temp_min": 280.37,
    "temp_max": 284.26,
    "pressure": 1023,
    "humidity": 100
},
"visibility": 10000,
"wind": {
    "speed": 1.5,
    "deg": 350
},
"clouds": {
    "all": 1
},
"dt": 1560350645,
"sys": {
    "type": 1,
    "id": 5122,
    "message": 0.0139,
    "country": "US",
    "sunrise": 1560343627,
    "sunset": 1560396563
},
"timezone": -25200,
"id": 420006353,
"name": "Mountain View",
"cod": 200
}
```

As a result, when we request the data with our APP key, or app ID, it will send us the data in the above format.

Certainly, we need to provide the latitude and longitude also.

As we progress, we will make it simple.

But at present it shortens the view page. Because we have defined the properties and methods in the model class.

The Object Oriented Approach

Now we can create a user location object. And, after that, we can access the methods.

```
UserLocation location = UserLocation();
```

It makes our code more object oriented than before.

There is one problem that we need to solve later.

What is that?

We still hard code the data. Instead we can make them dynamic.

Let's see the landing page.

```
import 'package:flutter/material.dart';

import '../model/location.dart';

/// expleining json
///
class MyAppHome extends StatefulWidget {
const MyAppHome({Key? key, required this.title}) : super(\
key: key);

final String title;

@override
State<MyAppHome> createState() => _MyAppHomeState();
}

class _MyAppHomeState extends State<MyAppHome> {
var city = 'Your ';
var description = 'Good Weather ';

@override
Widget build(BuildContext context) {
    UserLocation location = UserLocation();

    return Scaffold(
    appBar: AppBar(
        title: const Text('Get City and Weather'),
    ),
    body: Center(
        child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
            Text(
            'Welcome to ${city.toString()} city!',
            style: Theme.of(context).textTheme.headline6,
            ),
            Text(
            description.toString(),
            style: Theme.of(context).textTheme.headline4,
            ),
            const SizedBox(
            height: 20.0,
            ),
            TextButton(
            onPressed: () {
                location.getCityName().then((result) {
                setState(() {
                    city = result;
                });
                });
                location.getWeather().then((result) {
                setState(() {
                    description = result;
                });
                });
            },
            child: const Text(
                'Get City and Current Weather',
                style: TextStyle(
                color: Colors.white,
                fontSize: 30.0,
                fontWeight: FontWeight.bold,
                ),
            ),
            style: ButtonStyle(
                backgroundColor: MaterialStateProperty.al\
l(Colors.red),
                shape: MaterialStateProperty.all(
                RoundedRectangleBorder(
                    borderRadius: BorderRadius.circular(3\
0.0),
                ),
                ),
            ),
            )
        ],
        ),
    ),
    // This trailing comma makes auto-formatting nicer fo\
r build methods.
    );
}
}
// you can clone this Step
```

The code repositories for this branch[21]

In the above code, we have used a special Future method.

In the next section, we will dig deeper to know that.

Future Then: WithHer App – Step 5

We told before that we would come back and discuss the role of the "Future then" method in Flutter. Therefore, here we are.

In this step, we have accomplished many tasks. Certainly, we have done that with a special Future method – then().

Future, then, async, and await

In our App, why do we need the "Future then" method? What is the difference with the keywords – "Future async, and await"?

[21]https://github.com/sanjibsinha/with_her/tree/fourth-step-json-explained

In this section we will find the answers. In addition, we have also changed the location class in our weather App.

As a result, our weather app is now getting the latitude and longitude from the Geolocator plugin. We’re no longer hard coding the value.

Now our “WithHer” App becomes dynamic. We will discuss that part in the next section.

In this part we will concentrate on Future class.

Why?

Because the Future class plays an important role in our App.

What is Future.then() method?

We have learned that Flutter and Dart is single thread. It does all the work on its main thread. It has no worker thread that runs parallel.

The main thread handles small tasks. When it simultaneously performs heavy task, it may hang, or freeze.

That is why the multi thread concept comes up.

Now, the question is, how Flutter manages this heavy task?

The answer is asynchronous programming. In synchronous programming, everything goes step by step. One after another.

Consequently, if a heavy task falls in the middle, or in the beginning, the whole program halts.

Asynchronous programming solves that issue. Flutter and Dart lets the small tasks to perform before the heavy task. And the heavy task is performed in the background.

Why the Future class is important?

To understand the importance of Future class, let us see a Dart program first where we are downloading a dummy weather data.

The time to download the data takes 8 seconds.

Therefore when we call this function it takes 8 seconds and after that it prints the output.

```
void main() {
performAnotherHevyTaskWithThen();
print('main thread starts....');
print('main thread ends....');
}

void performAnotherHevyTaskWithThen() {
Future<String> result = getWeatherDataUsingAnotherAPI();
result.then((value) {
    print('The content of the file with Future then - $va\
lue');
});
}

Future<String> getWeatherDataUsingAnotherAPI() {
Future<String> result = Future.delayed(const Duration(sec\
onds: 8), () {
    return 'Downloading File completed after 8 seconds.';
});
return result;
}

/**
OUTPUT:

main thread starts....
main thread ends....
```

```
The content of the file with Future then - Downloading Fi\
le completed after 8 seconds.

*
*
*/
```

In the above code, we have used the "Future then" method. But before that we have told Dart to perform a heavy task that takes 8 seconds to finish.

```
Future<String> getWeatherDataUsingAnotherAPI() {
Future<String> result = Future.delayed(const Duration(sec\
onds: 8), () {
    return 'Downloading File completed after 8 seconds.';
});
return result;
}
```

After that, we use the "Future then" method to call the above function.

Watch this part.

```
void performAnotherHevyTaskWithThen() {
Future<String> result = getWeatherDataUsingAnotherAPI();
result.then((value) {
    print('The content of the file with Future then - $va\
lue');
});
}
```

Lastly, when we call this function in our main thread it executes after 8 seconds. However, Dart allows the small tasks to perform first.

```
void main() {
performAnotherHevyTaskWithThen();
print('main thread starts....');
print('main thread ends....');
}

/**
OUTPUT::::

main thread starts....
main thread ends....
The content of the file with Future then - Downloading Fi\
le completed after 8 seconds.

*
*
*/
```

Before the "Future async and await", we used the "Future then".

But the "Future async and await" makes it more simple.

Let us see the code where we use the "Future async and await" instead of "Future then".

```
void main() {
performAHevyTaskWithAsyncAndAwait();
print('main thread starts....');
print('main thread ends....');
}
void performAHevyTaskWithAsyncAndAwait() async {
String result = await getWeatherDataUsingAnAPI();
print('The content of the file with Future, async and awa\
it - $result');
}

```

```
Future<String> getWeatherDataUsingAnAPI() {
Future<String> result = Future.delayed(const Duration(sec\
onds: 6), () {
    return 'Downloading File completed after 6 seconds.';
});
return result;
}
/**
OUTPUT:::

main thread starts....
main thread ends....
The content of the file with Future, async and await - Do\
wnloading File completed after 6 seconds.

*
*
*/
```

The difference between Future then and async, await

As such, both type of Future functions program asynchronously. But there is a major difference.

In "Future then" method, although we return a String value, yet we cannot explicitly declare it.

As a result, we have to declare it as the Future<String> value.

```
void performAnotherHevyTaskWithThen() {
Future<String> result = getWeatherDataUsingAnotherAPI();
result.then((value) {
...
```

But, in case of "Future async, await" we confirm that we are getting a String value.

```
void performAHevyTaskWithAsyncAndAwait() async {
String result = await getWeatherDataUsingAnAPI();
...
```

Certainly that makes our task easier than the "Future then" method.

But in some cases, we still prefer to use the "Future then" method. We have done the same thing in our weather App. Take a look at the following image.

The page displays a static Text output that will change as we press the button.

The Geolocator plugin gets the current latitude and longitude dynamically. After that, based on that value, the user gets the current weather data.

The Text Button onPressed property updates the data using the "Future then" method.

```
TextButton(
            onPressed: () {
                location.getCitynameWithGeolocator().then\
((result) {
                setState(() {
                    city = result;
                });
                });
                location.getWeatherDescriptionWithGeoloca\
tor().then((result) {
                setState(() {
                    description = result;
                });
                });
            },
...
// code is incomplete for brevity
// please clone the project from this branch of GitHub Re\
pository
```

To sum up, the Future object acts as a promise token. When Flutter finds that some heavy task is waiting it promises the user that in Future you will get it. So please wait.

What is the advantage?

Many.

Because the Future object can return any data type. In the above code it returns String data type. But we can pass and return any data type with the help of the Future object.

The code repositories for this branch[22]

[22]https://github.com/sanjibsinha/with_her/tree/fifth-step-future-then

Future in Flutter: WithHer App – Step 6

While we've building the weather app, we have learned a few key concepts in Flutter. Whenever we get data from outside resource, we need to use Future in Flutter.

In the "WithHer" App, which we have been building, we have used Future. In fact, in the first step, we have introduced the concept of Future first.

After that, in the fifth step, we have discussed Future then, async and await in great detail.

To sum up, we define Future as a function in Flutter and Dart. But, instead of void, we use Future. When we return a value from Future, we pass it a Type.

In this section, we will finally show how we have organized our code in a model weather class with Future and return weather data in our view page.

What is Future in Flutter?

We have requested the data from open weather map website using their API.

After that, the website responded and posted data in JSON format.

As a result, in our location model class, we have defined all the properties and methods.

```
import 'dart:convert';
import 'package:geolocator/geolocator.dart';
import 'package:http/http.dart' as http;

/// this is sixth step
///

class UserLocation {
Future<String> getCitynameWithGeolocator() async {
    try {
    Position position = await Geolocator.getCurrentPositi\
on(
        desiredAccuracy: LocationAccuracy.lowest);
    double lat = position.latitude;
    double lon = position.longitude;
    return getCityName(lat, lon, 'Secret');
    } catch (e) {
    return e.toString();
    }
}

Future<String> getWeatherDescriptionWithGeolocator() asyn\
c {
    try {
    Position position = await Geolocator.getCurrentPositi\
on(
        desiredAccuracy: LocationAccuracy.lowest);
    double lat = position.latitude;
    double lon = position.longitude;
    return getWeatherDescription(lat, lon, 'Secret');
    } catch (e) {
    return e.toString();
    }
}

```

```
Future<String> getCityName(double lat, double lon, String\
 apiKey) async {
    final httpsUri = Uri.http('api.openweathermap.org', '\
/data/2.5/weather', {
    'lat': '$lat',
    'lon': '$lon',
    'appid': apiKey,
    });

    var request = await http.get(httpsUri);
    if (request.statusCode == 200) {
    String data = request.body.toString();
    var city = jsonDecode(data)['name'];

    return city;
    } else {
    return '${request.statusCode}';
    }
}

Future<String> getWeatherDescription(
    double lat, double lon, String apiKey) async {
    final httpsUri = Uri.http('api.openweathermap.org', '\
/data/2.5/weather', {
    'lat': '$lat',
    'lon': '$lon',
    'appid': apiKey,
    });

    var request = await http.get(httpsUri);
    if (request.statusCode == 200) {
    String data = request.body.toString();
    var description = jsonDecode(data)['weather'][0]['des\
cription'];

```

```
    return description;
    } else {
    return '${request.statusCode}';
    }
}

Future getWeatherWithGeolocator() async {
    try {
    Position position = await Geolocator.getCurrentPositi\
on(
        desiredAccuracy: LocationAccuracy.lowest);
    double lat = position.latitude;
    double lon = position.longitude;
    var data = getWeatherData(lat, lon, 'Secret');
    return data;
    } catch (e) {
    e;
    }
}

Future getWeatherData(double lat, double lon, String apiK\
ey) async {
    final httpsUri = Uri.http('api.openweathermap.org', '\
/data/2.5/weather', {
    'lat': '$lat',
    'lon': '$lon',
    'appid': apiKey,
    });

    var request = await http.get(httpsUri);
    if (request.statusCode == 200) {
    String data = request.body;
    var decodedData = jsonDecode(data);
    return decodedData;
    } else {
```

```
    '${request.statusCode}';
    }
}
}
```

As we see, in one method we have passed the latitude, longitude and API key with the "http" plugin.

Then finally we have called that method in another method that uses live data using the Geolocator plugin.

Consequently, we have got the city name and description in two separate methods.

However, in one method, we have got the whole JSON Map data.

```
Future getWeatherWithGeolocator() async {
    try {
    Position position = await Geolocator.getCurrentPositi\
on(
        desiredAccuracy: LocationAccuracy.lowest);
    double lat = position.latitude;
    double lon = position.longitude;
    var data = getWeatherData(lat, lon, 'Secret');
    return data;
    } catch (e) {
    e;
    }
}

Future getWeatherData(double lat, double lon, String apiK\
ey) async {
    final httpsUri = Uri.http('api.openweathermap.org', '\
/data/2.5/weather', {
    'lat': '$lat',
    'lon': '$lon',
    'appid': apiKey,
    });

    var request = await http.get(httpsUri);
    if (request.statusCode == 200) {
    String data = request.body;
    var decodedData = jsonDecode(data);
    return decodedData;
    } else {
    '${request.statusCode}';
    }
}
```

Since in this Future method we have not passed any Type, it automatically assumes that the Type would be dynamic.

In fact, we will get the data in JSON Map format.

So far, we have learned that JSON is a structured data that consists a Map. In addition, that Map has many Map or List inside.

Therefore, we can get the value by accessing key as follows.

```
var city = jsonDecode(data)['name'];
```

But in some cases, it could be complex as follows.

```
var description = jsonDecode(data)['weather'][0]['descrip\
tion'];
```

Display data with Future in Flutter

A Future is a promise token that we pass it to the user and says, please wait,you will get the data in Future.

We get the Future data either with the "then" method, or with "async, await".

Let us take a look at the code below that uses both to display current weather data based on user's location.

```
import 'package:flutter/material.dart';

import '../model/location.dart';

/// this is sixth step and last one on Future
///
class MyAppHome extends StatefulWidget {
const MyAppHome({Key? key, required this.title}) : super(\
key: key);

final String title;

@override
State<MyAppHome> createState() => _MyAppHomeState();
}

class _MyAppHomeState extends State<MyAppHome> {
UserLocation location = UserLocation();
var city = '';
var description = '';

@override
void initState() {
    super.initState();
    var description;
    location.getWeatherWithGeolocator().then((value) {
    description = value;
    print(description);
    });
}
```

```

@override
Widget build(BuildContext context) {
    return Scaffold(
    appBar: AppBar(
        title: Text(widget.title),
    ),
    body: Center(
        child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
            Text(
            city.toString(),
            style: Theme.of(context).textTheme.headline6,
            ),
            Text(
            description.toString(),
            style: Theme.of(context).textTheme.headline4,
            ),
            const SizedBox(
            height: 20.0,
            ),
            TextButton(
            onPressed: () async {
                city = await location.getCitynameWithGeol\
ocator();
                setState(() {
                city = city;
                });
                description =
                    await location.getWeatherDescriptionW\
ithGeolocator();
                setState(() {
                description = description;
                });
```

```
              },
              child: const Text(
                  'Get City and Current Weather',
                  style: TextStyle(
                  color: Colors.white,
                  fontSize: 30.0,
                  fontWeight: FontWeight.bold,
                  ),
              ),
              style: ButtonStyle(
                  backgroundColor: MaterialStateProperty.al\
l(Colors.red),
                  shape: MaterialStateProperty.all(
                  RoundedRectangleBorder(
                      borderRadius: BorderRadius.circular(3\
0.0),
                  ),
                  ),
              ),
              )
          ],
          ),
      ),
      // This trailing comma makes auto-formatting nicer fo\
r build methods.
      );
}
}
```

In the above code, we have first get the whole weather data in JSON Map format through init() method.

In the second step, we discussed the State object.

Therefore, once we run the app, we get the weather data in JSON Map format on our console.

```
{coord: {lon: xx.3163, lat: xx.6066}, weather: [{id: 721,\
 main: Haze, description: haze, icon: 50d}], base: statio\
ns,
main: {temp: 301.14, feels_like: 305.69, temp_min: 301.14\
, temp_max: 301.14, pressure: 1008, humidity: 83}, visibi\
lity:
2800, wind: {speed: 4.12, deg: 180}, clouds: {all: 75}, d\
t: 1648860766, sys: {type: 1, id: 9114, country: XX, sunr\
ise:
1648857516, sunset: 1648902114}, timezone: 19800, id: 134\
8747, name: xxxx, cod: 200}
```

As we see in the above code, the weather key has a value of List that again consists of a Map. In that map, we have a description keyword that has a value "haze".

What does that mean?

The Geolocator and the http plugins have fetched the current weather description of the place where the user is now at present.

Therefore, we can get that description key as follows.

```
var description = jsonDecode(data)['weather'][0]['descrip\
tion'];
```

And finally, we have displayed the description in our Text Widget.

```
Text(
            description.toString(),
            style: Theme.of(context).textTheme.headline4,
),
```

What we have requested, we have got in clear text. The open weather map website responded in JSON format which is also clear text.

For that reason, the Future didn't take much time and returned data in a fast manner.

It doesn't happen all the time.

Why?

Because sometimes the API may have to handle a large file, or open an image which is big.

Anyway, we have progressed a lot.

Now we can design our app, take input from the user and show the weather data.

The code repositories for this branch[23]

Pass data to State Flutter: Final Weather App

How do we pass data to State object in Flutter? In other words, how we can pass data to a Stateful Widget? Is it same as we do in the Stateless Widget?

The answer is no. Not exactly the same way we do that.

But in a similar vein, we pass data through class constructor first. And, after that, we use the "widget" property of the State class to access that data.

Firstly, in this section we will learn how we can accomplish that task. Secondly, at the same way, we will also finish our weather App "WithHer" that we have been building for some time.

Therefore, the first thing first.

Let us create a model class location. And keep that file in our model folder.

[23]https://github.com/sanjibsinha/with_her/tree/sixth-step-future-in-flutter

```
import 'dart:convert';
import 'package:geolocator/geolocator.dart';
import 'package:http/http.dart' as http;

/// this is seventh step
///

class UserLocation {
Future<dynamic> getWeatherWithGeolocator() async {
    try {
    Position position = await Geolocator.getCurrentPositi\
on(
        desiredAccuracy: LocationAccuracy.lowest);
    double lat = position.latitude;
    double lon = position.longitude;
    var data = getWeatherData(lat, lon, 'Secret Key');
    return data;
    } catch (e) {
    e;
    }
}

Future<dynamic> getWeatherData(double lat, double lon, St\
ring apiKey) async {
    final httpsUri = Uri.http('api.openweathermap.org', '\
/data/2.5/weather', {
    'lat': '$lat',
    'lon': '$lon',
    'appid': apiKey,
    });

    var request = await http.get(httpsUri);
    if (request.statusCode == 200) {
    String data = request.body;
    var decodedData = jsonDecode(data);
    return decodedData;
    } else {
    '${request.statusCode}';
    }
}
}
```

We have used three packages. The Dart convert library will help us to convert the JSON data to a Future dynamic type.

As a result, later we can use this location object to get the current weather data in our App through the API of the open weather map website.

Passing data from a Stateful Widget

First, we need a Stateful Widget to load the current weather Data from the API. Right?

Next, we will pass that dynamic Future data to a loading screen where we will display the current weather of the location where the user is staying at present.

For example, in this weekend, I had come to the Howrah City. Therefore, my weather app should show that result.

However, since I have kept my location accuracy at the lowest point, it might vary. Consequently, the App may display some other place around the Howrah city.

```
desiredAccuracy: LocationAccuracy.lowest);
```

Anyway, let us see the first page where we have created a location object.

```
import 'package:flutter/material.dart';

import '../model/location.dart';
import 'weather_page.dart';

/// this is sixth step and last one on Future
///
class MyAppHome extends StatefulWidget {
const MyAppHome({Key? key, required this.title}) : super(\
key: key);

final String title;

@override
State<MyAppHome> createState() => _MyAppHomeState();
}

class _MyAppHomeState extends State<MyAppHome> {
UserLocation location = UserLocation();

@override
void initState() {
    super.initState();
    updateWeather();
}

void updateWeather() {
    var weather;
    location.getWeatherWithGeolocator().then((value) {
    weather = value;
    Navigator.push(context, MaterialPageRoute(builder: (c\
ontext) {
        return WeatherPage(
        weather: weather,
        title: 'Get City and Weather',
            );
    }));
    });
}

@override
Widget build(BuildContext context) {
    return Scaffold(
    backgroundColor: Colors.grey.shade600,
    );
}
}
```

Very simple logic that we have followed. Firstly, we have created a location object.

```
UserLocation location = UserLocation();
```

Secondly, we have got the necessary weather data as a dynamic Future object. After that we have sent that weather data to the weather page where we will display the result.

```
void updateWeather() {
    var weather;
    location.getWeatherWithGeolocator().then((value) {
    weather = value;
    Navigator.push(context, MaterialPageRoute(builder: (c\
ontext) {
        return WeatherPage(
        weather: weather,
        title: 'Get City and Weather',
        );
    }));
    });
}
```

Finally, we have called the above method in our "initState()" method.

```
@override
void initState() {
    super.initState();
    updateWeather();
}
```

As the first screen loads it asks for the user permission first.

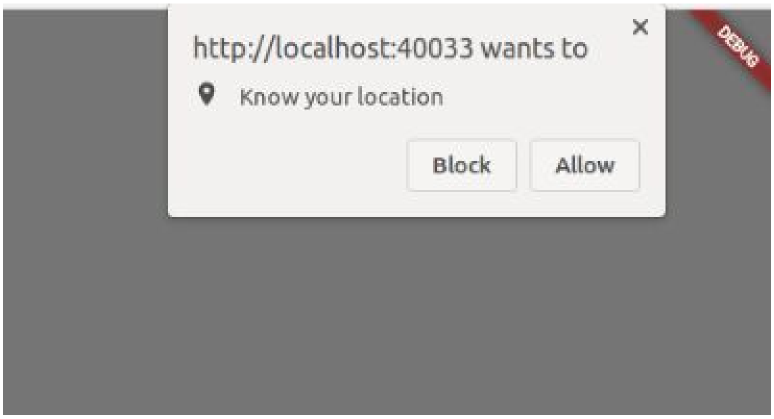


Figure 7.1 – Assking for User's permision to display location on User's screen

To get the current weather, we need to give the permission by clicking the allow button.

As a result, it will pass the weather data as a dynamic Future object.

Pass data to a State object in Flutter

Getting data in a Stateless Widget does not make things tricky. Rather it's simple.

We have received the data in final variables and display them. Right?

But in a Stateful Widget, we need to use the State "widget" property.

Certainly, we first receive that data in final variables just as we do in a Stateless Widget.

But after that, we need to access that data through the "widget" property.

Let us see the code. After that, we will discuss how it works.

```
import 'package:flutter/material.dart';

class WeatherPage extends StatefulWidget {
const WeatherPage({Key? key, required this.weather, requi\
red this.title})
    : super(key: key);

final dynamic weather;
final String title;

@override
State<WeatherPage> createState() => _WeatherPageState();
}

class _WeatherPageState extends State<WeatherPage> {
@override
Widget build(BuildContext context) {
    return Scaffold(
    appBar: AppBar(
        title: Text(widget.title),
    ),
    body: Center(
        child: Container(
        decoration: const BoxDecoration(
            image: DecorationImage(
            image: NetworkImage(
                'https://cdn.pixabay.com/photo/2022/02/19\
/22/48/forest-7023487_960_720.jpg'),
            fit: BoxFit.cover,
            ),
        ),
        constraints: const BoxConstraints.expand(),
        child: SafeArea(
            child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: <Widget>[
                Text(
                'Welcome to ${widget.weather['name']}',
                style: Theme.of(context).textTheme.headli\
ne3,
                ),
                Text(
                'Current temperature: ${(widget.weather['\
main']['temp'] - 273.15).toStringAsFixed(2)} Celsius',    \

                style: Theme.of(context).textTheme.headli\
ne6,
                ),
                Text(
                'Current condition: ${widget.weather['wea\
ther'][0]['description']}',
                style: Theme.of(context).textTheme.headli\
ne4,
                ),
                const SizedBox(
                height: 20.0,
                ),
            ],
            ),
        ),
          // This trailing comma makes auto-formatting nice\
r for build methods.
          ),
      ),
      );
}
}
```

As we can see we have received two data from the parent widget. One is the "weather" and the other is the "title". As a result, we have displayed the title as follows.

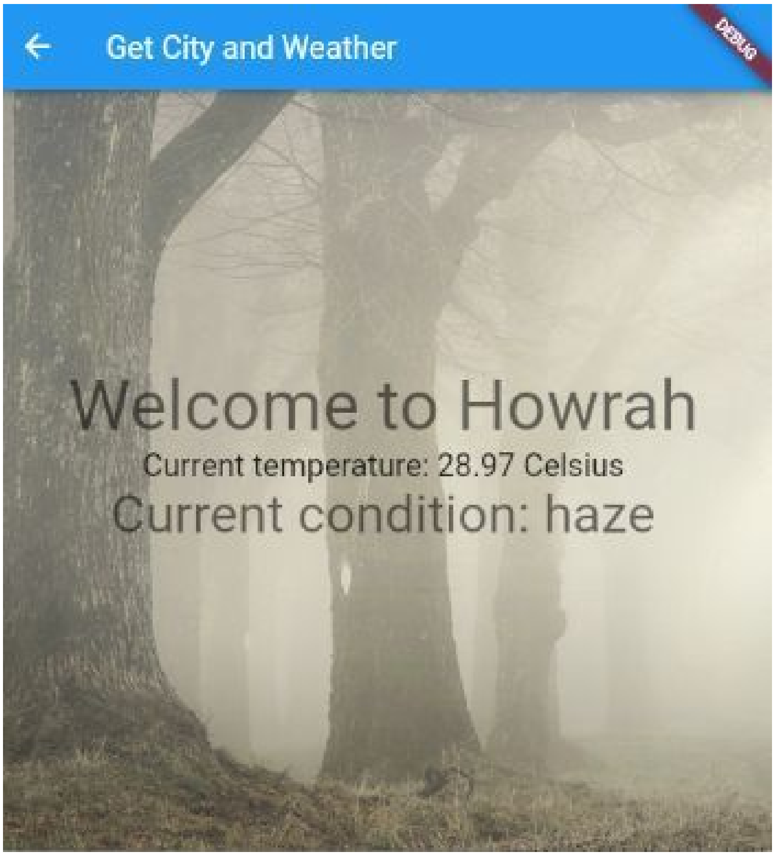
```
appBar: AppBar(
          title: Text(widget.title),
      ),
```

Here the "widget" property of the State class can access the properties of the Stateful Widget in this way.

For example, we have displayed the city name as follows.

```
Text(
                    'Welcome to ${widget.weather['name']}',
                    style: Theme.of(context).textTheme.headli\
ne3,
      ),
```

Now we can display the current weather data of the user.



Welcome to Howrah
Current temperature: 28.97 Celsius
Current condition: haze

Figure 7.2 – Current Weather displayed on User's screen

Remember, in our previous steps we have shown that the weather data is actually a JSON Map as follows.

```
{coord: {lon: 88.3163, lat: 22.6066}, weather: [{id: 721,\
 main: Haze, description: haze, icon: 50d}], base: statio\
ns,
main: {temp: 301.14, feels_like: 305.69, temp_min: 301.14\
, temp_max: 301.14, pressure: 1008, humidity: 83}, visibi\
lity:
2800, wind: {speed: 4.12, deg: 180}, clouds: {all: 75}, d\
t: 1648860766, sys: {type: 1, id: 9114, country: IN, sunr\
ise:
1648857516, sunset: 1648902114}, timezone: 19800, id: 134\
8747, name: Howrah, cod: 200}
```

Here the "name" acts as a key. So we have accessed the value of the city name by using the key.

If you want to clone the whole project and test in your machine, please clone the GitHub repository.

But you need to create and pass your own API key as app ID.

The code repositories for the final branch[24]

[24]https://github.com/sanjibsinha/with_her

8. How we can build a Blog App and learn Flutter backend programming using SQLite

Before we start building the a Blog App, and learn CRUD in Flutter we must know what are new features in Flutter 2.8.

Why?

Because in this version of Flutter Future, await, async and backend database play an important role.

Flutter 2.8, Future, await, async and Database

Flutter 2.8 has added a major performance booster to our Flutter Applications.

What is new in Flutter 2.8

Using Flutter's single codebase we can create applications for Android, iOS, Windows, Linux, Web and many more. Flutter 2.8 has added many interesting features that makes this development much easier. In addition it's added a major performance booster.

If you've been reading my blog, you may have noticed that I have written about the earlier Flutter revisions and releases.

However, Flutter 2.8 has overtaken 2.5 and improving the performance of Flutter applications on mobile devices.

We can simply upgrade to the new version by issuing this command on our terminal.

```
flutter upgrade
```

It takes some time to upgrade depending on the speed of your internet.

Then you can check the Flutter and Dart version.

```
Flutter 2.8.0 • channel stable • https://github.com/flutt\
er/flutter.git
Framework • revision cf44000065 (3 days ago) • 2021-12-08\
 14:06:50 -0800
Engine • revision 40a99c5951
Tools • Dart 2.15.0
....
Dart SDK version: 2.15.0 (stable) (Fri Dec 3 14:23:23 202\
1 +0100) on "linux_x64"
```

As a result, our Flutter applications open faster and consume less memory.

As some of Google's core apps like Google Play and Stadia uses Flutter, it's quite expected that Google will keep improving Flutter core.

Flutter 2.8 ha made it much easier to connect to Firebase. The good news is Firebase plugins for Flutter have been upgraded from "Beta" to "Stable."

Now, sign-in has also become easier with a Widget.

We’ve just seen that with the revision and new release of Flutter 2.8, Dart programming language SDK has also been updated to 2.15.

We’ll discuss in a separate post how we can utilise Dart’s new features. Some features will definitely boost the development of Flutter User Interfaces.

Dart 2.15 also brings concurrency improvements, enhanced enumerations, and optimisations that provide a 10 percent reduction in memory utilisation.

However, I personally like the web view part.

The webview_flutter plugin has been upgraded to 3.0 and that means provides preliminary support for a new platform: the web.

To use it, we can add the following line to your pubspec.yaml:

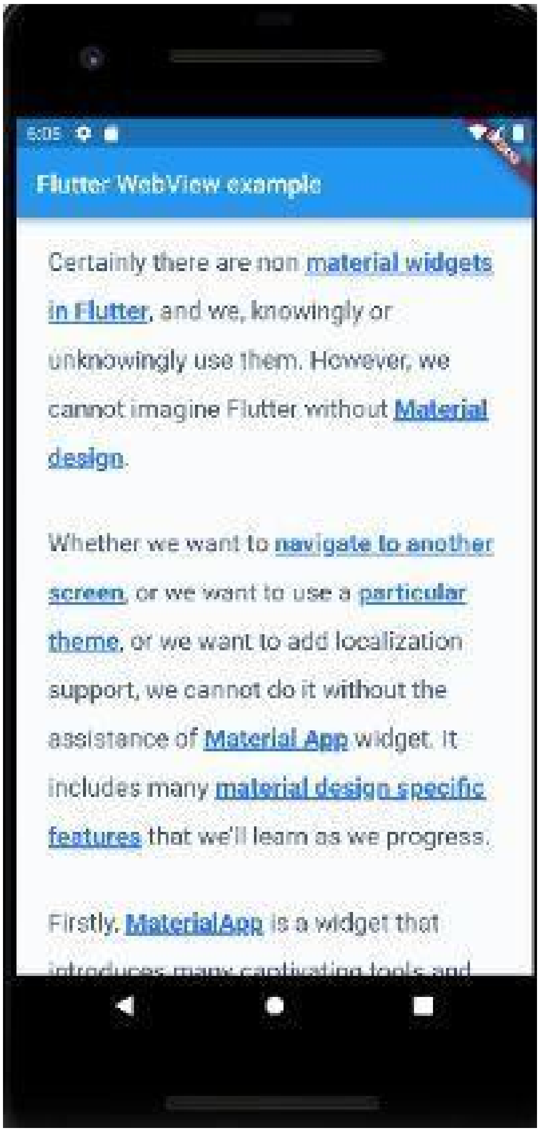
```
dependencies:
webview_flutter: ^3.0.0
webview_flutter_web: ^0.1.0
```

To sum up, Flutter 2.8 promises faster startup and lower resource requirements for mobile apps. We can also easily connect with the back-end services.

Let us try webview_flutter plugin in a simple Flutter 2.8 application.

```
import 'package:flutter/material.dart';
import 'package:webview_flutter/webview_flutter.dart';

void main() {
runApp(const MyApp());
}

class MyApp extends StatelessWidget {
const MyApp({Key? key}) : super(key: key);

// This widget is the root of your application.
@override
Widget build(BuildContext context) {
    return MaterialApp(
    title: 'Flutter Demo',
    theme: ThemeData(
        primarySwatch: Colors.blue,
    ),
    home: const MyHomePage(title: 'Flutter Demo Home Page\
'),
    );
}
}

class MyHomePage extends StatefulWidget {
const MyHomePage({Key? key, required this.title}) : super\
(key: key);

final String title;

@override
State<MyHomePage> createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
@override
Widget build(BuildContext context) => Scaffold(
        appBar: AppBar(title: const Text('Flutter WebView\
 example')),
        body: const WebView(initialUrl: 'https://sanjibsi\
nha.com'),
    );
}
```

Just run the code and you'll find that web view has brought the web pages as we want.



Certainly there are non material widgets in Flutter, and we, knowingly or unknowingly use them. However, we cannot imagine Flutter without Material design.

Whether we want to navigate to another screen, or we want to use a particular theme, or we want to add localization support, we cannot do it without the assistance of Material App widget. It includes many material design specific features that we'll learn as we progress.

Firstly, MaterialApp is a widget that introduces many captivating tools and

Figure 8.1 – Flutter 2.8 and web view 3.0 plugin

We can always check out the new webview codelab, which might give us a proper guide to host web content in our Flutter app.

Future, await and async

Future in Flutter or Dart gives us a promise token and returns a value at in future.

Flutter is mainly single thread. Why so? Because Dart language is a single threaded language. However, Flutter uses several threads to do its work.

Does it sound confusing?

Don’t worry. Basically, in Flutter all of our Dart code runs on User Interface thread. In spite of that, it uses other threads besides that. Just to name a few, there are Platform thread, I/O thread, etc.

On the contrary Android applications are not single threaded. Besides the main thread Android applications can run the heavy jobs in worker or background thread.

What do we mean by heavy jobs?

Well, let me explain. In any mobile application, users need to accomplish many tasks simultaneously. Consider an event like button click. That might do some small logical operations, such as get the result of a small addition. The main thread can handle such small operations and doesn’t take much time.

However, an image to be opened using a network, or downloading a file isn’t a small operation. As a result, the main thread needs to do the heavy lifting. And, that might halt the whole program.

However, Dart and Flutter have its answer. They together perform long-running operations with the help of Future API, async, await keywords, and then functions.

Together they perform asynchronous programming in Flutter.

Asynchronous doesn't mean multi-threaded. Basically it means the code shouldn't run at the same time and the job of main thread shouldn't be burdensome.

In case of Flutter, asynchronous means that an operation is scheduled to be run on the same thread after other tasks have finished.

With reference to asynchronous programming we need to understand isolate also. In a separate section, we'll discuss isolate.

When the conception of isolate comes into the picture, Dart no longer remains single threaded language. Because it can create separate isolate. Although within an isolate Dart again runs on a single thread.

Future in Flutter or Dart gives us a promise token and says that a value will be returned at some point in future.

It never says in which thread it will have its job done at that certain point.

This part is little tricky. However, we need to see how we can write a simple Dart code to understand this concept.

```
import 'dart:async';

void main(List<String> args) {
print('main thread starts >>>>>');
print('main process starts >>>>>');

openImage();

print('main process ends and starts counting seconds... >\
>>>>');
}

void openImage() async {
var imageFile = await downloadImage();
print('The downloaded file is --> $imageFile');
print('main thread ends after 10 seconds....');
}

Future<String> downloadImage() {
var image = Future<String>.delayed(Duration(seconds: 10),\
 () {
    return 'Here is an imaginative image downloaded.';
});
return image;
}
```

Let's try to understand the above code.

Depending on that, as a consequence, we can later create a database operations in Flutter. Remember, database queries takes time and we must do using Future.

Future works on <T>, or type. In our case, we've used String type and returns a text after 10 seconds.

However, when we run the program, the default process starts the main thread. And it prints, main thread starts and main process starts.

Then after 10 seconds we get the result that our Future object returns. And at the same time, the main thread closes down.

Let's run and watch the output, so that everything makes sense.

```
main thread starts >>>>>
main process starts >>>>>
main process ends and starts counting seconds... >>>>>
The downloaded file is --> Here is an imaginative image d\
ownloaded.
main thread ends after 10 seconds....
Exited
```

To sum up, here a Future object has produced a value of type

String. A Future object belongs to any one of the states – either it's completed, or it's uncompleted.

When the main thread starts and we call a Future, it queues up work and returns an uncompleted state.

But we don't see that part in our bare eyes.

When the Future's operation is finished, it returns a completed state with a value of same type.

If it cannot, Future completes with an error.

Which database we use in Flutter

SQLite database with future, await, and async create, retrieve, update, or delete data in Flutter.

We can use SQLite database in Flutter. However, we need to use a special package or plugin sqflite which is available in pub.dev. We also need to use Future API, async, await keywords, and then functions to make it successful.

We've discussed this feature for absolute beginners in previous section, is Flutter single thread?

Anyway, as a result, the sqflite package provides classes and functions to interact with a SQLite database. Moreover, using SQLite database is better than using a local file, or key-value store.

There are reasons to do that.

SQLite database provides faster CRUD. That is, we can create, retrieve, update and delete data. And, it's always better than the local persistent solutions.

In this section we'll see how we can create a users table in our SQLite database, and retrieve that data on our Flutter application.

It will be a gentle introduction to SQLite database with Flutter. We'll see how we can create a database, insert some data and after that, retrieve them.

Later as we progress, we'll see how we can improve the other functionalities. To get the code snippets used in this section, we have a respective GitHub repository. If you have interest, please download and run the code.

Besides sqflite package, we need to use another package path, that will define the location for storing the database on the disk.

For the beginners, here is a guide what SQLite database is.

What is SQLite database?

SQLite is a C-language library that implements many features at one go. It is small, fast, self-contained, high-reliability, full-featured, SQL database engine.

By the way, SQLite is the most used database engine in the world. Besides, SQLite database file format is stable, cross-platform, and backwards compatible.

There are over 1 trillion SQLite databases in active use at present.

Therefore, let's go ahead and make our first Flutter Application with SQLite database.

How do you make a database on Flutter app?

Let's add the dependencies in our pubspec.yaml file first.

```
dependencies:
flutter:
    sdk: flutter
sqflite:
path:
```

After that, create a model folder, inside lib folder, and create one user class and a database helper class there.

Firstly, let's take a look at the user model class.

```
class User {
final int? id;
final String name;
final String location;

User({
    this.id,
    required this.name,
    required this.location,
});

User.fromMap(Map<String, dynamic> res)
    : id = res["id"],
        name = res["name"],
        location = res["location"];

Map<String, Object?> toMap() {
    return {
    'id': id,
    'name': name,
    'location': location,
    };
}
}
```

To get data stored in SQLite database, we need to convert them to a map. The reason is simple. After all, in our Flutter Application we need to convert them to a list of items.

That’s why we have created a named constructor User.fromMap() and a method toMap().

Secondly, we’ll create a table with the help of the helper class.

```
import 'package:sqflite/sqflite.dart';
import 'package:path/path.dart';

import 'user.dart';

class DatabaseHandler {
Future<Database> initializeDB() async {
    String path = await getDatabasesPath();
    return openDatabase(
    join(path, 'usersfirst.db'),
    onCreate: (database, version) async {
        await database.execute(
        "CREATE TABLE usersfirst(id INTEGER PRIMARY KEY A\
UTOINCREMENT, name TEXT NOT NULL, location TEXT NOT NULL)\
",
        );
    },
    version: 1,
    );
}

Future<int> insertUser(List<User> users) async {
    int result = 0;
    final Database db = await initializeDB();
    for (var user in users) {
    result = await db.insert('usersfirst', user.toMap());
    }
    return result;
}

Future<List<User>> retrieveUsers() async {
    final Database db = await initializeDB();
    final List<Map<String, Object?>> queryResult = await \
db.query('usersfirst');
    return queryResult.map((e) => User.fromMap(e)).toList\
();
}
}
```

The method getDatabasePath() of sqflite package will get the default database location. However, the join() method is inside the package path that will join the given path into a single path.

As a matter of fact, two packages sqflite and path are necessary for this reason.

In addition, to keep our first Flutter SQLite database application simple, we're going to create, insert, and retrieve the users. We'll add the list of users manually in our main method.

Now, we're going to display the users data.

```
import 'package:flutter/material.dart';

import 'model/database_handler.dart';
import 'model/user.dart';

void main() {
runApp(const MyApp());
}

/// adding first branch
class MyApp extends StatelessWidget {
const MyApp({Key? key}) : super(key: key);

```

```
// This widget is the root of your application.
@override
Widget build(BuildContext context) {
    return MaterialApp(
    title: 'Flutter smimple database',
    theme: ThemeData(
        primarySwatch: Colors.blue,
    ),
    home: const MyHomePage(title: 'Flutter smimple databa\
se'),
    );
}
}

class MyHomePage extends StatefulWidget {
const MyHomePage({Key? key, required this.title}) : super\
(key: key);

final String title;

@override
State<MyHomePage> createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
late DatabaseHandler handler;

@override
void initState() {
    Future<int> addUsers() async {
    User firstUser = User(name: "Mana", location: "Nabagr\
am");
    User secondUser = User(name: "Babu", location: "Nabag\
ram");
    User thirdUser = User(name: "Pata", location: "Nabagr\
am");
    List<User> listOfUsers = [
        firstUser,
        secondUser,
        thirdUser,
    ];
    return await handler.insertUser(listOfUsers);
    }

    super.initState();
    handler = DatabaseHandler();
    handler.initializeDB().whenComplete(() async {
    await addUsers();
    setState(() {});
    });
}

@override
Widget build(BuildContext context) {
    return Scaffold(
    appBar: AppBar(
        title: Text(widget.title),
    ),
    body: FutureBuilder(
        future: handler.retrieveUsers(),
        builder: (BuildContext context, AsyncSnapshot<Lis\
t<User>> snapshot) {
        if (snapshot.hasData) {
            return ListView.builder(
            itemCount: snapshot.data?.length,
            itemBuilder: (BuildContext context, int index\
) {
                return Card(
                child: ListTile(
                    key: ValueKey<int>(snapshot.data![ind\
ex].id!),
                    contentPadding: const EdgeInsets.all(\
8.0),
                    title: Text(
                    snapshot.data![index].name,
                    style: Theme.of(context).textTheme.he\
adline3,
                    ),
                    subtitle: Text(
                    snapshot.data![index].location.toStri\
ng(),
                    style: Theme.of(context).textTheme.he\
adline5,
                    ),
                ),
                );
            },
            );
        } else {
            return const Center(child: CircularProgressIn\
dicator());
        }
        },
   ),
   );
}
}
```

Next, we create an instance of class DatabaseHandler first. With the help of the database handler object we can call initalizeDb() method to create the SQLite database.

We know that Future in Flutter or Dart gives us a promise token and says that a value will be returned at some point in future. Therefore, when Future is completed, addUsers() method is called.

Consequently, the addUsers() method calls insertUsers() method to insert the list of users to the SQLite database.

Next, the FutureBuilder widget builds itself based on the latest snapshot of interaction with a Future. Moreover, unless the Future is completed, it gives us the uncompleted state which shows a circular progress indicator.

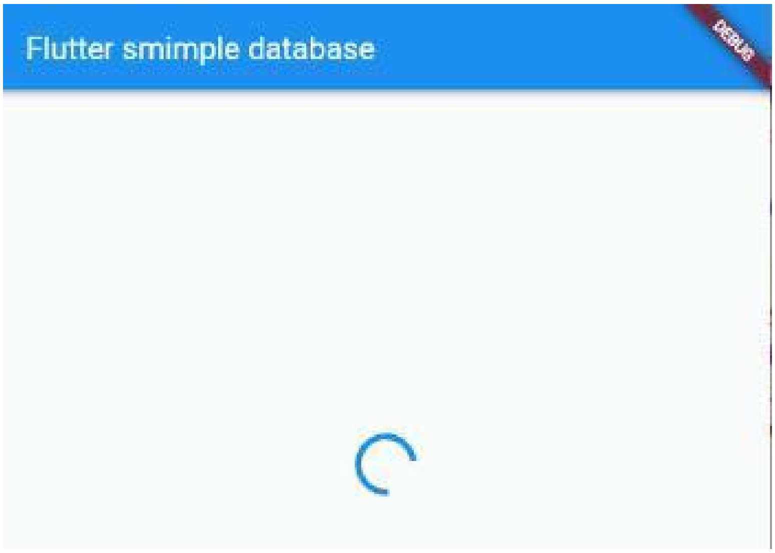
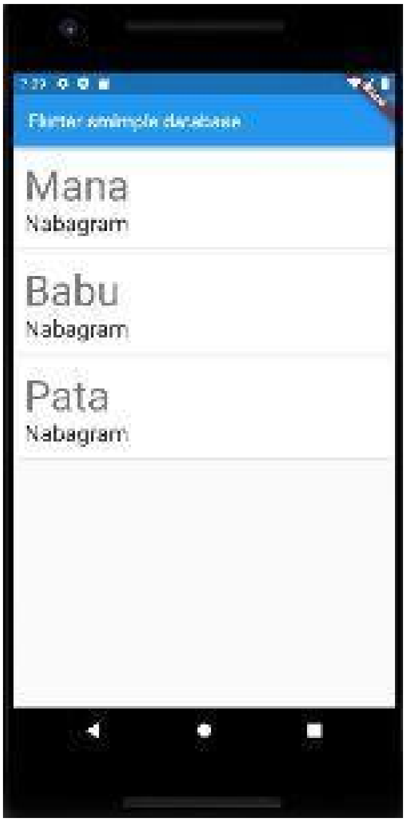


Figure 8.2 – Flutter application tries to retrieve data from SQLite database with FutureBuilder

However, when the value is returned in Future, it retrieves data from SQLite database successfully.


Flutter simple database

Mana
Nabagram

Babu
Nabagram

Pata
Nabagram

Figure 8.3 – FutureBuilder retrieves data from SQLite database in Flutter

By the way, we’ve added this list of users manually.

```
Future<int> addUsers() async {
    User firstUser = User(name: "Mana", location: "Nabagr\
am");
    User secondUser = User(name: "Babu", location: "Nabag\
ram");
    User thirdUser = User(name: "Pata", location: "Nabagr\
am");
    List<User> listOfUsers = [
        firstUser,
        secondUser,
        thirdUser,
    ];
    return await handler.insertUser(listOfUsers);
    }
```

In the next section, we'll try to implement other features of CRUD in our Flutter Application.

With the help of SQLite database in accordance with future, await, and async we can insert, retrieve, update, or delete data in Flutter.

SQLite Database and Flutter

In this section, we'll take a look at how we can insert data to SQL database and display them. As we progress, we'll learn the other techniques to update and delete data.

It is preferable to use SQLite database in Flutter, because it is faster than local file. However, we need to use a special package or plugin sqflite which is available in pub.dev. We also need to use Future API, async, await keywords, and then functions to make it successful.

We've discussed Future, await and async for absolute beginners in previous section, is Flutter single thread?

Anyway, as a result, the sqflite package provides classes and functions to interact with a SQLite database.

What is SQLite database?

SQLite is a C-language library that implements many features at one go. It is small, fast, self-contained, high-reliability, full-featured, SQL database engine.

By the way, SQLite is the most used database engine in the world. Besides, SQLite database file format is stable, cross-platform, and backwards compatible.

There are over 1 trillion SQLite databases in active use at present.

Therefore, let's go ahead and make our first Flutter Application with SQLite database.

How to insert data in SQL database in Flutter?

Firstly, we need a Text Controller to type on the screen. Right?

Then, we need a Text Button to press, so that that piece of data will be inserted into the SQLite database.

Secondly, we need to add the dependency.

```
dependencies:
cupertino_icons: ^1.0.2
flutter:
    sdk: flutter
intl: ^0.17.0
path_provider: ^2.0.8
provider: ^6.0.1
sqflite:
```

Next, we need a model data class and Database Handler helper class that will connect our SQLite database to the model data class.

First, data model class.

```
class User {
final int? id;
final String name;

User({
    this.id,
    required this.name,
});

User.fromMap(Map<String, dynamic> res)
    : id = res["id"],
        name = res["name"];

Map<String, Object?> toMap() {
    return {
    'id': id,
    'name': name,
    };
}
}
```

We need to map that data model class so that later we can work on it in Flutter.

To get data stored in SQLite database, we need to convert them to a map. The reason is simple. After all, in our Flutter Application we need to convert them to a list of items.

That's why we have created a named constructor User.fromMap() and a method toMap().

Secondly, we'll create a table with the help of the helper class.

Why?

Because, the database helper class will provide the methods that will create the database table, and help us to insert and retrieve data from SQLite database.

```
import 'package:sqflite/sqflite.dart';
import 'package:path/path.dart';

import 'user.dart';

class DatabaseHandler {
Future<Database> initializeDB() async {
    String path = await getDatabasesPath();
    return openDatabase(
    join(path, 'usersix.db'),
    onCreate: (database, version) async {
        await database.execute(
        "CREATE TABLE usersix(id INTEGER PRIMARY KEY AUTO\
INCREMENT, name TEXT NOT NULL)",
        );
    },
    version: 1,
    );
}

Future<int> insertUser(List<User> users) async {
    int result = 0;
    final Database db = await initializeDB();
    for (var user in users) {
    result = await db.insert('usersix', user.toMap());
    }
    return result;
}

Future<List<User>> retrieveUsers() async {
    final Database db = await initializeDB();
    final List<Map<String, Object?>> queryResult = await \
db.query('usersix');
    return queryResult.map((e) => User.fromMap(e)).toList\
();
}
}
```

It’s a good practice that we break down these code snippets and keep this data model and helper class in our model folder inside lib folder.

The method getDatabasePath() of sqflite package will get the default database location. However, the join() method is inside the package path that will join the given path into a single path.

As a matter of fact, two packages sqflite and path are necessary for this reason.

In addition, to keep our first Flutter SQLite database application simple, we’re going to create, insert, and retrieve the users. We’ll add the list of users manually in our main method.

Our next challenge is to show a text field to the user so that she can type any text and press the button.

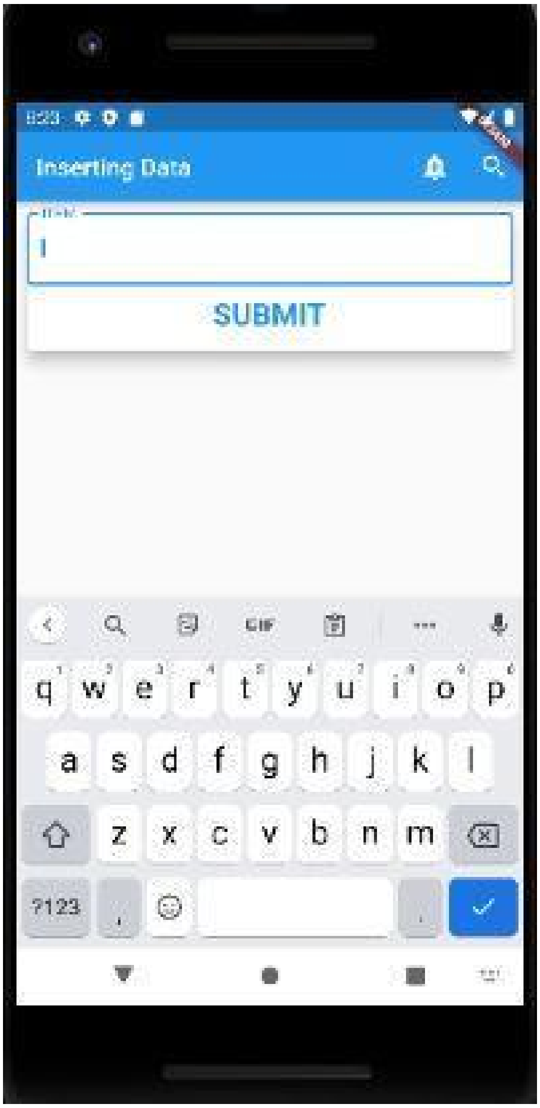


Figure 8.4 – Inserting data in SQLite database in Flutter

At the same page we need to show the Navigate button, that will take us to another screen where the inserted data will be displayed.

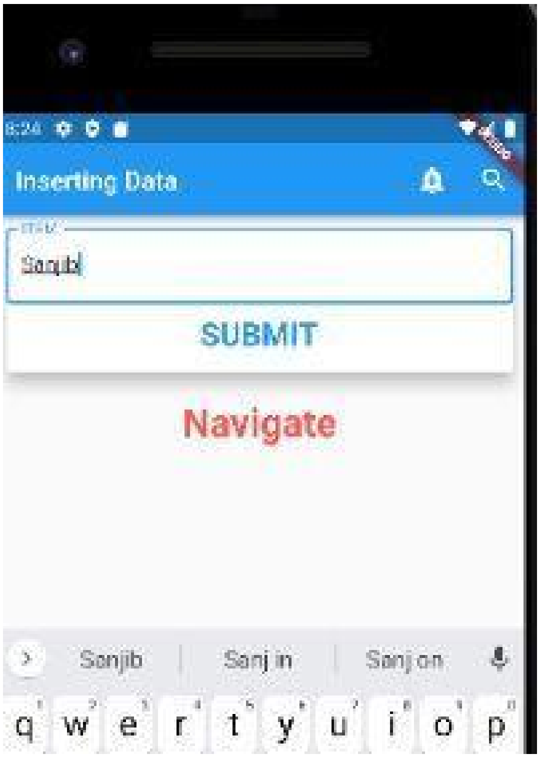


Figure 8.5 – Data is being inserted in SQLite database in Flutter

```
import 'package:flutter/material.dart';
import 'package:flutter_data_and_backend/view/future_dark\
.dart';

import 'model/user.dart';

void main() {
runApp(const MyApp());
}

/// we're now in branch six
///
```

```
class MyApp extends StatelessWidget {
const MyApp({Key? key}) : super(key: key);

@override
Widget build(BuildContext context) {
    return const MaterialApp(
    title: 'data',
    home: MyAppHome(),
    );
}
}

class MyAppHome extends StatefulWidget {
const MyAppHome({Key? key}) : super(key: key);

@override
State<MyAppHome> createState() => _MyAppHomeState();
}

class _MyAppHomeState extends State<MyAppHome> {
final List<User> usersList = [];

final nameController = TextEditingController();

void addName(String name) {
    final user = User(
    name: name,
    );
    setState(() {
    usersList.add(user);
    });
}

@override
Widget build(BuildContext context) {
    return Scaffold(
    appBar: AppBar(
        title: const Text('Inserting Data'),
        actions: <Widget>[
        IconButton(
            icon: const Icon(Icons.add_alert),
            tooltip: 'Show Snackbar',
            onPressed: () {
            ScaffoldMessenger.of(context).showSnackBar(
                const SnackBar(
                content: Text('A SnackBar'),
                ),
            );
            },
        ),
        IconButton(
            icon: const Icon(Icons.search_outlined),
            tooltip: 'Search',
            onPressed: () {
            // our code
            },
        ),
        ],
    ),
    body: Center(
        child: Column(
        children: [
            Container(
            padding: const EdgeInsets.all(5),
            child: Card(
                elevation: 10,
                child: Column(
                children: [
                    TextField(
                    decoration: const InputDecoration(
                        border: OutlineInputBorder(),
                        labelText: 'ITEM',
                        suffixStyle: TextStyle(
                        fontSize: 50,
                        fontWeight: FontWeight.bold,
                        ),
                    ),
                    controller: nameController,
                    ),
                    TextButton(
                    onPressed: () {
                        addName(
                        nameController.text,
                        );
                    },
                    child: const Text(
                        'SUBMIT',
                        style: TextStyle(
                        fontSize: 25,
                        fontWeight: FontWeight.bold,
                        ),
                    ),
                    ),
                    const SizedBox(
                    height: 5,
                    ),
                ],
                ),
            ),
            ),
            NavigationWidget(usersList: usersList),
        ],
        ),
    ),
    );
```

```
}
}

class NavigationWidget extends StatelessWidget {
const NavigationWidget({
    Key? key,
    required this.usersList,
}) : super(key: key);

final List<User> usersList;

@override
Widget build(BuildContext context) {
    return Center(
    child: Container(
        padding: const EdgeInsets.all(5),
        height: 150,
        width: 350,
        child: Column(
        children: usersList.map((e) {
            return Column(
            children: [
                TextButton(
                onPressed: () {
                    Navigator.push(
                    context,
                    MaterialPageRoute(
                        builder: (context) => FutureDark(
                        name: e.name,
                        ),
                    ),
                    );
                },
                child: const Text(
                    'Navigate',
                    style: TextStyle(
                    fontSize: 30.0,
                    fontWeight: FontWeight.bold,
                    color: Colors.redAccent,
                    ),
                ),
                ),
            ],
            );
        }).toList(),
        ),
    ),
    );
}
}
```

The above code could be broken down to more pages or screens using more custom widgets.

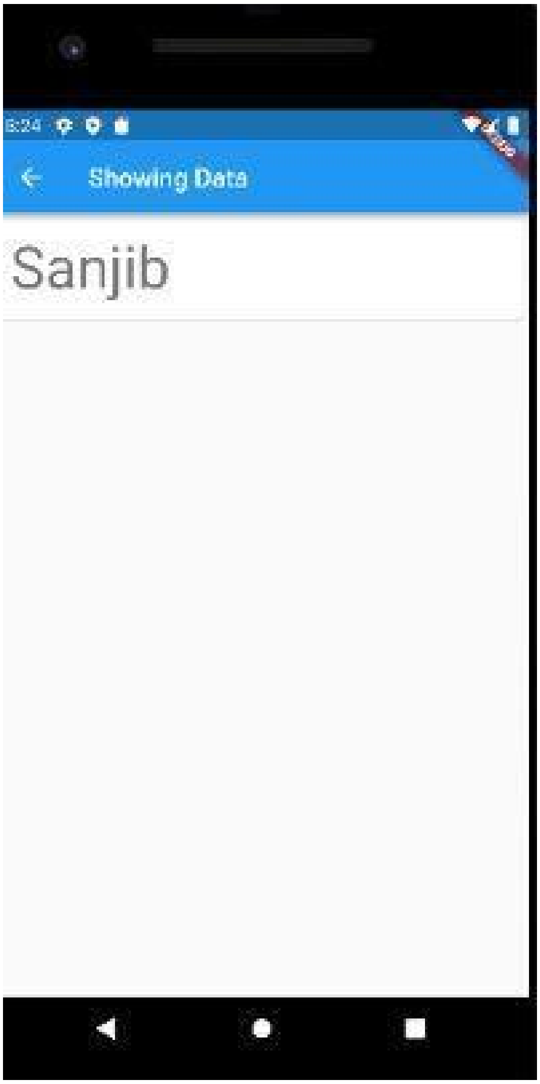


Figure 8.6 – Data from SQLite database is being shown on Flutter screen

However, in one place will help us to understand the mechanism of how we have used Text Controller, Text Button and a Material Page Route to insert data and after that, we can see them.

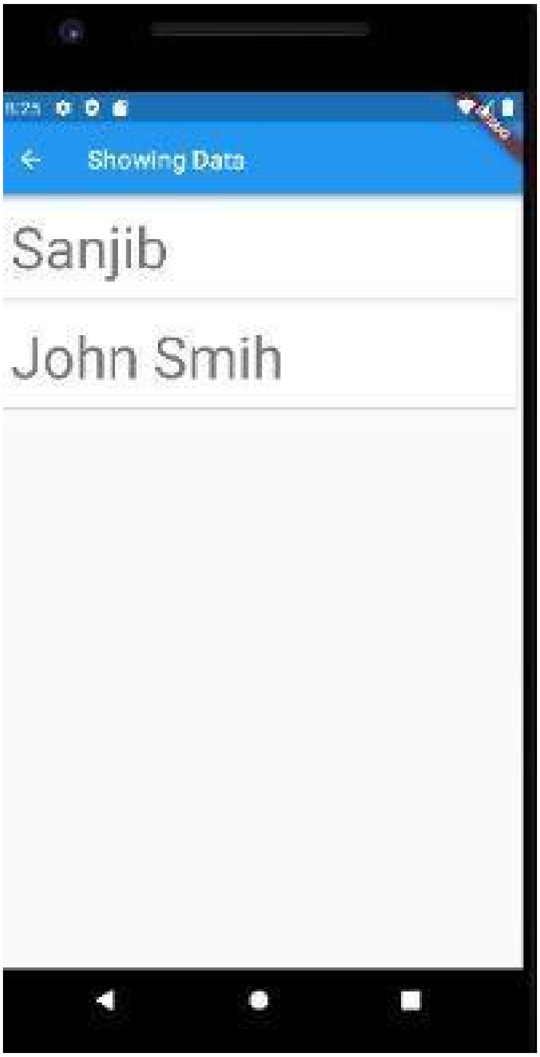


Figure 8.7 – Second Data from SQLite database is being shown on Flutter screen

Second Data from SQLite database is being shown on Flutter screen How to retrieve data from SQLite database in Flutter?

Retrieving data from SQLite database is much easier than inserting data.

With the help of Future API, async, await keywords, and then functions and FutureBuilder widget, we can do that.

At the same screen we also catch the data or list item that we've sent from the home page.

```
import 'package:flutter/material.dart';

import 'package:flutter_data_and_backend/model/database_h\
andler.dart';
import 'package:flutter_data_and_backend/model/user.dart';

class FutureDark extends StatefulWidget {
const FutureDark({
    Key? key,
    required this.name,
}) : super(key: key);

final String name;

@override
State<FutureDark> createState() => _FutureDarkState();
}

class _FutureDarkState extends State<FutureDark> {
DatabaseHandler? handler;
@override
void initState() {
    List<User> users = [
    User(name: widget.name.toString()),
    ];
    Future<int> addUsers() async {
    return await handler!.insertUser(users);
    }

    super.initState();
    handler = DatabaseHandler();
    handler!.initializeDB().whenComplete(() async {
    await addUsers();
    setState(() {});
    });
}

@override
Widget build(BuildContext context) {
    return Scaffold(
    appBar: AppBar(
        title: const Text('Showing Data'),
    ),
    body: FutureBuilder(
        future: handler!.retrieveUsers(),
        builder: (BuildContext context, AsyncSnapshot<Lis\
t<User>> snapshot) {
        if (snapshot.hasData) {
            return ListView.builder(
            itemCount: snapshot.data?.length,
            itemBuilder: (BuildContext context, int index\
) {
                return Card(
                child: ListTile(
                    key: ValueKey<int>(snapshot.data![ind\
ex].id!),
                    contentPadding: const EdgeInsets.all(\
8.0),
                    title: Text(
                    snapshot.data![index].name,
                    style: Theme.of(context).textTheme.he\
adline3,
                    ),
                ),
                );
```

```
              },
              );
          } else {
              return const Center(child: CircularProgressIn\
dicator());
          }
          },
      ),
      );
}
}
```

The Future Builder is a widget that builds itself based on the latest snapshot of interaction with a Future.

We've obtained the future must have been obtained earlier, during State.initState, State.didUpdateWidget, or State.didChangeDependencies.

FutureBuilder must not be created during the State.build or StatelessWidget.build method call when constructing the FutureBuilder.

Create, Retrieve, Update and Delete with SQLite Database

In the previous chapter, we have had a gentle introduction on SQLite database and Flutter. We have learned how to create a SQLite database with the help of "sqflite" package.

In addition we've also seen how to use "path" package to define a local path to create the database.

In the following two sections, we'll learn how we can plan, create and modify a SQLite database in Flutter, so that we can successfully do the CRUD, or Create, Retrieve, Update and Delete data and build a Blog Application using that knowledge.

And in the last section we'll convert the existing Blog Application and refactor it to a "My Diary" Aplication.

SQLite Blog in Flutter: First Part

This is the first part of building a Blog application with SQLite database in Flutter.

We're going to learn how we can build a SQLite Blog Application in Flutter. As it sounds, a Blog App must fulfill some criteria.

What are they?

The SQLite database should allow us to insert, update or delete data. Moreover, we need to retrieve data as well.

For better understanding, let's break this flutter tutorial in a few parts.

So, in this first part will teach us to learn a couple of things.

Firstly, we'll use a Flutter package or plugin, sqflite which is available in pub.dev.

Secondly, we also need to use Future API, async, await keywords, and then functions to make it successful.

We've discussed Future, await and async for absolute beginners in previous section, is Flutter single thread? Therefore, if you're a beginner, you might take a look before we start.

Which database is best for Flutter?

The sqflite package provides classes and functions to interact with a SQLite database.

And, finally, we need a Text Controller to type on the screen. Right?

We also need a Text Button or Elevated Button to press, so that that piece of data will be inserted into the SQLite database.

To begin with, we need to add the dependencies in pubspec.yaml.

```
dependencies:
cupertino_icons: ^1.0.2
flutter:
    sdk: flutter
flutter_staggered_grid_view: ^0.4.1
intl: ^0.17.0
path:
provider: ^6.0.1
sqflite:
```

After that, we need to use await and async in our entry point.

```
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'view/all_pages.dart';

Future main() async {
WidgetsFlutterBinding.ensureInitialized();
await SystemChrome.setPreferredOrientations([
    DeviceOrientation.portraitUp,
    DeviceOrientation.portraitDown,
]);

runApp(const MyApp());
}

class MyApp extends StatelessWidget {
static const String title = 'Blogs';

const MyApp({Key? key}) : super(key: key);

@override
Widget build(BuildContext context) => MaterialApp(
        debugShowCheckedModeBanner: false,
        title: title,
        themeMode: ThemeMode.light,
        theme: ThemeData(
        primaryColor: Colors.pink.shade200,
        scaffoldBackgroundColor: Colors.pink.shade600,
        appBarTheme: const AppBarTheme(
            backgroundColor: Colors.transparent,
            elevation: 0,
        ),
        ),
        home: const AllPages(),
    );
}
```

In our Material App widget, we've defined the global theme. In addition, we pass that theme to our Home page AllPages.

Next, we need to define couple of things in this page, AllPages. If there is no data in our SQLite database, then it will show a page like below.

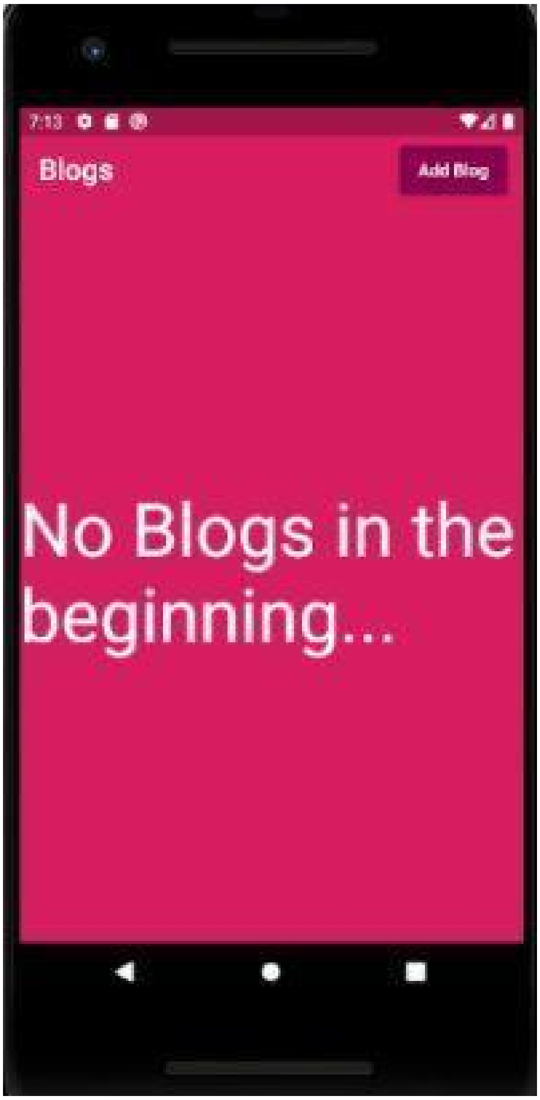


Figure 8.8 – Home page of the Blog App in Flutter

As a result, we can start adding blog items to this Flutter Application. However, in our Home page, AllPages stateful Widget, we must define that.

Let's take a look at the AllPages code snippet.

```
import 'package:flutter/material.dart';
import 'package:flutter_staggered_grid_view/flutter_stagg\
ered_grid_view.dart';
import '/model/blogs.dart';
import '/model/blog.dart';
import 'edit.dart';
import 'detail.dart';
import '/controller/blog_card.dart';

class AllPages extends StatefulWidget {
const AllPages({Key? key}) : super(key: key);

@override
_AllPagesState createState() => _AllPagesState();
}

class _AllPagesState extends State<AllPages> {
late List<Blog> blogs;
bool isLoading = false;

@override
void initState() {
    super.initState();

    refreshingAllBogs();
}

@override
void dispose() {
    BlogDatabaseHandler.instance.close();

    super.dispose();
}

```

```
Future refreshingAllBogs() async {
    setState(() => isLoading = true);

    blogs = await BlogDatabaseHandler.instance.readAllBlo\
gs();

    setState(() => isLoading = false);
}

@override
Widget build(BuildContext context) => Scaffold(
        appBar: AppBar(
        title: const Text(
            'Blogs',
            style: TextStyle(fontSize: 24),
        ),
        actions: [
            Padding(
            padding: const EdgeInsets.symmetric(vertical:\
 8, horizontal: 12),
            child: ElevatedButton(
                style: ElevatedButton.styleFrom(
                onPrimary: Colors.white,
                primary: Colors.pink.shade900,
                ),
                onPressed: () async {
                await Navigator.of(context).push(
                    MaterialPageRoute(builder: (context) \
=> const EditPage()),
                );

                refreshingAllBogs();
                },
                child: const Text('Add Blog'),
            ),
            )
        ],
        ),
        body: Center(
        child: isLoading
            ? const CircularProgressIndicator()
            : blogs.isEmpty
                ? const Text(
                    'No Blogs in the beginning...',
                    style: TextStyle(color: Colors.white,\
 fontSize: 60),
                    )
                : buildingAllBlogs(),
        ),
    );

Widget buildingAllBlogs() => StaggeredGridView.countBuild\
er(
        padding: const EdgeInsets.all(8),
        itemCount: blogs.length,
        staggeredTileBuilder: (index) => const StaggeredT\
ile.fit(2),
        crossAxisCount: 4,
        mainAxisSpacing: 4,
        crossAxisSpacing: 4,
        itemBuilder: (context, index) {
        final blog = blogs[index];

        return GestureDetector(
            onTap: () async {
            await Navigator.of(context).push(MaterialPage\
Route(
                builder: (context) => DetailPage(blogId: \
blog.id!),
            ));
```

```

              refreshingAllBogs();
              },
              child: BlogCard(blog: blog, index: index),
          );
          },
      );
}
```

The above Widget plays a very important role in our Flutter SQLite Blog Application.

Firstly, it would let us allow to add a new blog. To do that, this widget takes us to a different Widget, EditPage.

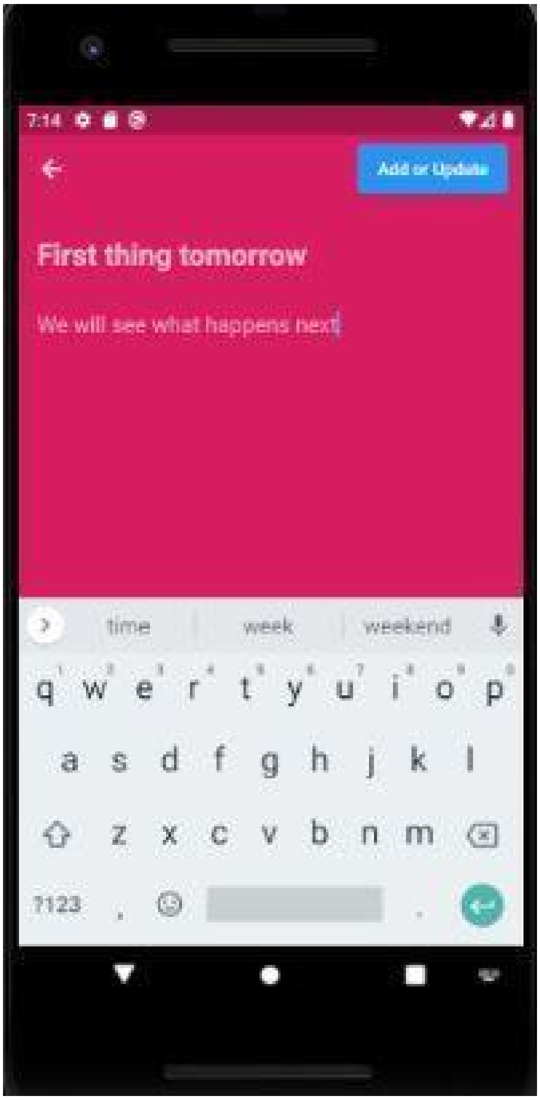


Figure 8.9 – Inserting data into Blog App in Flutter

After that, once we are done in the EditPage, that again sends us back to this Widget, AllPages. And, here, we start seeing all the blogs.

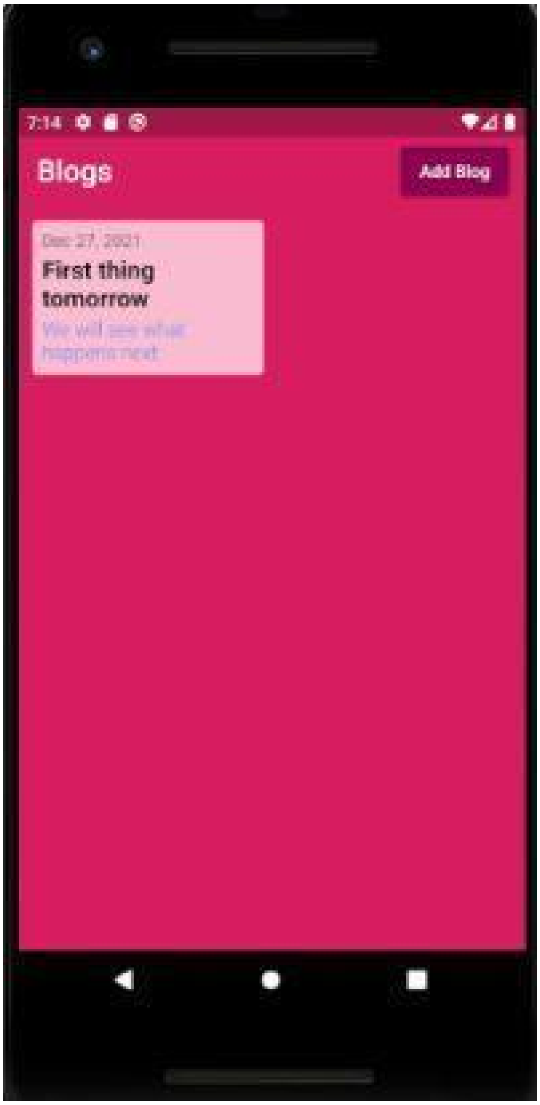


Figure 8.10 – Retrieving data in Blog App in Flutter

Retrieving data from SQLite database is much easier than inserting data. However, we'll learn everything from scratch here while building this Blog app in Flutter.

Meanwhile, with the help of Future API, async, await keywords, and then functions and FutureBuilder widget, we can do that.

At the same screen we also catch the data or list item that we've sent from the home page.

In the above code, we've found a couple of interesting thing.

Firstly, the home page imports couple of other pages and two model classes. The model classes provide the data models.

Secondly, the model class also serves as database handler utilities.

And, finally, we needed the edit page and the detail page to serve other purposes, such as updating and deleting items.

At the end, after finishing all tasks, we come back to the home page again.

This is the flow of logic that runs this SQLite Blog application in Flutter.

In the next section, we'll take a look at how we can build the data model and database utilities with the help of sqflite package.

By that time, if you have interest, please visit the respective GitHub repository.

- All related Code Snippet in this GitHub repository[25]

SQLite Blog, Flutter: Second Part

This is the second part of building a Blog application with SQLite database in Flutter.

We've already started building the SQLite Blog application in Flutter. The previous section has discussed the application structure.

[25]https://github.com/sanjibsinha/flutter_data_and_backend/tree/test-blog

In this second part, we'll concentrate on database connection, data model classes.

As we proceed, we'll also learn how we can develop the same application thematically. Therefore, we'll add more functionalities. We'll keep in mind that our application should look great and should be user friendly.

First thing first. Let's recapitulate a few things about SQLite database and Flutter.

Firstly, we'll use a Flutter package or plugin, sqflite which is available in pub.dev.

Secondly, we also need to use Future API, async, await keywords, and then functions to make it successful.

We've discussed Future, await and async for absolute beginners in previous section, is Flutter single thread? Therefore, if you're a beginner, you might take a look before we start.

Which DB is best for Flutter?

SQLite database is one of the best DB for Flutter. Although there are couple more. Like Hive, Firebase, etc.

However we're using SQLite database for this Blog application in Flutter, because it's local, fast and easy to maintain.

First of all, we need a Blog data model.

Let's keep that class in model folder.

```
const String tableOfBlogs = 'Blogs';

class BlogFields {
static final List<String> values = [
    /// Adding all fields
    id, title, description, time
];

static const String id = '_id';
static const String title = 'title';
static const String description = 'description';
static const String time = 'time';
}

class Blog {
final int? id;
final String title;
final String description;
final DateTime createdTime;

const Blog({
    this.id,
    required this.title,
    required this.description,
    required this.createdTime,
});

Blog copy({
    int? id,
    String? title,
    String? description,
    DateTime? createdTime,
}) =>
    Blog(
        id: id ?? this.id,
        title: title ?? this.title,
        description: description ?? this.description,
        createdTime: createdTime ?? this.createdTime,
    );

static Blog fromJson(Map<String, Object?> json) => Blog(
        id: json[BlogFields.id] as int?,
        title: json[BlogFields.title] as String,
        description: json[BlogFields.description] as Stri\
ng,
        createdTime: DateTime.parse(json[BlogFields.time]\
 as String),
    );

Map<String, Object?> toJson() => {
        BlogFields.id: id,
        BlogFields.title: title,
        BlogFields.description: description,
        BlogFields.time: createdTime.toIso8601String(),
    };
}
```

We need to Map the data model object. It's because Flutter wants a list to display.

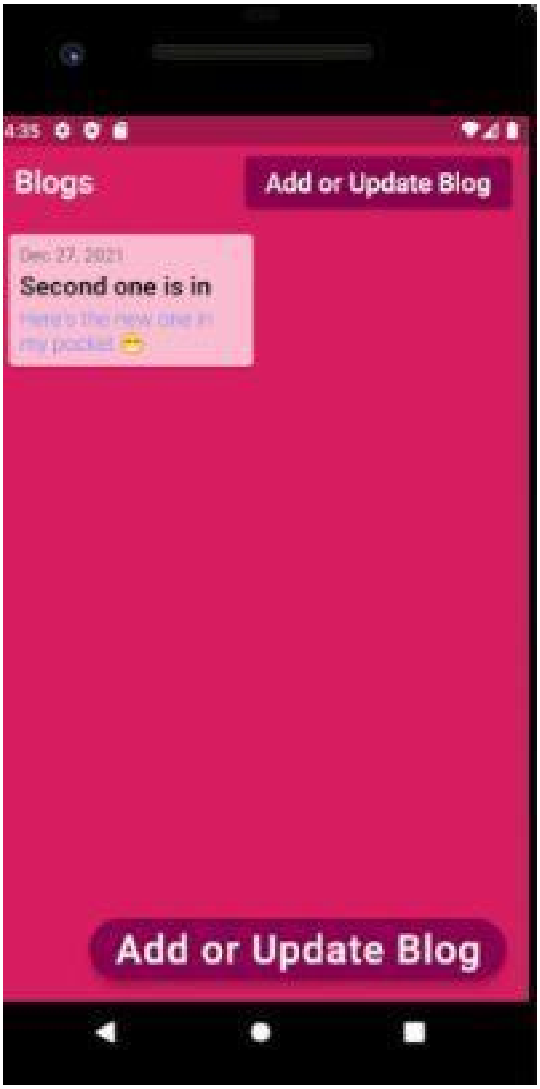


Figure 8.11 – Retrieving blog items on Home page

Second thing we need a database handler class that will create the table, and at the same time, it'll do the CRUD.

We've learned what CRUD is. Create, retrieve, update and delete.

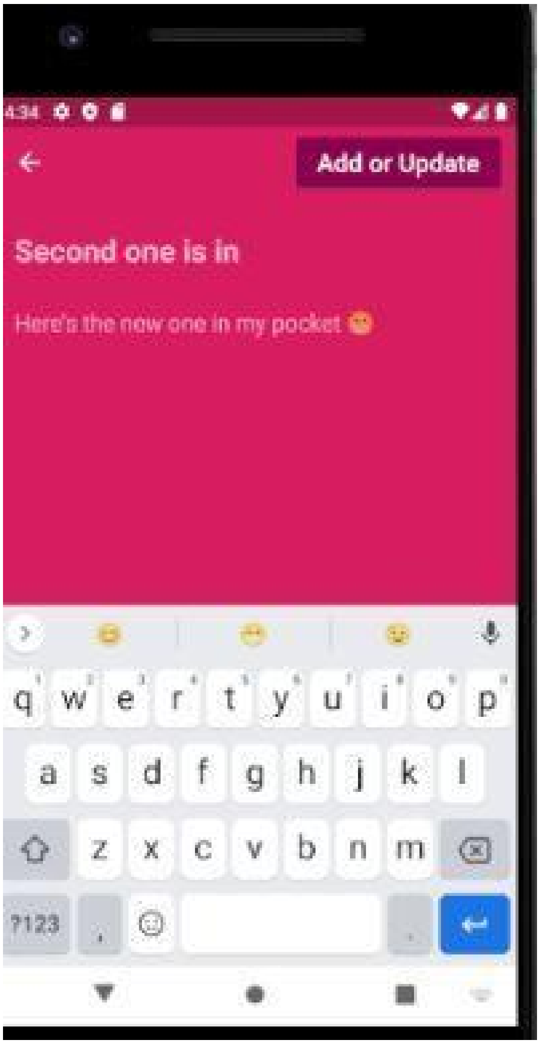


Figure 8.12 – Adding Blog items to SQLite database in Flutter

In this SQLite Blog application we'll also do the same. However, we've tweaked the previous home page code to accommodate a floating action button.

```
import 'package:path/path.dart';
import 'package:sqflite/sqflite.dart';
import '../model/blog.dart';

class BlogDatabaseHandler {
static final BlogDatabaseHandler instance = BlogDatabaseH\
andler._init();

static Database? _database;

BlogDatabaseHandler._init();

Future<Database> get database async {
    if (_database != null) return _database!;

    _database = await _initDB('newblogs.db');
    return _database!;
}

Future<Database> _initDB(String filePath) async {
    final dbPath = await getDatabasesPath();
    final path = join(dbPath, filePath);

    return await openDatabase(path, version: 1, onCreate:\
 _createDB);
}

Future _createDB(Database db, int version) async {
    const idType = 'INTEGER PRIMARY KEY AUTOINCREMENT';
    const textType = 'TEXT NOT NULL';

    await db.execute('''
CREATE TABLE $tableOfBlogs (
${BlogFields.id} $idType,
${BlogFields.title} $textType,
${BlogFields.description} $textType,
${BlogFields.time} $textType
)
''');
}

Future<Blog> create(Blog blog) async {
    final db = await instance.database;

    final id = await db.insert(tableOfBlogs, blog.toJson(\
));
    return blog.copy(id: id);
}

Future<Blog> readBlog(int id) async {
    final db = await instance.database;

    final maps = await db.query(
    tableOfBlogs,
    columns: BlogFields.values,
    where: '${BlogFields.id} = ?',
    whereArgs: [id],
    );

    if (maps.isNotEmpty) {
    return Blog.fromJson(maps.first);
    } else {
    throw Exception('ID $id not found');
    }
}

Future<List<Blog>> readAllBlogs() async {
    final db = await instance.database;

    const orderBy = '${BlogFields.time} ASC';

    final result = await db.query(tableOfBlogs, orderBy: \
orderBy);

    return result.map((json) => Blog.fromJson(json)).toLi\
st();
}

Future<int> update(Blog blog) async {
    final db = await instance.database;

    return db.update(
    tableOfBlogs,
    blog.toJson(),
    where: '${BlogFields.id} = ?',
    whereArgs: [blog.id],
    );
}

Future<int> delete(int id) async {
    final db = await instance.database;

    return await db.delete(
    tableOfBlogs,
    where: '${BlogFields.id} = ?',
    whereArgs: [id],
    );
}

Future close() async {
    final db = await instance.database;

    db.close();
}
}
```

In the above code, everything is very verbose.

Firstly, we've mentioned a path so the database gets created locally and stores data locally.

Secondly, we've created the database and table with fields. As we see, we've kept it simple.

Thirdly, we've created methods that will read, update, delete items.

Finally, we've closed the database.

At the same time, we've changed the code of our home page slightly. As a result, we can now add or update the blog items either from AppBar, or through the floating action button.

```
import 'package:flutter/material.dart';
import 'package:flutter_staggered_grid_view/flutter_stagg\
ered_grid_view.dart';
import '/model/blogs.dart';
import '/model/blog.dart';
import 'edit.dart';
import 'detail.dart';
import '/controller/blog_card.dart';

class AllPages extends StatefulWidget {
const AllPages({Key? key}) : super(key: key);

@override
_AllPagesState createState() => _AllPagesState();
}

class _AllPagesState extends State<AllPages> {
late List<Blog> blogs;
bool isLoading = false;

@override
void initState() {
    super.initState();

    refreshingAllBogs();
}

@override
void dispose() {
    BlogDatabaseHandler.instance.close();

    super.dispose();
}

Future refreshingAllBogs() async {
    setState(() => isLoading = true);

    blogs = await BlogDatabaseHandler.instance.readAllBlo\
gs();

    setState(() => isLoading = false);
}

@override
Widget build(BuildContext context) => Scaffold(
        appBar: AppBar(
        title: const Text(
            'Blogs',
            style: TextStyle(fontSize: 24),
        ),
        actions: [
            Padding(
            padding: const EdgeInsets.symmetric(vertical:\
 8, horizontal: 12),
            child: ElevatedButton(
                style: ElevatedButton.styleFrom(
                onPrimary: Colors.white,
                primary: Colors.pink.shade900,
                ),
                onPressed: () async {
                await Navigator.of(context).push(
                    MaterialPageRoute(builder: (context) \
=> const EditPage()),
                );

                refreshingAllBogs();
                },
                child: const Text(
                'Add or Update Blog',
                style: TextStyle(
                    fontSize: 20,
                ),
                ),
            ),
            )
        ],
        ),
        body: Center(
        child: isLoading
            ? const CircularProgressIndicator()
            : blogs.isEmpty
                ? const Text(
                    'No Blogs in the beginning...',
                    style: TextStyle(color: Colors.white,\
 fontSize: 60),
                    )
                : buildingAllBlogs(),
        ),
        floatingActionButton: FloatingActionButton.extend\
ed(
        tooltip: 'Add or Update Blog',
        foregroundColor: Colors.white,
        backgroundColor: Colors.pink.shade900,
        onPressed: () async {
            await Navigator.of(context).push(
            MaterialPageRoute(builder: (context) => const\
 EditPage()),
            );

            refreshingAllBogs();
        },
        label: const Text(
            'Add or Update Blog',
            style: TextStyle(
            fontSize: 30,
            ),
        ),
        ),
    );

Widget buildingAllBlogs() => StaggeredGridView.countBuild\
er(
        padding: const EdgeInsets.all(8),
        itemCount: blogs.length,
        staggeredTileBuilder: (index) => const StaggeredT\
ile.fit(2),
        crossAxisCount: 4,
        mainAxisSpacing: 4,
        crossAxisSpacing: 4,
        itemBuilder: (context, index) {
        final blog = blogs[index];

        return GestureDetector(
            onTap: () async {
            await Navigator.of(context).push(MaterialPage\
Route(
                builder: (context) => DetailPage(blogId: \
blog.id!),
            ));

            refreshingAllBogs();
            },
            child: BlogCard(blog: blog, index: index),
        );
        },
    );
}
```

As a result, when there is no blog items, the home page looks like the following one.

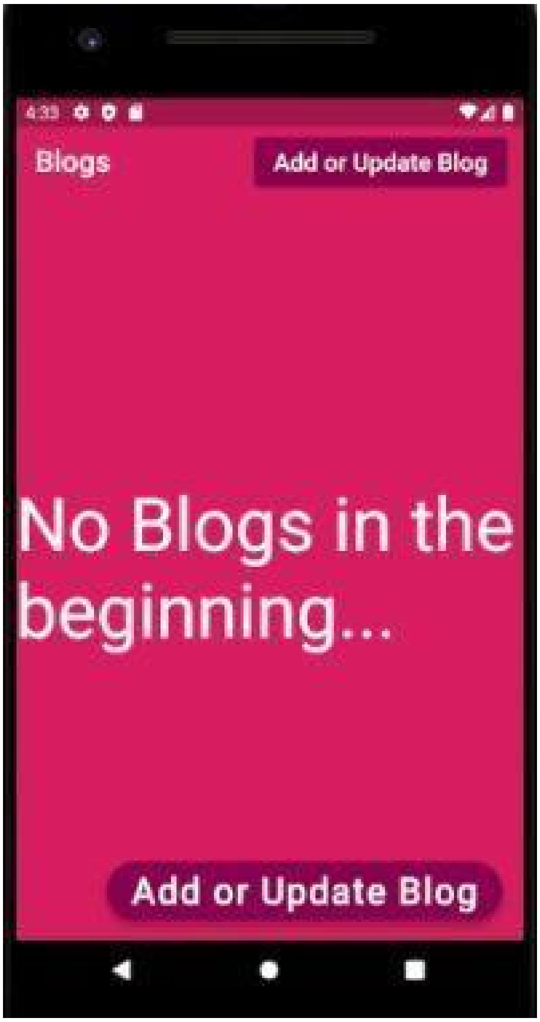


Figure 8.13 – Adding a Floating action button to SQLite Blog Home page in Flutter

In the next part, or section we'll take a look at the edit page, where actual action takes place.

After that, we'll also see how we can use the Card widget to display all inserted data.

As we're changing the previous code, we keep them in separate GitHub branch. For full code for this section, please visit the respective GitHub repository.

- All related Code Snippet in this GitHub repository[26]

SQLite Blog, Flutter: Final Part

This is the final part of our Blog application in Flutter where we use SQLite database and CRUD.

In this final part of building SQLite Blog application in Flutter we will accomplish the basic principle of CRUD. As a consequence, we'll create, retrieve, update and delete data in SQLite database.

We've already built a significant part of a SQLite Blog Application in Flutter. We might also see the progress of the initial phase in this section – SQLite Blog application in Flutter.

The previous section has discussed the application structure.

In the second part, we'd concentrated on database connection, data model classes.

In this final part we'll take a look at the logic flow and see how we can convert this Blog application to a My Diary application.

We've also changed the layout in a significant way.

Therefore, before we jump in, let's recapitulate a few things about SQLite database and Flutter.

Firstly, we'll use a Flutter package or plugin, sqflite which is available in pub.dev.

Secondly, we also need to use Future API, async, await keywords, and then functions to make it successful.

[26]https://github.com/sanjibsinha/flutter_data_and_backend/tree/one-blog

Finally, we've discussed Future, await and async for absolute beginners in previous section, is Flutter single thread?

Therefore, if you're a beginner, you might take a look before we proceed towards the final section.

What packages we need for SQLite database in Flutter?

We'll start with the pubspec.yaml file, where we must add all the dependencies.

```
dependencies:
cupertino_icons: ^1.0.2
flutter:
    sdk: flutter
flutter_staggered_grid_view: ^0.4.1
intl: ^0.17.0
path:
provider: ^6.0.1
sqflite:
```

To give this SQLite Blog Application in Flutter a final touch, we need some packages or plugins.

The "sqflite", "path", "intl", and "flutter_staggered_grid_view" packages will help us in many ways.

The "sqflite", and "path" packages work at tandem. They help each other as we'll find later. With the help of "path" package we define the path of the local database.

We need the packages "intl", and "flutter_staggered_grid_view" for different purposes.

The package "intl" helps us to format the date in our Blog. And, the "flutter_staggered_grid_view" package helps us in building the layout while we display the blog or diary's contents.

Does flutter need a backend?

Yes, Flutter needs a backend. Moreover, we can choose between two. Either we can go with a server, or we store our data locally with SQLite database.

In our case, we have decided to use SQLite database.

Consequently, we need a service or utility class that will create a database in local path first. Next, it will help us to use the SQL language to create a table with required fields.

```
import 'package:path/path.dart';
import 'package:sqflite/sqflite.dart';
import '../model/blog.dart';

class BlogDatabaseHandler {
static final BlogDatabaseHandler instance = BlogDatabaseH\
andler._init();

static Database? _database;

BlogDatabaseHandler._init();

Future<Database> get database async {
    if (_database != null) return _database!;

    _database = await _initDB('newblogs.db');
    return _database!;
}

Future<Database> _initDB(String filePath) async {
    final dbPath = await getDatabasesPath();
    final path = join(dbPath, filePath);

    return await openDatabase(path, version: 1, onCreate:\
 _createDB);
```

```
}

Future _createDB(Database db, int version) async {
    const idType = 'INTEGER PRIMARY KEY AUTOINCREMENT';
    const textType = 'TEXT NOT NULL';

    await db.execute('''
CREATE TABLE $tableOfBlogs (
${BlogFields.id} $idType,
${BlogFields.title} $textType,
${BlogFields.description} $textType,
${BlogFields.time} $textType
)
''');
}

Future<Blog> create(Blog blog) async {
    final db = await instance.database;

    final id = await db.insert(tableOfBlogs, blog.toJson(\
));
    return blog.copy(id: id);
}

Future<Blog> readBlog(int id) async {
    final db = await instance.database;

    final maps = await db.query(
    tableOfBlogs,
    columns: BlogFields.values,
    where: '${BlogFields.id} = ?',
    whereArgs: [id],
    );

    if (maps.isNotEmpty) {
```

```
    return Blog.fromJson(maps.first);
    } else {
    throw Exception('ID $id not found');
    }
}

Future<List<Blog>> readAllBlogs() async {
    final db = await instance.database;

    const orderBy = '${BlogFields.time} ASC';

    final result = await db.query(tableOfBlogs, orderBy: \
orderBy);

    return result.map((json) => Blog.fromJson(json)).toLi\
st();
}

Future<int> update(Blog blog) async {
    final db = await instance.database;

    return db.update(
    tableOfBlogs,
    blog.toJson(),
    where: '${BlogFields.id} = ?',
    whereArgs: [blog.id],
    );
}

Future<int> delete(int id) async {
    final db = await instance.database;

    return await db.delete(
    tableOfBlogs,
    where: '${BlogFields.id} = ?',
    whereArgs: [id],
    );
}

Future close() async {
    final db = await instance.database;

    db.close();
}
}
```

Firstly, we've mentioned a path so the database gets created locally and stores data locally.

Secondly, we've created the database and table with fields. As we see, we've kept it simple.

Thirdly, we've created methods that will read, update, delete items.

Finally, we've closed the database.

At the same time, we've changed the code of our home page slightly. As a result, we can now add or update the blog items either from AppBar, or through the floating action button.

Subsequently, to help the utility class we need a data model class.

```
const String tableOfBlogs = 'Blogs';

class BlogFields {
static final List<String> values = [
    /// Adding all fields
    id, title, description, time
];

static const String id = '_id';
static const String title = 'title';
static const String description = 'description';
static const String time = 'time';
}

class Blog {
final int? id;
final String title;
final String description;
final DateTime createdTime;

const Blog({
    this.id,
    required this.title,
    required this.description,
    required this.createdTime,
});

Blog copy({
    int? id,
    String? title,
    String? description,
    DateTime? createdTime,
}) =>
    Blog(
        id: id ?? this.id,
        title: title ?? this.title,
        description: description ?? this.description,
        createdTime: createdTime ?? this.createdTime,
    );

static Blog fromJson(Map<String, Object?> json) => Blog(
        id: json[BlogFields.id] as int?,
        title: json[BlogFields.title] as String,
        description: json[BlogFields.description] as Stri\
ng,
        createdTime: DateTime.parse(json[BlogFields.time]\
 as String),
   );

Map<String, Object?> toJson() => {
        BlogFields.id: id,
        BlogFields.title: title,
        BlogFields.description: description,
        BlogFields.time: createdTime.toIso8601String(),
   };
}
```

We've kept these files in our "model" sub-folder.

After that, we have built three pages and to keep them we have created a "view" sub-folder.

Which database is used for flutter?

As we've been discussing the topic we find, the Flutter team also recommends to use SQLite database. They say, "Flutter apps can make use of the SQLite databases via the sqflite plugin available on pub."

Why?

The reason is simple. And it's explained below.

If our app needs to persist and query large amounts of data on the local device, it's always better to use a database instead of a local file or key-value store.

In general, databases provide faster inserts, updates, and queries compared to other local persistence solutions, and SQLite is the best choice.

Let's proceed with our code.

First, we have a home page, that will handle the layout and backend at the same time.

If there is no entry it shows us a blank page and we start adding items.

If not, it shows like the following screenshot.

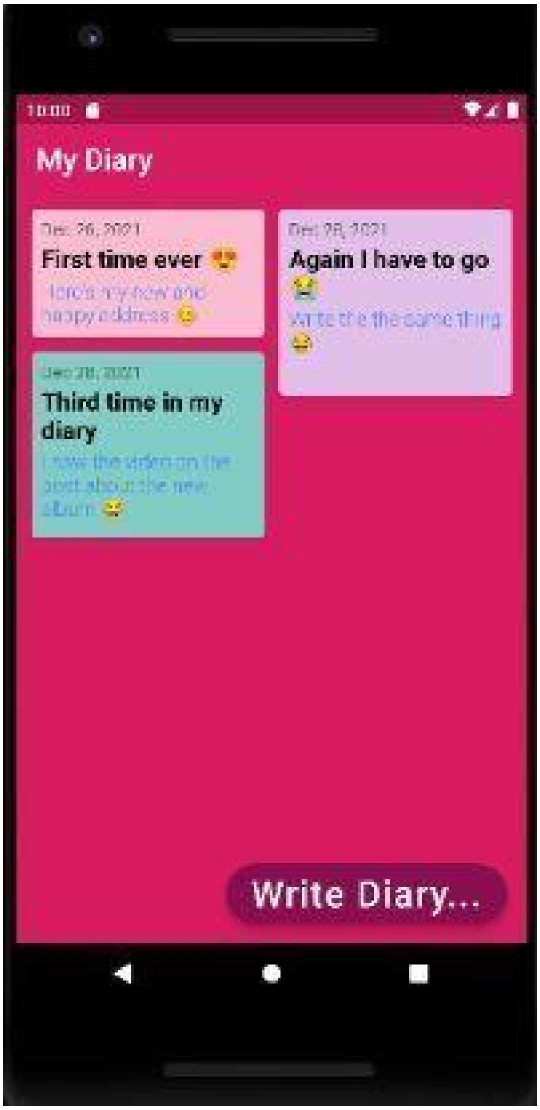


Figure 8.14 – Home page of My Diary SQLite database app in flutter

We've already added three entries. As a result it shows like the above screenshot.

Next, we'll see the code snippet of the home page.

```
import 'package:flutter/material.dart';
import 'package:flutter_staggered_grid_view/flutter_stagg\
ered_grid_view.dart';
import '/model/blogs.dart';
import '/model/blog.dart';
import 'edit.dart';
import 'detail.dart';
import '/controller/blog_card.dart';

class AllPages extends StatefulWidget {
const AllPages({Key? key}) : super(key: key);

@override
_AllPagesState createState() => _AllPagesState();
}

class _AllPagesState extends State<AllPages> {
late List<Blog> blogs;
bool isLoading = false;

@override
void initState() {
    super.initState();

    refreshingAllBogs();
}

@override
void dispose() {
    BlogDatabaseHandler.instance.close();

    super.dispose();
}

```

```
Future refreshingAllBogs() async {
    setState(() => isLoading = true);

    blogs = await BlogDatabaseHandler.instance.readAllBlo\
gs();

    setState(() => isLoading = false);
}

@override
Widget build(BuildContext context) => Scaffold(
        appBar: AppBar(
        title: const Text(
            'My Diary',
            style: TextStyle(fontSize: 24),
        ),
        ),
        body: Center(
        child: isLoading
            ? const CircularProgressIndicator()
            : blogs.isEmpty
                ? const Text(
                    'No Entry in the beginning...',
                    style: TextStyle(color: Colors.white,\
 fontSize: 60),
                    )
                : buildingAllBlogs(),
        ),
        floatingActionButton: FloatingActionButton.extend\
ed(
        tooltip: 'Write Diary...',
        foregroundColor: Colors.white,
        backgroundColor: Colors.pink.shade900,
        onPressed: () async {
            await Navigator.of(context).push(
            MaterialPageRoute(builder: (context) => const\
 EditPage()),
            );

            refreshingAllBogs();
        },
        label: const Text(
            'Write Diary...',
            style: TextStyle(
            fontSize: 30,
            ),
        ),
        ),
    );

Widget buildingAllBlogs() => StaggeredGridView.countBuild\
er(
        padding: const EdgeInsets.all(8),
        itemCount: blogs.length,
        staggeredTileBuilder: (index) => const StaggeredT\
ile.fit(2),
        crossAxisCount: 4,
        mainAxisSpacing: 4,
        crossAxisSpacing: 4,
        itemBuilder: (context, index) {
        final blog = blogs[index];

        return GestureDetector(
            onTap: () async {
            await Navigator.of(context).push(MaterialPage\
Route(
                builder: (context) => DetailPage(blogId: \
blog.id!),
            ));

```

```
                refreshingAllBogs();
                },
                child: BlogCard(blog: blog, index: index),
            );
            },
        );
}
```

We have a list of all blog entries. If it's empty, it will show an empty page with no contents. If not, it will take us to a controller.

```
import 'package:flutter/material.dart';
import 'package:intl/intl.dart';
import '../model/blog.dart';

/// these shades of colors will appear on
/// the display screen and it will appear
/// based on the index of the list
final shadeOfColors = [
Colors.pink.shade100,
Colors.purple.shade100,
Colors.teal.shade200,
Colors.orange.shade200,
Colors.white10,
];

class BlogCard extends StatelessWidget {
const BlogCard({
    Key? key,
    required this.blog,
    required this.index,
}) : super(key: key);

final Blog blog;
final int index;
```

```

@override
Widget build(BuildContext context) {
    final color = shadeOfColors[index % shadeOfColors.len\
gth];
    final time = DateFormat.yMMMd().format(blog.createdTi\
me);
    final minHeight = getMinHeight(index);

    return Card(
    color: color,
    child: Container(
        constraints: BoxConstraints(minHeight: minHeight),
        padding: const EdgeInsets.all(8),
        child: Column(
        mainAxisSize: MainAxisSize.min,
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
            Text(
            time,
            style: TextStyle(color: Colors.grey.shade700),
            ),
            const SizedBox(height: 4),
            Text(
            blog.title,
            style: const TextStyle(
                color: Colors.black,
                fontSize: 20,
                fontWeight: FontWeight.bold,
            ),
            ),
            const SizedBox(
            height: 5,
            ),
            Text(
            blog.description,
            style: const TextStyle(
                color: Colors.blueAccent,
                fontSize: 16,
                fontWeight: FontWeight.w300,
            ),
            ),
        ],
        ),
    ),
    );
}

double getMinHeight(int index) {
    switch (index % 4) {
    case 0:
        return 100;
    case 1:
        return 150;
    case 2:
        return 150;
    case 3:
        return 100;
    default:
        return 100;
    }
}
}
```

The each Card widget has been defined here to show all the items.

Moreover, we can also start writing the content also. In that case, the home page will take us to the Edit page.

```
import 'package:flutter/material.dart';
import '../model/blogs.dart';
import '../model/blog.dart';
import '../controller/blog_form.dart';

class EditPage extends StatefulWidget {
final Blog? blog;

const EditPage({
    Key? key,
    this.blog,
}) : super(key: key);
@override
_EditPageState createState() => _EditPageState();
}

class _EditPageState extends State<EditPage> {
final _formKey = GlobalKey<FormState>();
late bool isImportant;
late int number;
late String title;
late String description;

@override
void initState() {
    super.initState();

    title = widget.blog?.title ?? '';
    description = widget.blog?.description ?? '';
}

@override
Widget build(BuildContext context) => Scaffold(
        appBar: AppBar(
        actions: [buildButton()],
        ),
        body: Form(
        key: _formKey,
        child: BlogForm(
            title: title,
            description: description,
            onChangedTitle: (title) => setState(() => thi\
s.title = title),
            onChangedDescription: (description) =>
                setState(() => this.description = descrip\
tion),
        ),
        ),
    );

Widget buildButton() {
    final isFormValid = title.isNotEmpty && description.i\
sNotEmpty;

    return Padding(
    padding: const EdgeInsets.symmetric(vertical: 8, hori\
zontal: 12),
    child: ElevatedButton(
        style: ElevatedButton.styleFrom(
        onPrimary: Colors.white,
        onSurface: Colors.pink.shade900,
        shadowColor: Colors.grey.shade600,
        primary: isFormValid ? Colors.pink.shade900 : Col\
ors.pink.shade900,
        ),
        onPressed: addOrUpdateBlog,
        child: const Text(
        'Add or Update',
        style: TextStyle(
            fontSize: 20,
        ),
        ),
    ),
    );
}

void addOrUpdateBlog() async {
    final isValid = _formKey.currentState!.validate();

    if (isValid) {
    final isUpdating = widget.blog != null;

    if (isUpdating) {
        await updateBlog();
    } else {
        await addBlog();
    }

    Navigator.of(context).pop();
    }
}

Future updateBlog() async {
    final blog = widget.blog!.copy(
    title: title,
    description: description,
    );

    await BlogDatabaseHandler.instance.update(blog);
}

Future addBlog() async {
    final blog = Blog(
    title: title,
    description: description,
    createdTime: DateTime.now(),
    );

    await BlogDatabaseHandler.instance.create(blog);
}
}
```

Here we can start writing the fresh content just like the following screenshot.

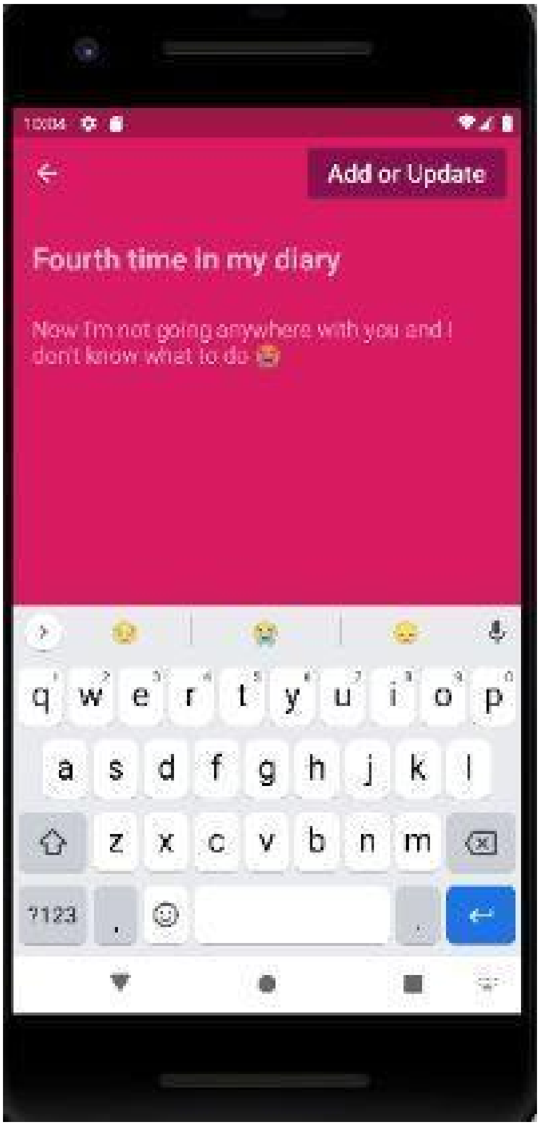

Add or Update
Fourth time in my diary
Now I'm not going anywhere with you and I don't know what to do

Figure 8.15 – Writing content for my diary SQLite database in Flutter

After the writing is over, we can press the "Add or Update" button and insert the data to SQLite database.

How Do we insert data to SQLite Database?

Well, we need a text controller or a Form that will take care of it? Right?

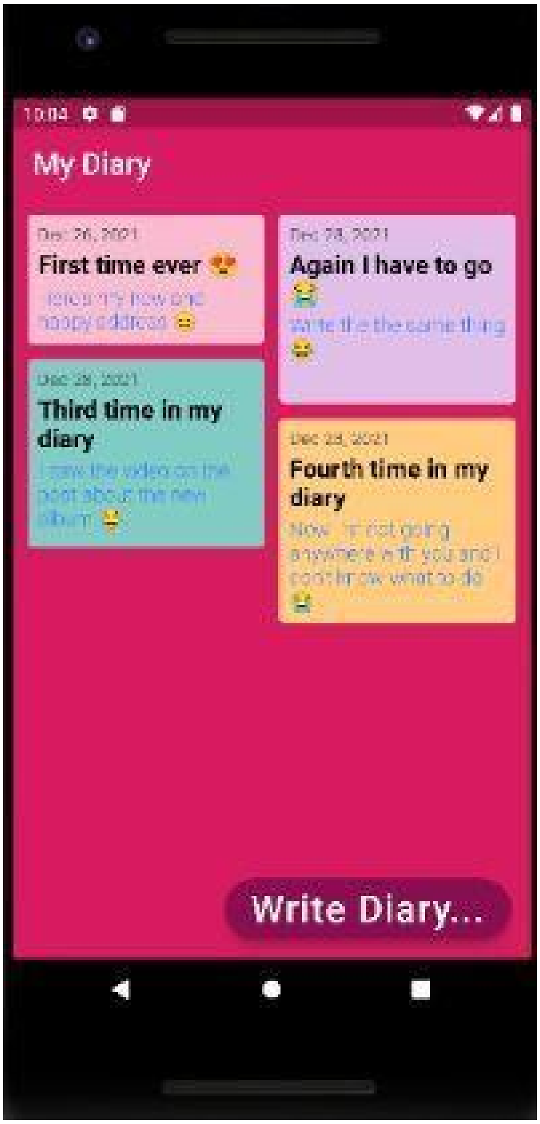
The user must be able to write her posts in this application. Whether, she is writing a blog post or adding her diary entry, that does not matter.

To make it happen, we need another blog form controller.

```
import 'package:flutter/material.dart';

class BlogForm extends StatelessWidget {
final String? title;
final String? description;

final ValueChanged<String> onChangedTitle;
final ValueChanged<String> onChangedDescription;

const BlogForm({
    Key? key,
    this.title = '',
    this.description = '',
    required this.onChangedTitle,
    required this.onChangedDescription,
}) : super(key: key);

@override
Widget build(BuildContext context) => SingleChildScrollVi\
ew(
        child: Padding(
        padding: const EdgeInsets.all(16),
        child: Column(
            mainAxisSize: MainAxisSize.min,
            children: [
            buildTitle(),
            const SizedBox(height: 8),
            buildDescription(),
            const SizedBox(height: 16),
            ],
        ),
        ),
    );

Widget buildTitle() => TextFormField(
        maxLines: 1,
        initialValue: title,
        style: const TextStyle(
        color: Colors.white70,
        fontWeight: FontWeight.bold,
        fontSize: 24,
        ),
        decoration: const InputDecoration(
        border: InputBorder.none,
        hintText: 'Here Title...',
        hintStyle: TextStyle(color: Colors.white70),
        ),
        validator: (title) =>
            title != null && title.isEmpty ? 'Title canno\
t be empty' : null,
        onChanged: onChangedTitle,
    );

Widget buildDescription() => TextFormField(
        maxLines: 5,
        initialValue: description,
        style: const TextStyle(color: Colors.white60, fon\
tSize: 18),
        decoration: const InputDecoration(
        border: InputBorder.none,
        hintText: 'Here description...',
        hintStyle: TextStyle(color: Colors.white60),
        ),
        validator: (title) => title != null && title.isEm\
pty
            ? 'Description cannot be empty'
            : null,
        onChanged: onChangedDescription,
    );
}
```

We need two text form field to accept the data from user.

Once we are over, the added items show on the screen.


10:04

My Diary

Dec 26, 2021
First time ever 😍
[illegible] 😑

Dec 28, 2021
Again I have to go 😭
[illegible] 😆

Dec 28, 2021
Third time in my diary
[illegible] 😝

Dec 28, 2021
Fourth time in my diary
[illegible] 😭

Write Diary...

Figure 8.16 – Added item show on Home page

If we want to edit any of the items, we can just tap the item and gesture detector will take us to detail page where we find the edit and delete buttons.

However, in this time, we can edit them. In addition, we can also delete them as well.

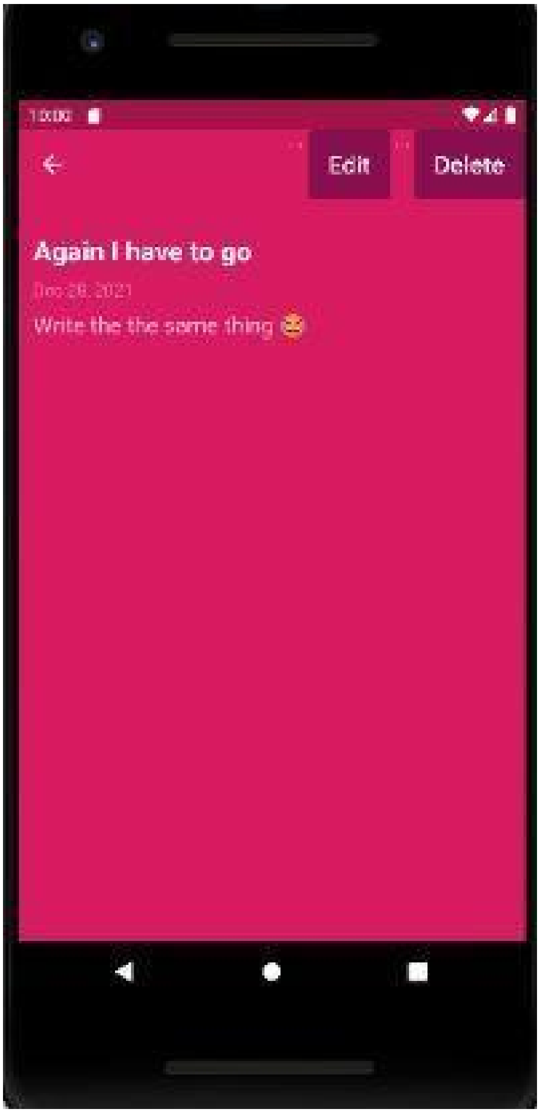


Figure 8.17 – Edit or delete any item in SQLite database

This page has been taken care of by another page, that basically assists our edit page to maintain the state and allow us to edit or delete.

```dart
import 'package:flutter/material.dart';
import 'package:intl/intl.dart';
import '../model/blogs.dart';
import '../model/blog.dart';
import 'edit.dart';

class DetailPage extends StatefulWidget {
final int blogId;

const DetailPage({
    Key? key,
    required this.blogId,
}) : super(key: key);

@override
_DetailPageState createState() => _DetailPageState();
}

class _DetailPageState extends State<DetailPage> {
late Blog blog;
bool isLoading = false;

@override
void initState() {
    super.initState();

    refreshBlog();
}

Future refreshBlog() async {
    setState(() => isLoading = true);

    blog = await BlogDatabaseHandler.instance.readBlog(wi\
dget.blogId);
```

```

    setState(() => isLoading = false);
}

@override
Widget build(BuildContext context) => Scaffold(
        appBar: AppBar(
        actions: [
            const Text(' ... '),
            editButton(),
            const Text(' ... '),
            deleteButton(),
        ],
        ),
        body: isLoading
            ? const Center(child: CircularProgressIndicat\
or())
            : Padding(
                padding: const EdgeInsets.all(12),
                child: ListView(
                padding: const EdgeInsets.symmetric(verti\
cal: 8),
                children: [
                    const SizedBox(height: 10),
                    Text(
                    blog.title,
                    style: const TextStyle(
                        color: Colors.white,
                        fontSize: 22,
                        fontWeight: FontWeight.bold,
                    ),
                    ),
                    const SizedBox(height: 10),
                    Text(
                    DateFormat.yMMMd().format(blog.create\
dTime),
                    style: const TextStyle(color: Colors.\
white38),
                    ),
                    const SizedBox(height: 10),
                    Text(
                    blog.description,
                    style:
                        const TextStyle(color: Colors.whi\
te70, fontSize: 18),
                    )
                ],
                ),
            ),
    );

Widget editButton() => ElevatedButton(
        style: ElevatedButton.styleFrom(
        onPrimary: Colors.white,
        onSurface: Colors.pink.shade900,
        shadowColor: Colors.grey.shade600,
        primary: Colors.pink.shade900,
        ),
        onPressed: () async {
        if (isLoading) return;

        await Navigator.of(context).push(
            MaterialPageRoute(
            builder: (context) => EditPage(blog: blog),
            ),
        );

        refreshBlog();
        },
        child: const Text(
        'Edit',
        style: TextStyle(
            fontSize: 20,
        ),
        ),
    );

Widget deleteButton() => ElevatedButton(
        style: ElevatedButton.styleFrom(
        onPrimary: Colors.white,
        onSurface: Colors.pink.shade900,
        shadowColor: Colors.grey.shade600,
        primary: Colors.pink.shade900,
        ),
        onPressed: () async {
        await BlogDatabaseHandler.instance.delete(widget.\
blogId);

        Navigator.of(context).pop();
        },
        child: const Text(
        'Delete',
        style: TextStyle(
            fontSize: 20,
        ),
        ),
    );
}
```

As we can see in the above code, we have edit and delete buttons both in our AppBar.

Now. we've accomplished the task of building a Blog SQLite database application in Flutter. Of course, we can use the same application as a My Diary.

For the full code snippet for this SQLite database application in

Flutter, please visit the respective GitHub repository.

- All related code with this section[27]

Scoped Model, Provider, SQLite Database and FutureBuilder

In this section we will delve deep into SQLite database and Flutter. However, we change the State using Scoped Model and Provider.

SQLite with Provider in Flutter

Without a stateful widget can we use SQLite database in Flutter? Yes, with Provider.

Can we use SQLite database with Provider package in Flutter? The answer is, yes! We can.

Not only that, we can also reduce pressure on system resource while we store persistent data with provider package.

Most importantly, we always want to make our Flutter app faster and performant. Since storing persistent data requires a lot of state management, the Provider package always helps us to achieve that target.

As a result, in this section, we try to use SQLite database with Provider.

No stateful widget. No extra widget rebuilding. We've kept things quite simple. However, if you are a beginner, please learn Future, await and async first. As we've discussed Future, await and async for absolute beginners in previous section- is Flutter single thread?

[27]https://github.com/sanjibsinha/flutter_data_and_backend

Therefore, if you're a beginner, you might take a look before we proceed towards the final section.

Before using provider package, we've built an entire Blog, or My Diary application using SQLite Database in Flutter.

Not only that, before doing that, in a step by step process we've built the SQLite Blog Application in Flutter. We might also see the progress of the initial phase in this section – SQLite Blog application in Flutter.

The previous section has discussed the application structure.

In the second part, we'd concentrated on database connection, data model classes.

Firstly, we'll use a Flutter package or plugin, sqflite which is available in pub.dev.

Secondly, we also need to use Future API, async, await keywords, and then functions to make it successful.

And finally, we are going to store persistent data in our local SQLite database, using provider package.

What is Sqflite flutter?

The sqflite is a very useful SQLite plugin for Flutter. It supports iOS, Android and MacOS.

For any type of complex CRUD operations, we get support from this plugin, or package. Moreover, this plugin supports transactions and batches.

Therefore, we have helpers for insert, query, update and delete queries. Above all, the DB operation executed in a background thread on iOS and Android. As a result, we get a much faster Flutter application than any other backend operation.

We need to add the dependencies first to our pubspec.yaml file.

```
dependencies:
cupertino_icons: ^1.0.2
flutter:
    sdk: flutter
path: ^1.8.0
provider: ^6.0.2
sqflite: ^2.0.1
```

The three packages in bold, are necessary to build our first Name-Keeper Flutter Application using SQLite database and Provider.

How do I get data from SQLite database in flutter?

Our next challenge is to create a helper class. It will not only create the SQLite database in a given path, but also create the table. Moreover, it will insert and retrieve data.

Besides the helper class, we need a User data model, and a User Provider class that will notify the listeners.

We'll take a look at the classes separately and try to understand how they work together. We have kept three classes in our model folder.

Firstly, the helper class.

```
import 'package:sqflite/sqflite.dart';
import 'package:path/path.dart';

import 'user.dart';

class DatabaseHandler {
Future<Database> initializeDB() async {
    String path = await getDatabasesPath();
    return openDatabase(
    join(path, 'usereleven.db'),
    onCreate: (database, version) async {
        await database.execute(
        "CREATE TABLE usereleven(id INTEGER PRIMARY KEY A\
UTOINCREMENT, name TEXT NOT NULL, location TEXT NOT NULL)\
",
        );
    },
    version: 1,
    );
}

Future<int> insertUser(List<User> users) async {
    int result = 0;
    final Database db = await initializeDB();
    for (var user in users) {
    result = await db.insert('usereleven', user.toMap());
    }
    return result;
}

Future<List<User>> retrieveUsers() async {
    final Database db = await initializeDB();
    final List<Map<String, Object?>> queryResult = await \
db.query('usereleven');
    return queryResult.map((e) => User.fromMap(e)).toList\
();
}
}
```

The above code is quite verbose and meaningful. The database handler class will first define a path where SQLite database gets created.

After that, it will create a table with ID auto increment, and two columns where we store the name and location.

Finally, it defines two methods to insert and retrieve data from the local database.

However, the Future object wants a list of Users that it can map to list so that we can finally get them on screen after the insertion is over.

Therefore, let's take a look at the User class, next.

```
class User {
final int? id;
final String name;
final String location;

User({
    this.id,
    required this.name,
    required this.location,
});

User.fromMap(Map<String, dynamic> res)
    : id = res["id"],
        name = res["name"],
        location = res["location"];

Map<String, Object?> toMap() {
    return {
    'id': id,
    'name': name,
    'location': location,
    };
}
}
```

Now we need a user provider class that will notify the listeners when we press the button "Add Users" like the following screenshot.

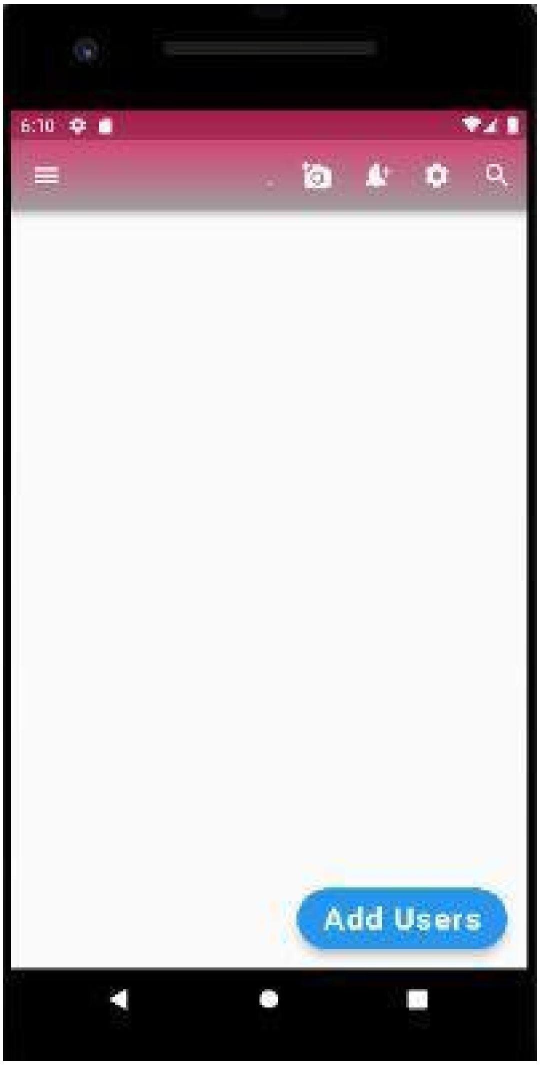


Figure 8.18 – SQLite database and Provider in Flutter first screen

At the same time, we'll also keep the Provider at the top of the root widget.

```
import 'package:flutter/material.dart';
import '/model/user_provider.dart';
import 'view/my_app.dart';

void main() {
Provider.debugCheckInvalidValueType = null;
runApp(
    MultiProvider(
    providers: [
        ChangeNotifierProvider(create: (_) => UserProvide\
r()),
    ],
    child: const MyApp(),
    ),
);
}
```

The User Provider class plays the most important role in this SQLite database and Flutter application.

```
import 'package:flutter/material.dart';

import 'database_handler.dart';
import 'user.dart';

final handler = DatabaseHandler();

class UserProvider with ChangeNotifier {
User _userOne = User(name: 'Hagudu', location: 'Japan');
User get userOne => _userOne;
/*
User _userTwo = User(name: 'Mutudu', location: 'Hokkaidu'\
);
User get userTwo => _userTwo;

User _userThree = User(name: 'John Smith', location: 'Eas\
t Coast');
User get userThree => _userThree;
*/

void addingUsers() {
    _userOne = _userOne;
    //_userTwo = userTwo;
    //_userThree = userThree;

    notifyListeners();
}
}
```

We've commented out other users as we want to insert one user at a time. We could have added them at once of course.

Is SQLite persistent?

We're going to see how the SQLite database persistently stores data, so we can keep adding one user name and location one after another.

The Change Notifier Provider works with Future builder here, in the my home page widget.

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '/model/user_provider.dart';
import '/model/database_handler.dart';
import '/model/user.dart';

class MyHomePage extends StatelessWidget {
const MyHomePage({Key? key}) : super(key: key);

static const String title = 'Database Handling';

@override
Widget build(BuildContext context) {
    final userProvider = Provider.of<UserProvider>(contex\
t);

    final handler = DatabaseHandler();
    Future<int> addUsers() async {
    User firstUser = User(
        name: userProvider.userOne.name,
        location: userProvider.userOne.location,
    );

    /*
User secondUser = User(
        name: userProvider.userTwo.name,
        location: userProvider.userTwo.location,
    );

    User thirddUser = User(
        name: userProvider.userThree.name,
        location: userProvider.userThree.location,
    );
*/
    List<User> listOfUsers = [
    firstUser,
    //secondUser,
    //thirddUser,
  ];
  return await handler.insertUser(listOfUsers);
  }

  return Scaffold(
  appBar: customAppBar(title),
  body: FutureBuilder(
    future: handler.retrieveUsers(),
    builder: (BuildContext context, AsyncSnapshot<Lis\
t<User>> snapshot) {
    if (snapshot.hasData) {
      return ListView.builder(
      itemCount: snapshot.data?.length,
      itemBuilder: (BuildContext context, int index\
) {
        return Card(
        child: ListTile(
          key: ValueKey<int>(snapshot.data![ind\
ex].id!),
          contentPadding: const EdgeInsets.all(\
8.0),
          title: Text(
          snapshot.data![index].name,
          style: const TextStyle(
            fontSize: 30,
            color: Colors.red,
          ),
          ),
          subtitle: Text(
          snapshot.data![index].location,
          style: const TextStyle(
            fontSize: 20,
                        color: Colors.red,
                    ),
                    ),
                ),
                );
            },
            );
        } else {
            return const Center(child: CircularProgressIn\
dicator());
        }
        },
    ),
    floatingActionButton: FloatingActionButton.extended(
        onPressed: () {
        handler.initializeDB().whenComplete(() async {
            await addUsers();
        });

        userProvider.addingUsers();
        },
        label: const Text(
        'Add Users',
        style: TextStyle(
            fontSize: 25,
            fontWeight: FontWeight.bold,
        ),
        ),
    ),
    );
}

AppBar customAppBar(String title) {
    return AppBar(
    centerTitle: true,
    //backgroundColor: Colors.grey[400],
    flexibleSpace: Container(
        decoration: const BoxDecoration(
        gradient: LinearGradient(
            colors: [
            Colors.pink,
            Colors.grey,
            ],
            begin: Alignment.topRight,
            end: Alignment.bottomRight,
        ),
        ),
    ),
    //elevation: 20,
    titleSpacing: 80,
    leading: const Icon(Icons.menu),
    title: Text(
        title,
        textAlign: TextAlign.left,
    ),
    actions: [
        buildIcons(
        const Icon(Icons.add_a_photo),
        ),
        buildIcons(
        const Icon(
            Icons.notification_add,
        ),
        ),
        buildIcons(
        const Icon(
            Icons.settings,
        ),
        ),
        buildIcons(
          const Icon(Icons.search),
          ),
      ],
      );
}

IconButton buildIcons(Icon icon) {
    return IconButton(
    onPressed: () {},
    icon: icon,
    );
}
}
```

The flow of logic is quite simple. Inside our build method, we've got the Provider of Type User Provider and its context.

Next, we have instantiated the Database handler object. Without this handler we cannot initiate the process of inserting and retrieving data.

As a result, we can press the "Add Users" button that fires the event of inserting and retrieving data from the SQLite database.

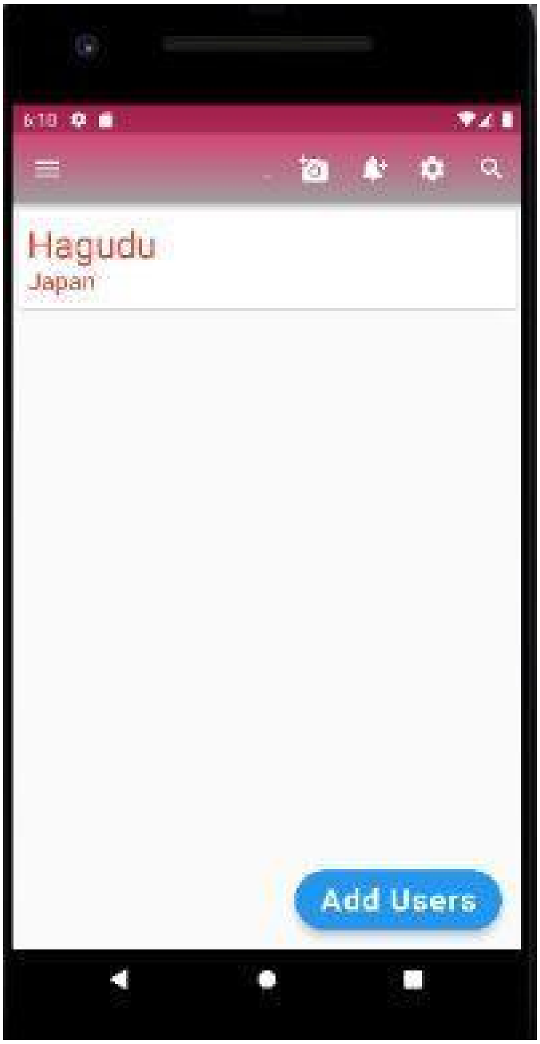


Figure 8.19 – SQLite database and Provider in Flutter with first user

Once we have inserted the first user name and location, we can comment out the first user in User and User Handler class.

Then we can insert the second user's name and location.

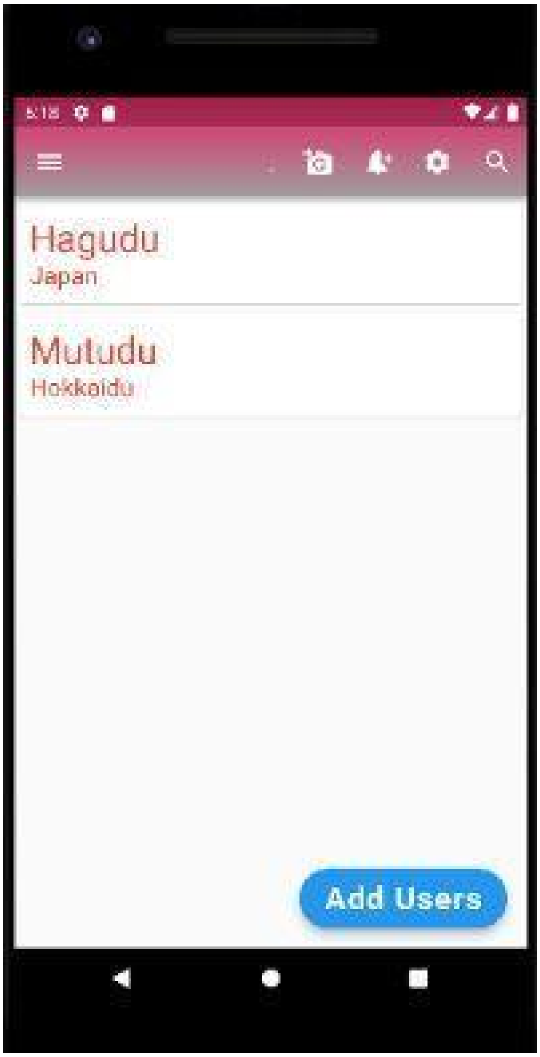


Figure 8.20 – SQLite database and Provider in Flutter with second user

We can clearly see that how SQLite database persists data. However, we don't have to use stateful widget to manage state. The provider package helps us to notify listeners which is a Future builder.

Now, we can add as many user's name and location.

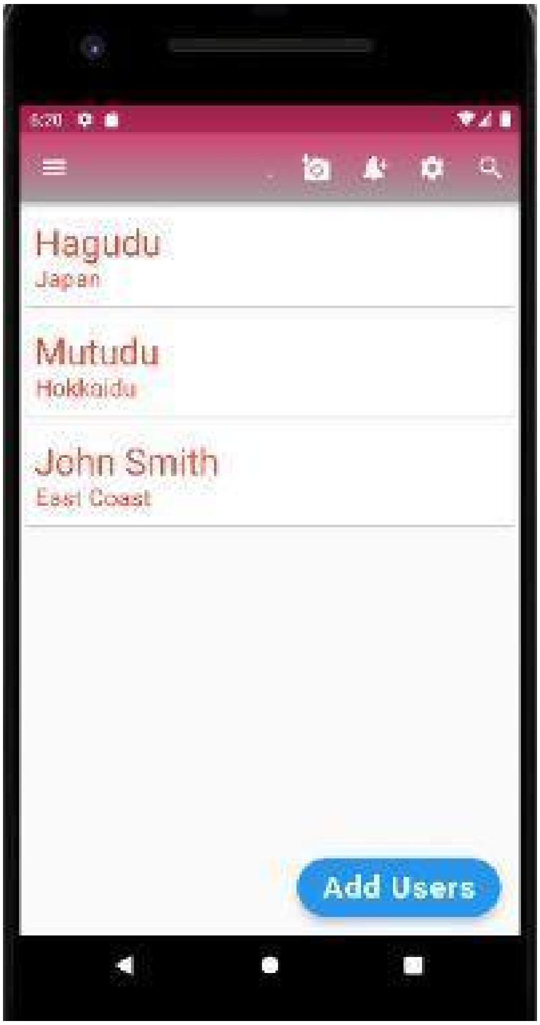


Figure 8.21 – SQLite database and Provider in Flutter with third user

- For the full code snippet please visit the respective GitHub repository.[28]

[28]https://github.com/sanjibsinha/flutter_artisan/tree/final-provider-sqflite

What is Scoped Model in flutter

Scoped Model is the simplest version of Inherited Widget, managing State in Flutter.

As the name suggests, the Scoped Model in Flutter examines the scope and passes data downwards. We can create the scope at the top first with a type. And, after that in the descendant widgets, we can pass that data type.

If you have already learned Provider, then you might sense the similarity. However, provider is more versatile and might be complex. Moreover, provider package uses ChangeNotifier of Flutter.

We'll come to that point later.

Before that, let's see what is Scoped Model, and how it works. In addition, whether we can use scoped model with other packages or not.

The best way to understand any topic is to view images first. Therefore, let's view the Flutter application we've built using Scoped Model.

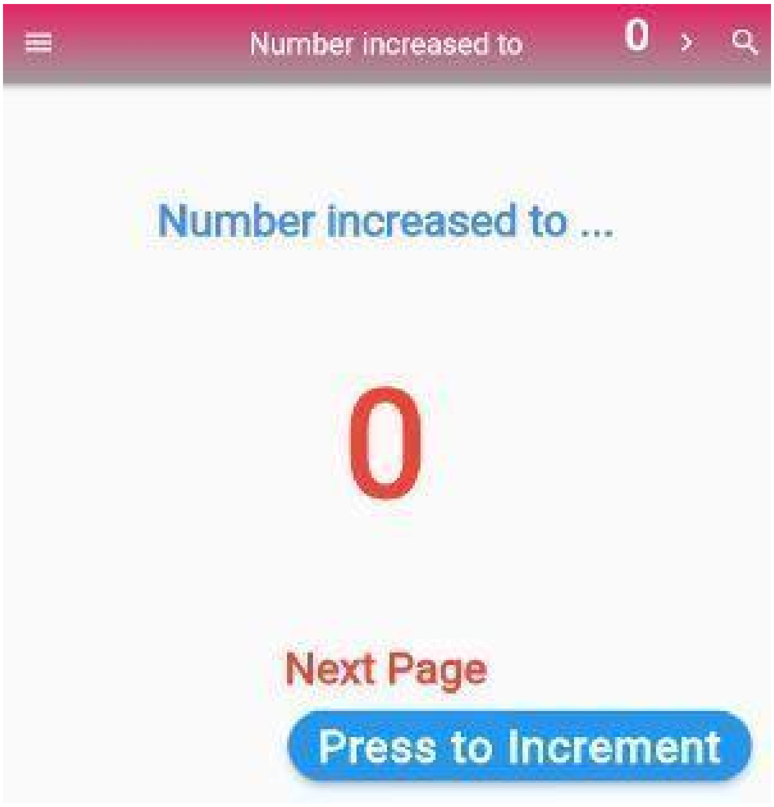


Figure 8.22 – Scoped model example one

While we press the button, the state of the Text widget which display the number changes. As a result, the number increases.

The increment reflects also on the AppBar, and on the screen facing us.

Let’s see the next stage.

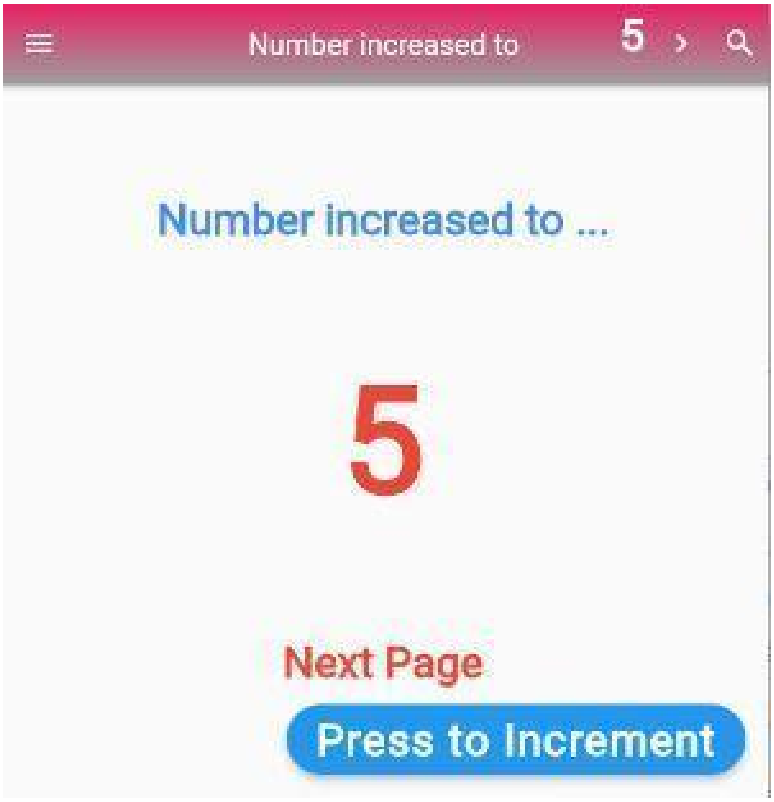


Figure 8.23 – Scoped model example two

We've pressed the button five times, and the number changes in two places.

Now, we're going to press the Next Page button to navigate to another page.

Why?

Because the Next Page is also another descendant of Scoped Model. Therefore, in that case, the number should increase there too.

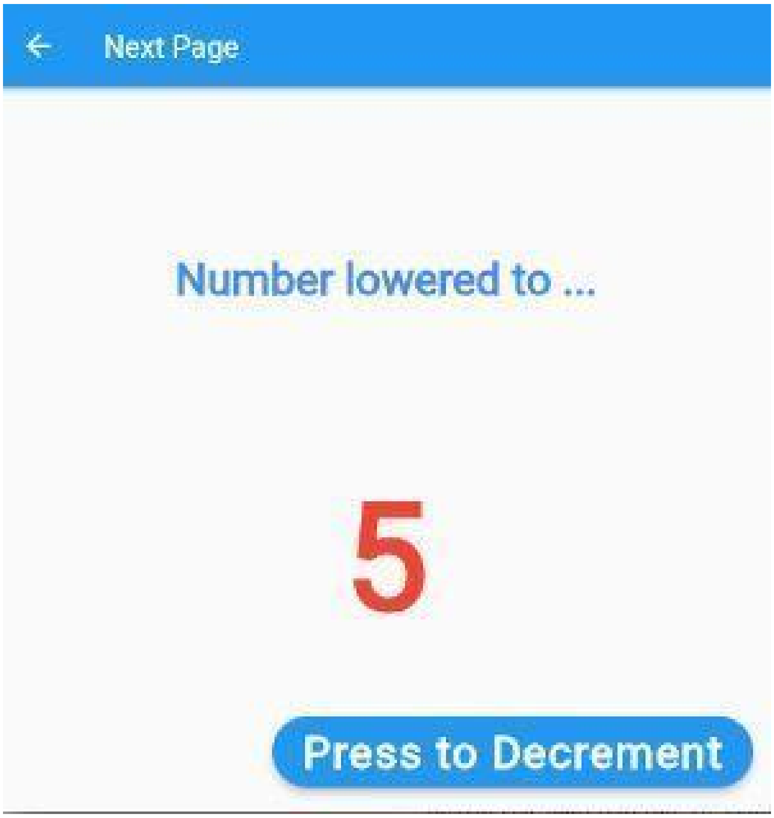


Figure 8.24 – Scoped model example three

Voila! It works.

The number in the Next Page changes simultaneously along with the home page.

Now, in this page, we will press the decrement button. As a result, the number will lower down. But, will that change the number in home page?

Let’s see. First, let’s press the decrement button.

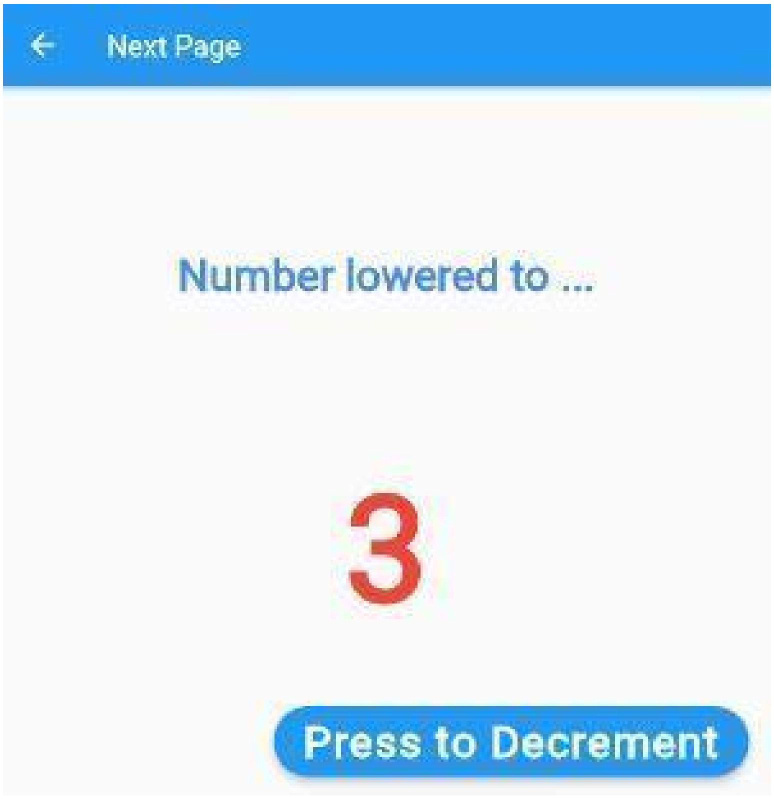


Figure 8.25 – Scoped model example four

We've pressed the decrement button twice and the number lowers down to 3.

Fine.

Next, we move back to the home page.

What do we see?

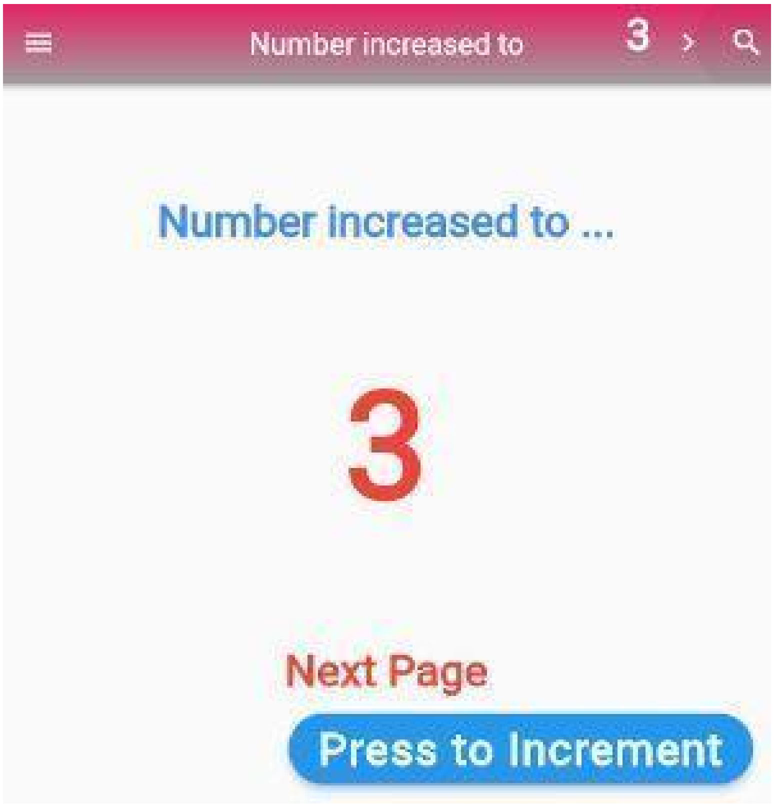


Figure 8.26 – Scoped model example five

The number reduces to 3.

Consequently, we've successfully managed state with the help of Scoped Model.

To sum up, Scoped Model helps us to manage state in a very simple way.

Now, the time has come to inspect the code.

How do you use the scoped model in Flutter?

Firstly, we need to add the dependency.

```
dependencies:
cupertino_icons: ^1.0.4
flutter:
    sdk: flutter

scoped_model: ^2.0.0-nullsafety.0
```

We've made it sure that the scoped_model package we're using must adhere to the null safety.

Next, we should create a model class of Counter.

```
import 'package:scoped_model/scoped_model.dart';

class Counter extends Model {
int _counter = 0;
int get counter => _counter;
void increment() {
    _counter++;
    notifyListeners();
}

void decrement() {
    _counter--;
    notifyListeners();
}
}
```

Our Counter class extends the Model class provided by the package. As a result, the Counter object can notify all the components that subscribe to all the public properties and methods of Counter type.

Therefore, we need to declare the scope at the top and mention the type which is Counter here.

```
import 'package:flutter/material.dart';
import 'package:provider_et_sqflite/model/counter.dart';
import 'package:scoped_model/scoped_model.dart';

import 'my_home_page.dart';

class MyApp extends StatelessWidget {
const MyApp({
    Key? key,
    required this.counter,
}) : super(key: key);
final Counter counter;

// This widget is the root of your application.
@override
Widget build(BuildContext context) {
    return ScopedModel<Counter>(
    model: Counter(),
    child: MaterialApp(
        debugShowCheckedModeBanner: false,
        title: 'Scoped Model Simple',
        theme: ThemeData(
        primarySwatch: Colors.blue,
        ),

        /// child widgets are now under its scope
        /// and we can use this model anywhere below
        ///
        home: const MyHomePage(),
    ),
    );
}
}
```

As a result, we can pass the Counter object to all the descendants.

First see the Home page code.

```
import 'package:flutter/material.dart';
import 'package:provider_et_sqflite/model/counter.dart';
import 'package:scoped_model/scoped_model.dart';

import 'next_page.dart';

class MyHomePage extends StatelessWidget {
const MyHomePage({Key? key}) : super(key: key);

static const String title = 'Number increased to';

@override
Widget build(BuildContext context) {
    return Scaffold(
    appBar: customAppBar(title),

    /// the child widget below can use the scoped model
    ///
    floatingActionButton: ScopedModelDescendant<Counter>(
        builder: (context, child, model) => FloatingActio\
nButton.extended(
        onPressed: () {
            model.increment();
        },
        label: const Text(
            'Press to Increment',
            style: TextStyle(
            fontSize: 30,
            fontWeight: FontWeight.w600,
            ),
        ),
        ),
    ),

    /// the child widget below can use the scoped model
    ///
    body: ScopedModelDescendant<Counter>(
        builder: (context, child, model) => Column(
                mainAxisAlignment: MainAxisAlignment.spac\
eEvenly,
                children: [
                const Text(
                    'Number increased to ...',
                    style: TextStyle(
                    fontSize: 30,
                    fontWeight: FontWeight.w600,
                    color: Colors.blueAccent,
                    ),
                ),
                Center(
                    child: Text(
                    model.counter.toString(),
                    style: const TextStyle(
                        fontSize: 100,
                        fontWeight: FontWeight.w600,
                        color: Colors.red,
                    ),
                    ),
                ),
                TextButton(
                    onPressed: () {
                    Navigator.push(
                        context,
                        MaterialPageRoute(
                        builder: (context) => const NextP\
age(),
                        ),
                    );
                    },
                    child: const Text(
                    'Next Page',
                    style: TextStyle(
                        fontSize: 30,
                        fontWeight: FontWeight.w600,
                        color: Colors.red,
                    ),
                    ),
                )
                ],
            )),
    );
}

AppBar customAppBar(String title) {
    return AppBar(
    centerTitle: true,
    //backgroundColor: Colors.grey[400],
    flexibleSpace: Container(
        decoration: const BoxDecoration(
        gradient: LinearGradient(
            colors: [
            Colors.pink,
            Colors.grey,
            ],
            begin: Alignment.topRight,
            end: Alignment.bottomRight,
        ),
        ),
    ),
    //elevation: 20,
    titleSpacing: 80,
    leading: const Icon(Icons.menu),
    title: Text(
        title,
        textAlign: TextAlign.left,
    ),
    actions: [
        ScopedModelDescendant<Counter>(
        builder: (context, child, model) => Container(
            padding: const EdgeInsets.all(5),
            child: Text(
            model.counter.toString(),
            style: const TextStyle(
                fontSize: 30,
                fontWeight: FontWeight.w900,
                color: Colors.white,
            ),
            ),
        ),
        ),
        buildIcons(
        const Icon(
            Icons.navigate_next,
        ),
        ),
        buildIcons(
        const Icon(Icons.search),
        ),
    ],
    );
}

IconButton buildIcons(Icon icon) {
    return IconButton(
    onPressed: () {},
    icon: icon,
    );
}
}
```

Now, as we go down the widget tree, we can use the Scoped Model Descendant with the type. So the descendant widgets can access that type with the help of model.

```
floatingActionButton: ScopedModelDescendant<Counter>(
        builder: (context, child, model) => FloatingActio\
nButton.extended(
        onPressed: () {
            model.increment();
        },
....
```

In the same vein, the Next Page is the another descendant of Scoped Model. Consequently, at that page we can access all properties and methods of Counter class.

Moreover, the same state prevails across all the descendant widgets.

Let's take a look at the Next Page code.

```
import 'package:flutter/material.dart';
import 'package:provider_et_sqflite/model/counter.dart';
import 'package:scoped_model/scoped_model.dart';

class NextPage extends StatelessWidget {
const NextPage({Key? key}) : super(key: key);

static const String title = 'Next Page';

@override
Widget build(BuildContext context) {
    return Scaffold(
    appBar: AppBar(
        title: const Text(title),
    ),

    /// the child widget below can use the scoped model
    ///
    floatingActionButton: ScopedModelDescendant<Counter>(
        builder: (context, child, model) => FloatingActio\
nButton.extended(
        onPressed: () {
            model.decrement();
        },
        label: const Text(
            'Press to Decrement',
            style: TextStyle(
            fontSize: 30,
            fontWeight: FontWeight.w600,
            ),
        ),
        ),
    ),

    /// the child widget below can use the scoped model
    ///
    body: ScopedModelDescendant<Counter>(
        builder: (context, child, model) => Column(
                mainAxisAlignment: MainAxisAlignment.spac\
eEvenly,
                children: [
                const Text(
                    'Number lowered to ...',
                    style: TextStyle(
                    fontSize: 30,
                    fontWeight: FontWeight.w600,
                    color: Colors.blueAccent,
                    ),
                ),
                Center(
                    child: Text(
                    model.counter.toString(),
                    style: const TextStyle(
                        fontSize: 100,
                        fontWeight: FontWeight.w600,
                        color: Colors.red,
                    ),
                    ),
                ),
                ],
            )),
    );
}

AppBar customAppBar(String title) {
    return AppBar(
    centerTitle: true,
    //backgroundColor: Colors.grey[400],
    flexibleSpace: Container(
        decoration: const BoxDecoration(
        gradient: LinearGradient(
            colors: [
            Colors.pink,
            Colors.grey,
            ],
            begin: Alignment.topRight,
            end: Alignment.bottomRight,
        ),
        ),
    ),
    //elevation: 20,
    titleSpacing: 80,
    leading: const Icon(Icons.menu),
    title: Text(
        title,
        textAlign: TextAlign.left,
    ),
    actions: [
        ScopedModelDescendant<Counter>(
        builder: (context, child, model) => Container(
            padding: const EdgeInsets.all(5),
            child: Text(
            model.counter.toString(),
            style: const TextStyle(
                fontSize: 30,
                fontWeight: FontWeight.w900,
                color: Colors.white,
            ),
            ),
        ),
        ),
        buildIcons(
        const Icon(
            Icons.navigate_next,
        ),
        ),
        buildIcons(
        const Icon(Icons.search),
        ),
    ],
    );
}

IconButton buildIcons(Icon icon) {
    return IconButton(
    onPressed: () {},
    icon: icon,
    );
}
}
```

As a result, we've seen how we can manage the state across the whole flutter application.

Passing data across the Flutter Application

A Flutter User Interface can pass data all the way down the tree from parent to child. We can always pass them through constructors, or use Inherited widget.

But, doing that manually makes it cumbersome as our Flutter application gets bigger.

To solve this issue, we can use Scoped Model, which is a simplified version of Inherited widget. Of course, Provider works on the same principle, although having a lot more options.

Moreover, using Provider is not as easy as using Scoped Model. In both cases, we need to make a new Context.

Why?

Because the exposed data is included in the Context. Now any widget that uses that Context can access that data.

Now it's always a good practice to place that access point as low as possible. In fact, we should use Scoped Model, or Provider as closest as possible to the widget that uses that exposed data.

The biggest advantage of using Scoped Model is it separates the User Interface and Business Logic. And that too in a very simple way.

To get the full code please visit the respective GitHub Repository.

- For the full code snippet please visit the respective GitHub repository.[29]

[29]https://github.com/sanjibsinha/provider_et_sqflite/tree/scoped-model-first

Scoped Model and SQLite in Flutter

Just like Provider, we can use Scoped Model with SQLite database in Flutter.

In the previous section we've seen how we can use Scoped Model in Flutter. In a couple of previous sections, we've also examined the scope of using SQLite Database in Flutter. In this section we'll discuss how we can join Scoped Model and SQLite database, so that they work at tandem in Flutter.

We're going to build a Note-keeper app where we'll store Name and Location of users. Likewise, we've already built the same application with Provider and SQLite Database. If you have interest, please check it.

Firstly, we can always use SQLite database in Flutter. However, we need to use a special package or plugin sqflite which is available in pub.dev. We also need to use Future API, async, await keywords, and then functions to make it successful.

We've discussed this feature for absolute beginners in previous section, is Flutter single thread? If you're a complete beginner searching to know about Future in Flutter, please check it.

First thing first, to use SQLite Database in Flutter, we use the sqflite package.

Why?

Because this package provides classes and functions to interact with a SQLite database. There are other reasons too, using SQLite database is better than using a local file, or key-value store.

In addition, SQLite database provides faster CRUD. That is, we can create, retrieve, update and delete data. And, it's always better than any other local persistent solutions.

Besides sqflite package, we need to use another package path, that will define the location for storing the database on the disk.

For the beginners, here is a guide what SQLite database is.

What is SQLite database and how it works?

SQLite is a C-language library that implements many features at one go. It is small, fast, self-contained, high-reliability, full-featured, SQL database engine.

By the way, SQLite is the most used database engine in the world. Besides, SQLite database, file format is stable, cross-platform, and backwards compatible.

There are over 1 trillion SQLite databases in active use at present.

Therefore, let's go ahead and make our first Flutter Application with SQLite database.

Besides, using Scoped Model, and SQLite database, we'll also use Future Builder widget. We'll discuss Future Builder in detail later, meanwhile let's learn a few key points about Future Builder.

Scoped Model, SQLite database and Future Builder

How about getting a gentle introduction to Future Builder?

Well, to understand the whole mechanism behind building a Note-keeper application in Flutter, this initiation might help us.

Future Builder is a widget that builds itself based on the latest snapshot of interaction with a Future.

According to the documentation we must obtain the future object during State.initState, State.didUpdateWidget, or State.didChangeDependencies.

However, while using with Provider or Scoped Model, we use the Future Builder in a different way. We'll see that in a minute.

Next, let’s have another gentle introduction to “scoped_model” package also. Because without this very useful package, we couldn’t use Inherited Widget in the simplest way in Flutter.

How do you use scoped_model in flutter?

To start with we need to add all the dependencies first.

```
dependencies:
cupertino_icons: ^1.0.4
flutter:
    sdk: flutter
path: ^1.8.0
scoped_model: ^2.0.0-nullsafety.0
sqflite: ^2.0.1
```

After that, with the help of Scoped Model we can easily pass a data model from a parent Widget to a child Widget.

Moreover, it also rebuilds all of the children that use the model when the model is updated.

The Scoped Model package provides three main classes.

Let’s see what they are.

Firstly, the Model class as we’ve used in the User Model, like the following.

```
import 'package:scoped_model/scoped_model.dart';

import 'database_handler.dart';
import 'user.dart';

class UserModel extends Model {
User _userOne = User(name: 'Json Web', location: 'Detroit\
');
User get userOne => _userOne;

void addingUsers() {
    _userOne = userOne;

    notifyListeners();
}
}
```

We need another User class that will define the behaviour of user object and map the items to give an output in a List of items.

```
class User {
final int? id;
final String name;
final String location;

User({
    this.id,
    required this.name,
    required this.location,
});

User.fromMap(Map<String, dynamic> res)
    : id = res["id"],
        name = res["name"],
        location = res["location"];

Map<String, Object?> toMap() {
    return {
    'id': id,
    'name': name,
    'location': location,
    };
}
}
```

As a result, we can construct User object inside User model class.

Secondly, we need two more widgets.

The first one is the Scoped Model Widget where we tell the parent widget to pass the Model class properties and methods to its descendant children widgets.

Since we need to pass a Model deep down your Widget hierarchy, you can wrap our Model in a ScopedModel Widget. This will make the Model available to all descendant Widgets.

```
import 'package:flutter/material.dart';

import 'package:provider_et_sqflite/model/user_model.dart\
';
import 'package:scoped_model/scoped_model.dart';

import 'my_home_page.dart';

class MyApp extends StatelessWidget {
const MyApp({
    Key? key,
    required this.user,
}) : super(key: key);
final UserModel user;

// This widget is the root of your application.
@override
Widget build(BuildContext context) {
    return MaterialApp(
    debugShowCheckedModeBanner: false,
    title: 'Scoped Model Simple',
    theme: ThemeData(
        primarySwatch: Colors.blue,
    ),

    /// child widggets are now under its scope
    /// and we can use this model anywhere below
    ///
    home: ScopedModel<UserModel>(
        model: UserModel(),
        child: const MyHomePage(),
    ),
    );
}
}
```

As a result, the Home Page can now make the model available to all descendant widgets.

Finally, we need the Scoped Model Descendant Widget that can listen to the Models for any changes.

```
import 'package:flutter/material.dart';
import 'package:scoped_model/scoped_model.dart';

import '../model/user_model.dart';
import '/model/database_handler.dart';
import '/model/user.dart';

class MyHomePage extends StatelessWidget {
const MyHomePage({Key? key}) : super(key: key);

static const String title = 'Database Handling';

@override
Widget build(BuildContext context) {
    final userModel = ScopedModel.of<UserModel>(context);

    final handler = DatabaseHandler();
    Future<int> addUsers() async {
    User firstUser = User(
        name: userModel.userOne.name,
        location: userModel.userOne.location,
    );
    List<User> listOfUsers = [
        firstUser,
    ];
    return await handler.insertUser(listOfUsers);
    }

    return ScopedModelDescendant<UserModel>(
    builder: (context, child, model) => Scaffold(
        appBar: customAppBar(title),
        body: FutureBuilder(
        future: handler.retrieveUsers(),
        builder: (BuildContext context, AsyncSnapshot<Lis\
t<User>> snapshot) {
```

```
            if (snapshot.hasData) {
            return ListView.builder(
                itemCount: snapshot.data?.length,
                itemBuilder: (BuildContext context, int i\
ndex) {
                return Card(
                    child: ListTile(
                    key: ValueKey<int>(snapshot.data![ind\
ex].id!),
                    contentPadding: const EdgeInsets.all(\
8.0),
                    title: Text(
                        snapshot.data![index].name,
                        style: const TextStyle(
                        fontSize: 30,
                        color: Colors.red,
                        ),
                    ),
                    subtitle: Text(
                        snapshot.data![index].location,
                        style: const TextStyle(
                        fontSize: 20,
                        color: Colors.blue,
                        ),
                    ),
                    ),
                );
                },
            );
            } else {
            return const Center(child: CircularProgressIn\
dicator());
            }
        },
        ),
        floatingActionButton: FloatingActionButton.extend\
ed(
        onPressed: () {
            handler.initializeDB().whenComplete(() async {
            await addUsers();
            });

            model.addingUsers();
        },
        label: const Text(
            'Add Users',
            style: TextStyle(
            fontSize: 25,
            fontWeight: FontWeight.bold,
            ),
        ),
        ),
    ),
    );
}

AppBar customAppBar(String title) {
    return AppBar(
    centerTitle: true,
    //backgroundColor: Colors.grey[400],
    flexibleSpace: Container(
        decoration: const BoxDecoration(
        gradient: LinearGradient(
            colors: [
            Colors.pink,
            Colors.grey,
            ],
            begin: Alignment.topRight,
            end: Alignment.bottomRight,
        ),
        ),
    ),
    //elevation: 20,
    titleSpacing: 80,
    leading: const Icon(Icons.menu),
    title: Text(
        title,
        textAlign: TextAlign.left,
    ),
    actions: [
        buildIcons(
        const Icon(Icons.add_a_photo),
        ),
        buildIcons(
        const Icon(
            Icons.notification_add,
        ),
        ),
        buildIcons(
        const Icon(
            Icons.settings,
        ),
        ),
        buildIcons(
        const Icon(Icons.search),
        ),
    ],
    );
}

IconButton buildIcons(Icon icon) {
    return IconButton(
    onPressed: () {},
    icon: icon,
    );
```

```
}
}
```

The Model has been passed down from the parent to the child Widget tree using an InheritedWidget. When an InheritedWidget is rebuilt, it will rebuild all of the Widgets that depend on its data.

As a result, when we press the "Add Users" button, one User with name and location is added to the SQLite Database. We've already defined the insert and retrieve methods in a Database helper class, like the following.

```
import 'package:sqflite/sqflite.dart';
import 'package:path/path.dart';

import 'user.dart';

class DatabaseHandler {
Future<Database> initializeDB() async {
    String path = await getDatabasesPath();
    return openDatabase(
    join(path, 'userthirteen.db'),
    onCreate: (database, version) async {
        await database.execute(
        "CREATE TABLE userthirteen(id INTEGER PRIMARY KEY\
 AUTOINCREMENT, name TEXT NOT NULL, location TEXT NOT NUL\
L)",
        );
    },
    version: 1,
    );
}

Future<int> insertUser(List<User> users) async {
    int result = 0;
    final Database db = await initializeDB();
    for (var user in users) {
    result = await db.insert('userthirteen', user.toMap()\
);
    }
    return result;
}

Future<List<User>> retrieveUsers() async {
    final Database db = await initializeDB();
    final List<Map<String, Object?>> queryResult =
        await db.query('userthirteen');
    return queryResult.map((e) => User.fromMap(e)).toList\
();
}
}
```

Now, we need to add users through our User Model class, in two steps, like the following.

```
class UserModel extends Model {
User _userOne = User(name: 'Json Web', location: 'Detroit\
');
User get userOne => _userOne;
...
Widget build(BuildContext context) {
    final userModel = ScopedModel.of<UserModel>(context);

    final handler = DatabaseHandler();
    Future<int> addUsers() async {
    User firstUser = User(
        name: userModel.userOne.name,
        location: userModel.userOne.location,
    );
    List<User> listOfUsers = [
        firstUser,
    ];
    return await handler.insertUser(listOfUsers);
...
```

Therefore, we get our first user added to our Name-keeper Flutter application as press the “Add Users” button.

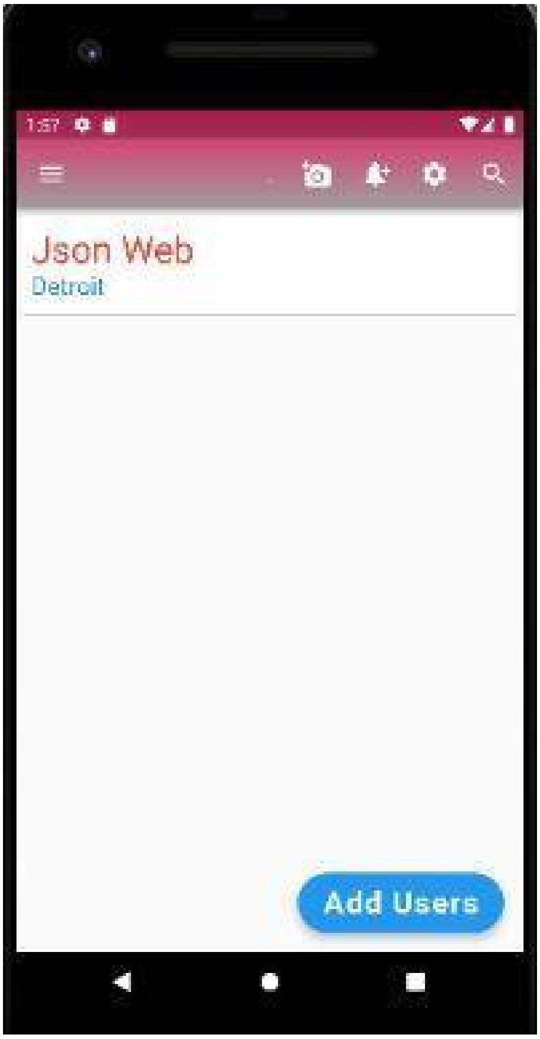


Figure 8.27 – Scoped Model and SQLite database example in Flutter

Now we can keep added them through User Model class by pressing the button. However, we need to manually add them in User Model class which is not advisable.

Yet, to understand the mechanism it's okay for the time being.

Later we'll pass that Model class to the Scoped Model Descendant Widget to initialise SQLite Database handling process by taking user inputs.

So stay tuned.

- For full code snippet please visit the respective GitHub Repository - [30]

What is future builder in Flutter

The FutureBuilder widget obtains Future by change of state, or by dependencies.

Future Builder is a widget that builds itself. But, it requires a snapshot of interaction with a Future.

How will we get the Future?

We must have obtained the Future in three ways.

They are State.initState, State.didUpdateWidget, or State.didChangeDependencies.

What does it mean?

We may have obtained Future either through change of state, or by change in dependencies.

It means a lot.

Why?

The reason is, we can use FutureBuilder using Scoped Model or Provider. In other Words using the Inherited Widget.

In fact, we are going to do the same thing here.

[30]https://github.com/sanjibsinha/provider_et_sqflite/tree/sqflite-scoped-first

We will use both Scoped Model and Provider to insert data to a SQLite database. As a result, the FutureBuilder widget will rebuild itself and retrieve the data.

Actually we will execute asynchronous functions. Consequently the asynchronous function will make the User Interface update.

By default FutureBuilder is stateful in nature. It maintains its own state.

How does FutureBuilder work?

The FutureBuilder shows messages like "loading". It does that based on connection state. And, after that based on new "data", or "snapshot", it updates the UI. As a consequence we get a new view.

The advantage of FutureBuilder is, it does not use two "state variables". When the new "data" arrives, it updates the "view".

To sum up, the FutureBuilder is a wrapper or boilerplate of what we do.

What is the difference between FutureBuilder and StreamBuilder?

We'll discuss #StreamBuilder class in a separate article. However, the StreamBuilder class has some similarities with the FutureBuilder class.

Firstly, they listen to changes that occur on their respective objects.

Secondly, after that, they trigger a new construction when they get the notification.

This is the basic functionality that they both practice.

Then, what is the difference?

The object makes the difference. This is the object that they are listening to.

As we told before, #Future is a representation of asynchronous function. As a result, the #Future has one and only answer.

If the #Future is completed successfully, we get the result. If not, we get the error.

On the other hand, the #Stream will get each new value. Even if it had an error, we would get the last value.

Therefore, the main difference is a #Future cannot listen to a change that varies. The #Future has one single answer always.

Enough talking.

Let us view the screenshots first. Viewing the images of our Flutter Applications will clarify the concept.

After that we will discuss the respective code.

When should I use FutureBuilder?

In our first Flutter Application we are going to use the Scoped Model and the Provider with a single FutureBuilder widget. As a result, it does not work properly.

However, the data has been inserted to the SQLite database properly.

Firstly, we will try to insert data to a SQLite database by a Provider object.

Let us press the Button. According to our code, it should insert two User objects. One from the Provider and the other from the Scoped Model class.

Because the FutureBuilder listens to both objects it should update the UI and display two user names.

But that did not happen.

The second user's name will only be displayed when we click the Button below.

If we click the Button "Add Users by Provider" again, only one name is displayed below the two Users. Although the second user is inserted, it does not reflect on the UI.

Let us check the code where we have mixed the Scoped Model and the Provider.

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';

import 'package:scoped_model/scoped_model.dart';

import '/model/user_model.dart';
import '/model/user_prvider.dart';
import '/model/database_handler.dart';
import '/model/user.dart';

class MyHomePage extends StatelessWidget {
const MyHomePage({Key? key}) : super(key: key);

static const String title = 'Provider, Scoped Model, SQLi\
te';

@override
Widget build(BuildContext context) {
    final userModel = ScopedModel.of<UserModel>(context);
    final userProvider = Provider.of<UserProvider>(contex\
t);

    final handler = DatabaseHandler();
    Future<int> addUsers() async {
    User firstUser = User(
        name: userModel.userOne.name,
        location: userModel.userOne.location,
    );
    User secondUser = User(
        name: userProvider.userTwo.name,
        location: userProvider.userTwo.location,
    );
    List<User> listOfUsers = [
        firstUser,
        secondUser,
    ];
    return await handler.insertUser(listOfUsers);
    }

    return ScopedModelDescendant<UserModel>(
    builder: (context, child, model) => Scaffold(
        appBar: customAppBar(title),
        body: FutureBuilder(
        future: handler.retrieveUsers(),
        builder: (BuildContext context, AsyncSnapshot<Lis\
t<User>> snapshot) {
            if (snapshot.hasData) {
            return ListView.builder(
                itemCount: snapshot.data?.length,
                itemBuilder: (BuildContext context, int i\
ndex) {
                return Column(
                    children: [
                    Card(
                        child: ListTile(
                        key: ValueKey<int>(snapshot.data!\
[index].id!),
                        contentPadding: const EdgeInsets.\
all(8.0),
                        title: Text(
                            snapshot.data![index].name,
                            style: const TextStyle(
                            fontSize: 30,
                            color: Colors.red,
                            ),
                        ),
                        subtitle: Text(
                            snapshot.data![index].locatio\
n,
                            style: const TextStyle(
                            fontSize: 20,
                            color: Colors.blue,
                            ),
                        ),
                        ),
                        elevation: 20,
                    ),
                    TextButton(
                        onPressed: () {
                        handler.initializeDB().whenComple\
te(() async {
                            await addUsers();
                        });
                        model.addingUsers();
                        },
                        child: const Text(
                        'Add Users by Scoped Model',
                        style: TextStyle(
                            fontSize: 20,
                            fontWeight: FontWeight.w600,
                        ),
                        ),
                    )
                    ],
                );
                },
            );
            } else {
            return const Center(child: CircularProgressIn\
dicator());
            }
        },
        ),
        floatingActionButton: FloatingActionButton.extend\
ed(
        onPressed: () {
            handler.initializeDB().whenComplete(() async {
            await addUsers();
            });
            userProvider.addingUsers();
        },
        label: const Text(
            'Add Users by Provider',
            style: TextStyle(
            fontSize: 25,
            fontWeight: FontWeight.bold,
            ),
        ),
        ),
    ),
    );
}

AppBar customAppBar(String title) {
    return AppBar(
    centerTitle: true,
    //backgroundColor: Colors.grey[400],
    flexibleSpace: Container(
        decoration: const BoxDecoration(
        gradient: LinearGradient(
            colors: [
            Colors.pink,
            Colors.grey,
            ],
            begin: Alignment.topRight,
            end: Alignment.bottomRight,
        ),
        ),
    ),
    elevation: 20,
    titleSpacing: 80,
    title: Text(
        title,
        textAlign: TextAlign.left,
    ),
    );
}
}
```

For full code please visit the respective GitHub Repository given at the end of this section.

However, we could have separated them. And that is what we have done for the second Flutter Application.

As a result, the FutureBuilder and our Flutter Application works perfectly.

How do you reset the future builder Flutter?

This time, we will add the first user by using the Scoped Model.

If we press the Button below it will add the user and it also displays the name and location of User. Moreover, below the display of the newly created User object, we can see the “Navigation Button”.

Let us see the code first. After that we will click the “Next Page” Button to add another User by the Provider.

Firstly, we see the User Model class that extends the Model class from the Scoped Model dependency.

```
import 'package:scoped_model/scoped_model.dart';

import 'user.dart';

class UserModel extends Model {
User _userModel = User(name: 'John Smith', location: 'Bac\
k East');
User get userModel => _userModel;

void addingUsers() {
    _userModel = userModel;

    notifyListeners();
}
}
```

Secondly we will see the FutureBuilder widget that uses the Dependent Scoped User Model to add the Users.

```
import 'package:flutter/material.dart';

import 'package:scoped_model/scoped_model.dart';

import '/model/user_model.dart';

import '/model/database_handler.dart';
import '/model/user.dart';
import 'provider_home_page.dart';

class ModelHomePage extends StatelessWidget {
const ModelHomePage({Key? key}) : super(key: key);

static const String title = 'Adding by Scoped Model';

@override
```

```
Widget build(BuildContext context) {
    final userModel = ScopedModel.of<UserModel>(context);

    final handler = DatabaseHandler();
    Future<int> addUsers() async {
    User firstUser = User(
        name: userModel.userModel.name,
        location: userModel.userModel.location,
    );

    List<User> listOfUsers = [
        firstUser,
    ];
    return await handler.insertUser(listOfUsers);
    }

    return ScopedModelDescendant<UserModel>(
    builder: (context, child, model) => Scaffold(
        appBar: customAppBar(title),
        body: FutureBuilder(
        future: handler.retrieveUsers(),
        builder: (BuildContext context, AsyncSnapshot<Lis\
t<User>> snapshot) {
            if (snapshot.hasData) {
            return ListView.builder(
                itemCount: snapshot.data?.length,
                itemBuilder: (BuildContext context, int i\
ndex) {
                return Column(
                    children: [
                    Card(
                        child: ListTile(
                        key: ValueKey<int>(snapshot.data!\
[index].id!),
                        contentPadding: const EdgeInsets.\
all(8.0),
                        title: Text(
                            snapshot.data![index].name,
                            style: const TextStyle(
                            fontSize: 30,
                            color: Colors.red,
                            ),
                        ),
                        subtitle: Text(
                            snapshot.data![index].locatio\
n,
                            style: const TextStyle(
                            fontSize: 20,
                            color: Colors.blue,
                            ),
                        ),
                        ),
                        elevation: 20,
                    ),
                    TextButton(
                        onPressed: () {
                        Navigator.push(
                            context,
                            MaterialPageRoute(
                            builder: (context) => const P\
roviderHomePage(),
                            ),
                        );
                        },
                        child: const Text(
                        'Next Page, Add by Provider',
                        style: TextStyle(
                            fontSize: 20,
                            fontWeight: FontWeight.w600,
                        ),
                        ),
                    )
                    ],
                );
                },
            );
            } else {
            return const Center(child: CircularProgressIn\
dicator());
            }
        },
        ),
        floatingActionButton: FloatingActionButton.extend\
ed(
        onPressed: () {
            handler.initializeDB().whenComplete(() async {
            await addUsers();
            });
            userModel.addingUsers();
        },
        label: const Text(
            'Add Users by Model',
            style: TextStyle(
            fontSize: 25,
            fontWeight: FontWeight.bold,
            ),
        ),
        ),
    ),
    );
}

AppBar customAppBar(String title) {
    return AppBar(
    centerTitle: true,
    //backgroundColor: Colors.grey[400],
    flexibleSpace: Container(
        decoration: const BoxDecoration(
        gradient: LinearGradient(
            colors: [
            Colors.pink,
            Colors.grey,
            ],
            begin: Alignment.topRight,
            end: Alignment.bottomRight,
        ),
        ),
    ),
    elevation: 20,
    titleSpacing: 80,
    title: Text(
        title,
        textAlign: TextAlign.left,
    ),
    );
}
}
```

Next, we go the next page where we can add the second User by using the Provider.

The next page correctly shows that one User has already been added to the SQLite database.

Now, when we add the Button “Add Users by Provider”, another user is added. Moreover, two users are displayed correctly on the UI.

Let us take a look at the code now.

Firstly we will take a look at the User Provider class that extends the ChangeNotifier. After that it notifies the listeners.

```
import 'package:flutter/material.dart';

import 'user.dart';

class UserProvider with ChangeNotifier {
User _userProvider = User(name: 'Json Web', location: 'De\
troit');
User get userProvider => _userProvider;

void addingUsers() {
    _userProvider = userProvider;

    notifyListeners();
}
}
```

Secondly we will see the FutureBuilder that rebuilds when the User object changes its state.

We have created another FutureBuilder for this page, so that we can add users by Provider.

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';

import '/model/user_prvider.dart';
import '/model/database_handler.dart';
import '/model/user.dart';

class ProviderHomePage extends StatelessWidget {
const ProviderHomePage({Key? key}) : super(key: key);

static const String title = 'Adding By Provider';

@override
Widget build(BuildContext context) {
    final userProvider = Provider.of<UserProvider>(contex\
t);

    final handler = DatabaseHandler();
    Future<int> addUsers() async {
    User secondUser = User(
        name: userProvider.userProvider.name,
        location: userProvider.userProvider.location,
    );
    List<User> listOfUsers = [
        secondUser,
    ];
    return await handler.insertUser(listOfUsers);
    }

    return Scaffold(
    appBar: AppBar(
        title: const Text(title),
    ),
    body: FutureBuilder(
        future: handler.retrieveUsers(),
        builder: (BuildContext context, AsyncSnapshot<Lis\
t<User>> snapshot) {
        if (snapshot.hasData) {
            return ListView.builder(
            itemCount: snapshot.data?.length,
            itemBuilder: (BuildContext context, int index\
) {
                return Column(
                children: [
                    Card(
                    child: ListTile(
                        key: ValueKey<int>(snapshot.data!\
[index].id!),
                        contentPadding: const EdgeInsets.\
all(8.0),
                        title: Text(
                        snapshot.data![index].name,
                        style: const TextStyle(
                            fontSize: 30,
                            color: Colors.red,
                        ),
                        ),
                        subtitle: Text(
                        snapshot.data![index].location,
                        style: const TextStyle(
                            fontSize: 20,
                            color: Colors.blue,
                        ),
                        ),
                    ),
                    elevation: 20,
                    ),
                ],
                );
            },
            );
        } else {
            return const Center(child: CircularProgressIn\
dicator());
        }
        },
    ),
    floatingActionButton: FloatingActionButton.extended(
        onPressed: () {
        handler.initializeDB().whenComplete(() async {
            await addUsers();
        });
        userProvider.addingUsers();
        },
        label: const Text(
        'Add Users by Provider',
        style: TextStyle(
            fontSize: 25,
            fontWeight: FontWeight.bold,
        ),
        ),
    ),
    );
}
}
```

In the second Flutter Application we have successfully used The Scoped Model and the Provider. However, we have implemented the FutureBuilder widget separately.

For the full code snippet for this Flutter Application, please visit the respective GitHub Repository.

- For full code snippet please visit the respective GitHub Repository - [31]

[31]https://github.com/sanjibsinha/provider_et_sqflite/tree/experiment-provider-scoped-one

9. NoWar App Challenge

We are going to build a NoWar Flutter App. While we build this app, we will discuss a few important list-concepts in Flutter and Dart. We will use various list methods. And one of them is the list.asMap() method.

Firstly, let us see how the home page of this NoWar app will look like.

Secondly, we will discuss the list method that we have used in the home page.

Finally, we will build the whole app.

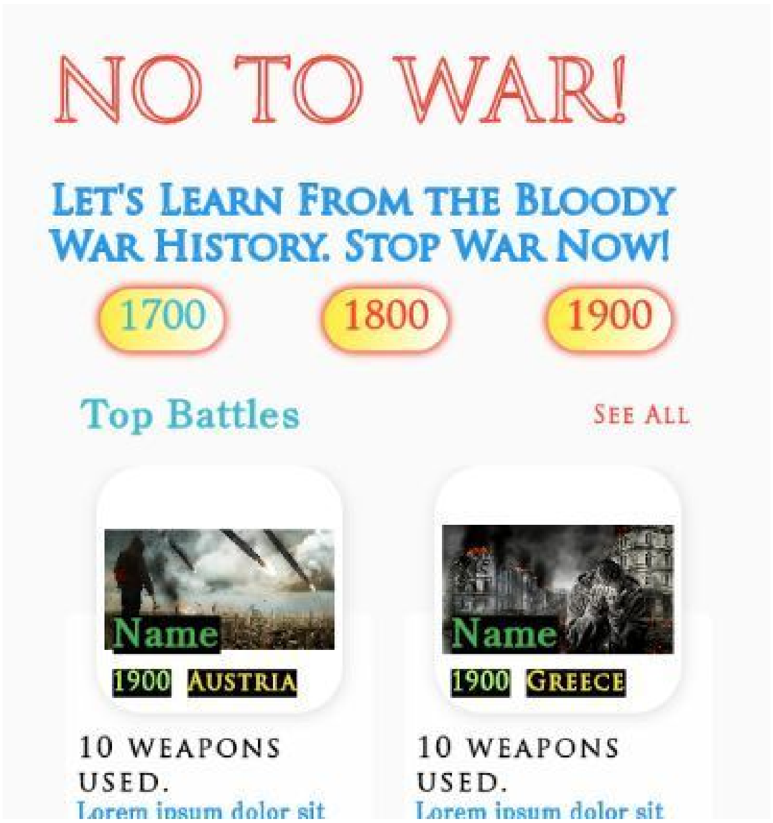


Figure 9.1 – Home Page

This is the home page which we can scroll down to see other parts. As we scroll down, we will find more topics that we need to build.

In addition, there will be other pages as well.

Therefore, as a whole, we will use the 'routes' property of the MaterialApp widget in a correct way.

Why?

Because we need to pass correct data. Besides, we will also design those pages.

The middle of the page will look as follows as we scroll down.

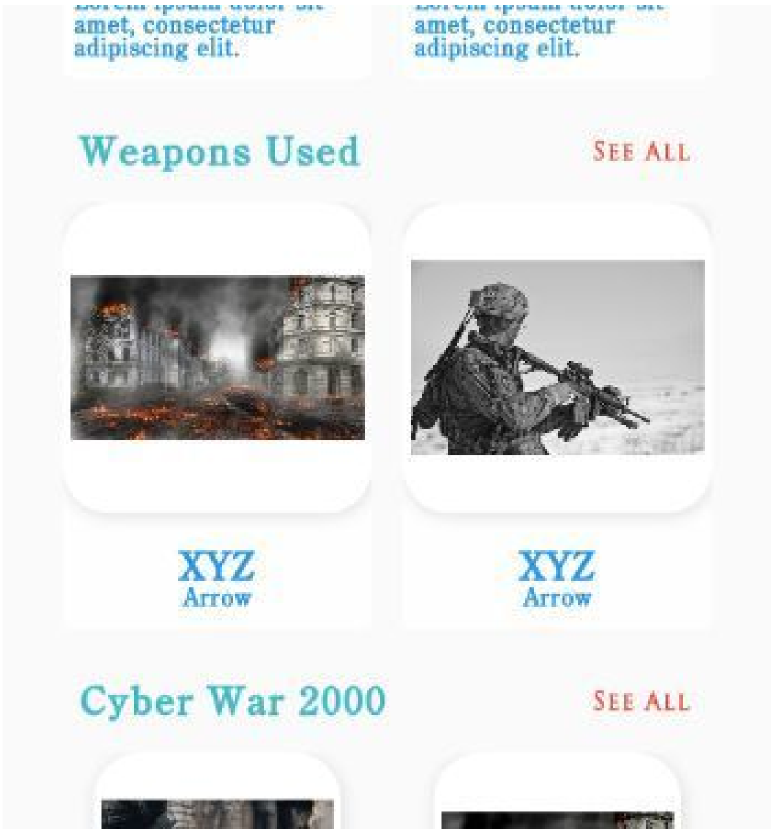

amet, consectetur adipiscing elit.
amet, consectetur adipiscing elit.
Weapons Used
SEE ALL
XYZ
Arrow
XYZ
Arrow
Cyber War 2000
SEE ALL

Figure 9.2 – Home Page Middle part

In the middle part, we need a route to the page where we will display what kind of weapons used in the last three centuries.

In addition, as our civilisation progresses, the style of war has changed a lot. Therefore, we have added a page that discuses cyber warfare.

That is the lower part of our NoWar app.

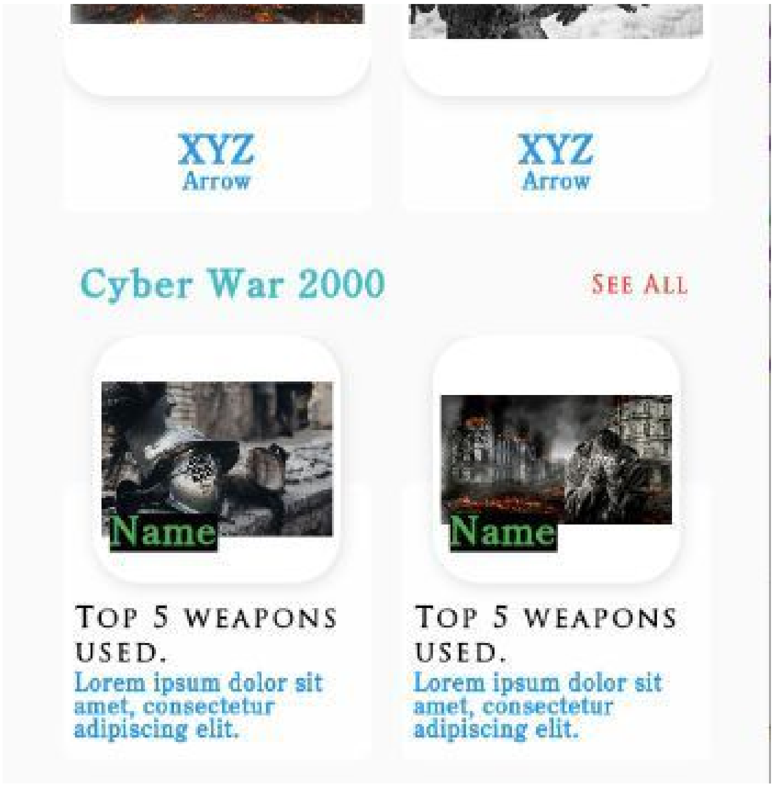

XYZ
Arrow
XYZ
Arrow
Cyber War 2000
SEE ALL
Name
Name
TOP 5 WEAPONS USED.
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
TOP 5 WEAPONS USED.
Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Figure 9.3 – Home Page lower part

What is Dart list asMap method?

In the upper part of the home page, we have three icons. These icons represent last three centuries.

Of course, we could have hard coded those icons one after another. But we will not do that.

Instead, we create a list of three icons and an index property, as follows.

```
int _selectedIndex = 0;
final List _icons = [
    '1700',
    '1800',
    '1900',
];
```

After that, we will map this list one after another so that the key represents the index, and the values are the elements at each index.

Let us a define it inside a Widget method like below.

```
Widget _buildIcons(int index) {
    return GestureDetector(
    onTap: () {
        setState(() {
        _selectedIndex = index;
        });
        Navigator.push(
        context,
        MaterialPageRoute(builder: (context) => const All\
Wars()),
        );
    },
    child: Container(
        margin: const EdgeInsets.all(10.0),
        width: 80.0,
        height: 40.0,
        alignment: Alignment.center,
        decoration: const BoxDecoration(
        borderRadius: BorderRadius.all(Radius.circular(30\
.00)),
        boxShadow: [
            BoxShadow(
            color: Colors.red,
            blurRadius: 4.00,
            spreadRadius: 2.00,
            ),
        ],
        gradient: LinearGradient(
            begin: Alignment.centerLeft,
            end: Alignment.centerRight,
            colors: [
            Colors.yellow,
            Colors.white,
            ],
        ),
        ),
        child: Text(
        '${_icons[index]}',
        textAlign: TextAlign.center,
        style: TextStyle(
            fontSize: 25.00,
            fontFamily: 'Trajan Pro',
            fontWeight: FontWeight.bold,
            color: _selectedIndex == index
                ? Theme.of(context).primaryColor
                : Colors.red,
        ),
        ),
    ),
    );
}
```

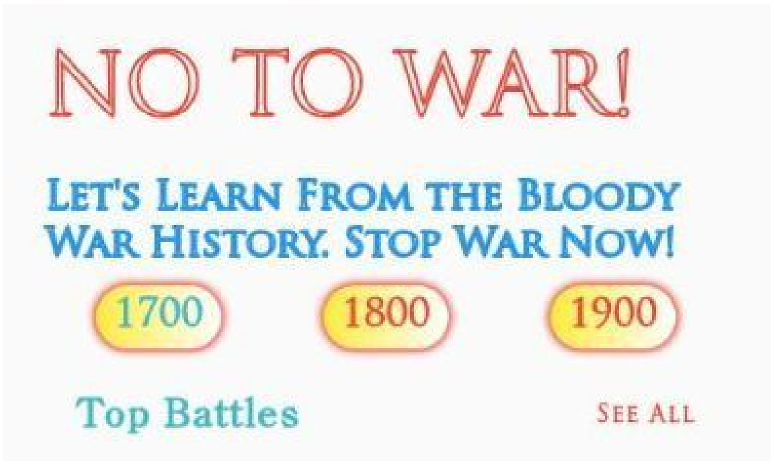
The above method will pass the index as key. Besides, it returns a Gesture Detector widget. As a result we can tap any of the icon, and reach a page.

Now, inside, the home page widget, we are going to use the Dart list asMap() method.

```
@override
Widget build(BuildContext context) {
    return Scaffold(
    body: SafeArea(
        child: ListView(
        padding: const EdgeInsets.all(32),
        children: <Widget>[
            Text(
            'NO TO WAR!',
            style: TextStyle(
                fontSize: 55.0,
                fontWeight: FontWeight.normal,
                fontFamily: 'Trajan Pro',
                foreground: Paint()
                    ..color = Colors.red
                    ..strokeWidth = 2.0
                    ..style = PaintingStyle.stroke),
            ),
            const SizedBox(
            height: 20.00,
            ),
            Text(
            'Let\'s Learn From the Bloody War History. St\
op War Now!',
            style: TextStyle(
                fontSize: 25.0,
                fontWeight: FontWeight.normal,
                fontFamily: 'Trajan Pro',
                foreground: Paint()
                    ..color = Colors.blue
                    ..strokeWidth = 2.0
                    ..style = PaintingStyle.stroke),
            ),
            Row(
            mainAxisAlignment: MainAxisAlignment.spaceAro\
und,
            children: _icons
                .asMap()
                .entries
                .map((MapEntry map) => _buildIcons(map.ke\
y))
                .toList(),
            ),
            const SizedBox(
            height: 20.00,
            ),
            const TopBattleController(),
            const SizedBox(
            height: 20.00,
            ),
            const WeaponController(),
            const SizedBox(
            height: 20.00,
            ),
            const CyberController(),
        ],
        ),
    ),
    );
}
}
```

As a result, we can display the last three centuries in a Row.

Map through a list in Flutter


NO TO WAR!

LET'S LEARN FROM THE BLOODY WAR HISTORY. STOP WAR NOW!

Figure 9.4 – List of Icons in a Row Widget

In this section we will concentrate how we have used the List asMap() method. Certainly, we can map a list in Flutter other way also.

Therefore, let us a take a close look at that part now.

```
Row(
              mainAxisAlignment: MainAxisAlignment.spaceAro\
und,
              children: _icons
                   .asMap()
                   .entries
                   .map((MapEntry map) => _buildIcons(map.ke\
y))
                   .toList(),
              ),
```

To understand this list method, let's consider a simple example first.

```
void main() {

List<String> words = ['A', 'B', 'C', 'D'];

words.asMap().forEach((index, value) => print(value));

}
// output
A
B
C
D
```

Let's add two more lines to the above code that will clarify the home page code.

```
print(map[0]);   // 'A';
map.keys.toList();
```

Now, let's take a look at the whole code again.

Let us redefine the code in a new way where we can get the key as index, and value as the element at that index position.

```
void main() {

List<String> words = ['A', 'B', 'C', 'D'];

words.asMap().entries.map((MapEntry map) => print('${map.\
key} : ${map.value}')).toList();

}

// output:
0 : A
```

```
1 : B
2 : C
3 : D
```

We have displayed the list of icons in the same way.

Our next task will be to build three separate pages that will represent three centuries.

How to design a Flutter UI or user interface? That is one of the main challenges that very Flutter developer face while building a Flutter App.

For example, a challenge always ignites our passion to develop our skill. Certainly, this is no exception.

We have just started and built a part of the NoWar app. However, we can take it as a challenge to complete and make it a finished Flutter App.

We have already seen the First Step where we have mainly built the home page. In this section, we will build a few other pages. In addition, we will also use some material deign-related Widgets.

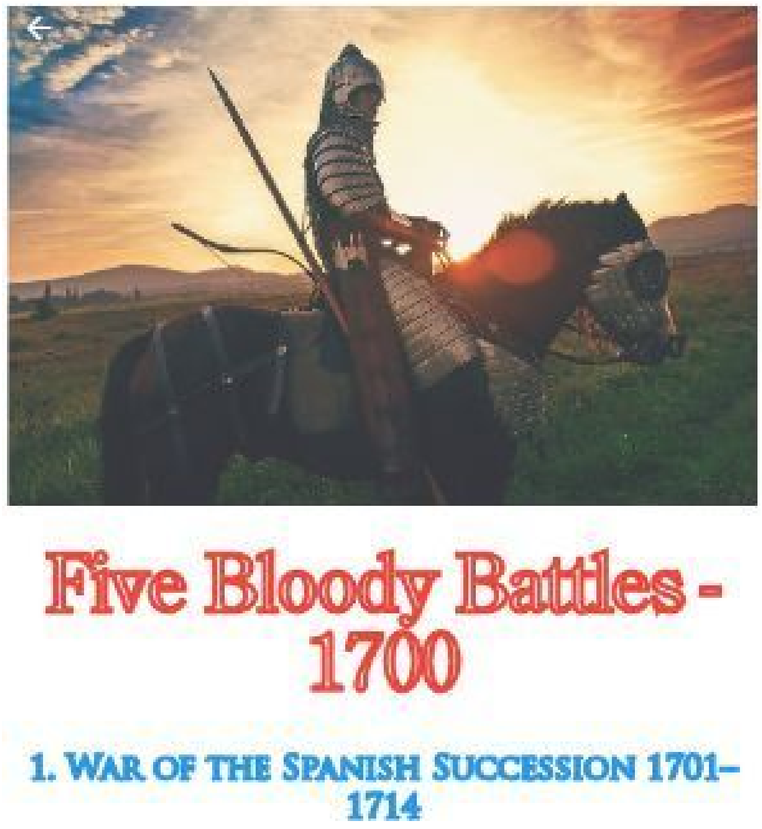
Firstly, let us see the screenshots of the pages that we have finished so far.

Secondly, we will watch the code of the pages where we have used the material design-related widgets.

Finally, I will leave this NoWar App to your disposal to complete as a challenge. I will give the link of the GitHub repository at the end of this section.

Certainly, ending war and bringing peace to this planet is a challenge.

Always.



Five Bloody Battles - 1700

1. WAR OF THE SPANISH SUCCESSION 1701–1714

Figure 9.5 – Changed home page

This is our home page from where we can move to several pages. However, we have defined each section as a class in our model folder, so that we can follow the MVC design.

The plan is simple. We should click each icon to reach a home page. Like the 1700 century icon represents a home page from where we can go to other page that describes a war.

How to design Flutter UI

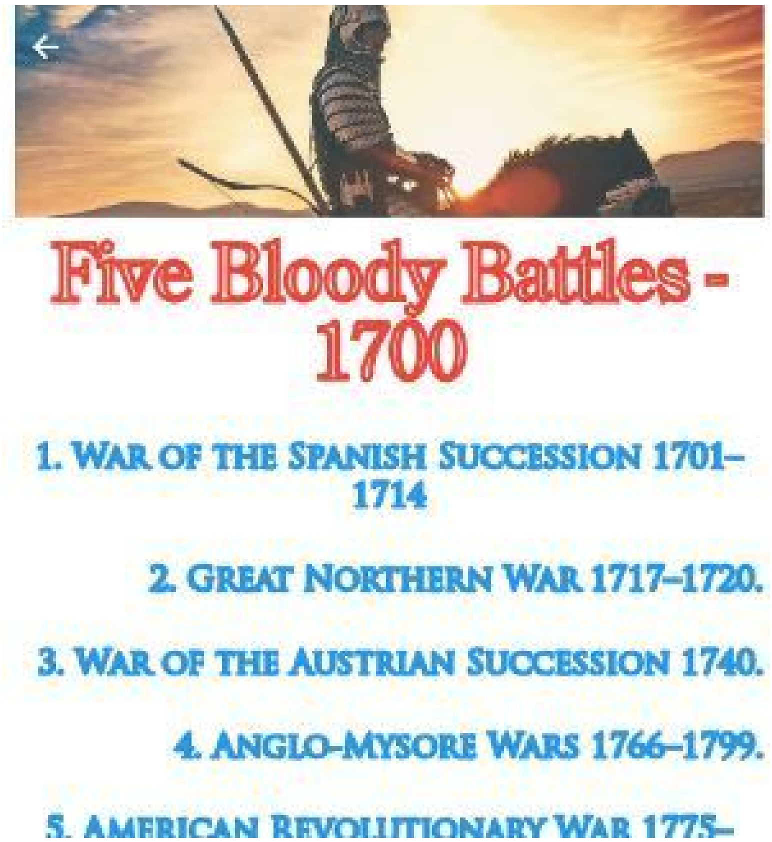
As we will follow the same material design pattern, that will take time to build the pages. But, more or less, we will have to do the same task for several times.

Again from that page, we can go back to the home page, or go to the next page.

As you're guessing, we need to create several pages. In addition, we will link the pages. So our app should not crash while we navigate to other pages, or come back.

That part belongs to the "routes" property of the MaterialApp widget which we have discussed earlier.

Suppose we click the 1700 century icon. As a result, that will take us to the home page where we will describe five links.



Five Bloody Battles - 1700

1. WAR OF THE SPANISH SUCCESSION 1701–1714

2. GREAT NORTHERN WAR 1717–1720.

3. WAR OF THE AUSTRIAN SUCCESSION 1740.

4. ANGLO-MYSORE WARS 1766–1799.

5. AMERICAN REVOLUTIONARY WAR 1775–

Figure 9.6 – 1700 century home page

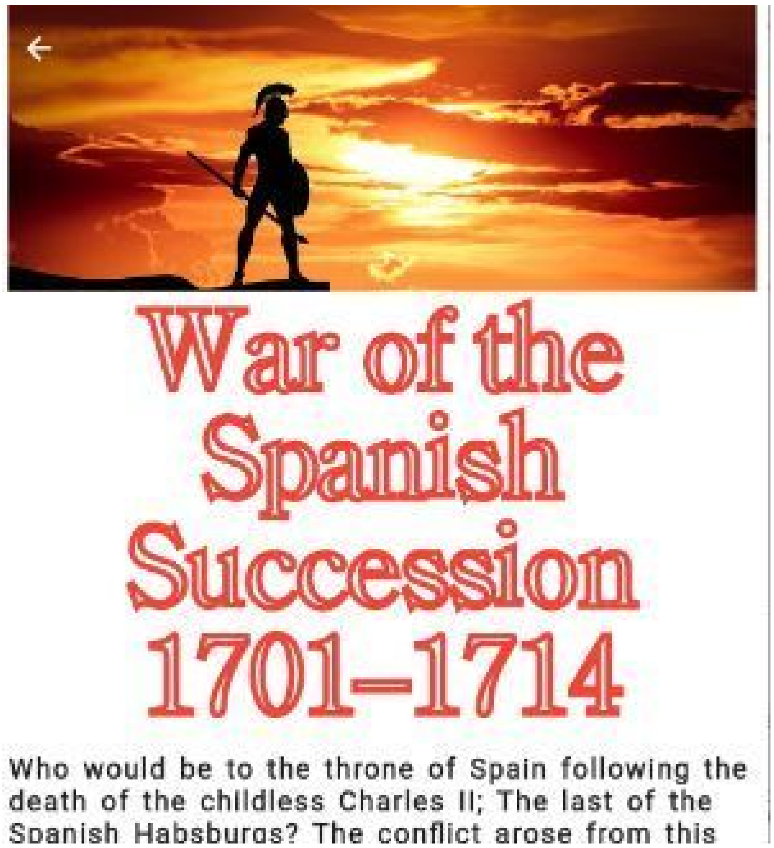
Let us see the code of this page first.

```
import 'package:flutter/material.dart';
import '../../model/seventeen_hundred_wars.dart';
import 'seventeen_first.dart';

class SeventeenHome extends StatefulWidget {
static const routeNname = '/seventen-home';
const SeventeenHome({Key? key}) : super(key: key);

@override
_SeventeenHomeState createState() => _SeventeenHomeState(\
);
}

class _SeventeenHomeState extends State<SeventeenHome> {
List<SeventeenHundredWars> seventeenWars = [
    seventeenHundredWars[0],
    seventeenHundredWars[1],
    seventeenHundredWars[2],
    seventeenHundredWars[3],
    seventeenHundredWars[4],
];

@override
Widget build(BuildContext context) {
    return Scaffold(
    body: CustomScrollView(
        slivers: <Widget>[
        SliverAppBar(
            backgroundColor: Colors.white,
            stretch: true,
            expandedHeight: 350.0,
            flexibleSpace: FlexibleSpaceBar(
            background: Container(
                padding: const EdgeInsets.all(8.0),
                child: Image.network(
                    'https://cdn.pixabay.com/photo/2017/0\
8/01/14/42/knight-2565957_960_720.jpg'),
            ),
            stretchModes: const [
                StretchMode.zoomBackground,
            ],
            ),
        ),
        SliverFixedExtentList(
            itemExtent: 450,
            delegate: SliverChildListDelegate([
            Container(
                color: Colors.white,
                child: Column(
                mainAxisSize: MainAxisSize.max,
                mainAxisAlignment: MainAxisAlignment.spac\
eAround,
                crossAxisAlignment: CrossAxisAlignment.en\
d,
                // text direction does the same thing hor\
izontally
                verticalDirection: VerticalDirection.down,
                children: <Widget>[
                    Padding(
                    padding: const EdgeInsets.all(8.0),
                    child: Text(
                        'Five Bloody Battles - 1700',
                        style: TextStyle(
                            fontSize: 50.0,
                            fontWeight: FontWeight.bold,
                            fontFamily: 'Schuyler',
                            foreground: Paint()
                            ..color = Colors.red
                            ..strokeWidth = 2.0
                            ..style = PaintingStyle.strok\
e),
                            textAlign: TextAlign.center,
                        ),
                        ),
                        GestureDetector(
                        onTap: () {
                            Navigator.push(
                            context,
                            MaterialPageRoute(
                                builder: (context) => const S\
eventeenFirst()),
                            );
                        },
                        child: Padding(
                            padding: const EdgeInsets.all(8.0\
),
                            child: Text(
                            '1. ${seventeenWars[0].name}',
                            style: TextStyle(
                                fontSize: 21.0,
                                fontWeight: FontWeight.bold,
                                fontFamily: 'Trajan Pro',
                                foreground: Paint()
                                    ..color = Colors.blue
                                    ..strokeWidth = 2.0
                                    ..style = PaintingStyle.s\
troke),
                            textAlign: TextAlign.center,
                            ),
                        ),
                        ),
                        Padding(
                        padding: const EdgeInsets.all(8.0),
                        child: Text(
                            '2. ${seventeenWars[1].name}',
                            style: TextStyle(
                                fontSize: 21.0,
                                fontWeight: FontWeight.bold,
                                fontFamily: 'Trajan Pro',
                                foreground: Paint()
                                ..color = Colors.blue
                                ..strokeWidth = 2.0
                                ..style = PaintingStyle.strok\
    e),
                            textAlign: TextAlign.center,
                        ),
                        ),
                        Padding(
                        padding: const EdgeInsets.all(8.0),
                        child: Text(
                            '3. ${seventeenWars[2].name}',
                            style: TextStyle(
                                fontSize: 21.0,
                                fontWeight: FontWeight.bold,
                                fontFamily: 'Trajan Pro',
                                foreground: Paint()
                                ..color = Colors.blue
                                ..strokeWidth = 2.0
                                ..style = PaintingStyle.strok\
    e),
                            textAlign: TextAlign.center,
                        ),
                        ),
                        Padding(
                        padding: const EdgeInsets.all(8.0),
                        child: Text(
                            '4. ${seventeenWars[3].name}',
                            style: TextStyle(
                                fontSize: 21.0,
                                fontWeight: FontWeight.bold,
                                fontFamily: 'Trajan Pro',
                                foreground: Paint()
                                ..color = Colors.blue
                                ..strokeWidth = 2.0
                                ..style = PaintingStyle.strok\
    e),
                            textAlign: TextAlign.center,
                        ),
                        ),
                        Padding(
                        padding: const EdgeInsets.all(8.0),
                        child: Text(
                            '5. ${seventeenWars[4].name}',
                            style: TextStyle(
                                fontSize: 21.0,
                                fontWeight: FontWeight.bold,
                                fontFamily: 'Trajan Pro',
                                foreground: Paint()
                                ..color = Colors.blue
                                ..strokeWidth = 2.0
                                ..style = PaintingStyle.strok\
    e),
                            textAlign: TextAlign.center,
                        ),
                        ),
                    ],
                    ),
                ),
                Container(),
                ]),
            ),
            ],
        ),
        );
    }
```

```
}
```

As we see, we have used the CustomScrollView Widget so that we can scroll down the page.

After that we want that the AppBar section will shrink or resize as we scroll down as follows.



Who would be to the throne of Spain following the death of the childless Charles II; The last of the Spanish Habsburgs? The conflict arose from this

Figure 9.7 – AppBar section will shrink or resize as we scroll down

We can make it happen by using the SliverAppBar Widget. You may read how to collapse the header section.

We follow the same technique in the next page.

Figure 9.8 – SliverAppBar Widget resizes

How to use model class to design Flutter UI?

In the previous section we have seen how to use the List methods to use our model class where we have defined many properties.

Let us see one of the model class from where we have got the images.

```dart
import 'weapon_used.dart';

class SeventeenHundredWars {
String imageUrl;
int centuries;
String name;
String place;
int howManyDied;
String description;
List<WeaponUsed> weapons;
String summary;

SeventeenHundredWars({
    required this.imageUrl,
    required this.centuries,
    required this.name,
    required this.place,
    required this.howManyDied,
    required this.description,
    required this.weapons,
    required this.summary,
});
}

List<SeventeenHundredWars> seventeenHundredWars = [
SeventeenHundredWars(
    imageUrl:
        'https://cdn.pixabay.com/photo/2016/03/27/07/38/p\
olice-1282330_960_720.jpg',
    centuries: 1700,
    name: 'War of the Spanish Succession 1701–1714',
    place: 'Europe',
    howManyDied: 100000,
    description:
        'Who would be to the throne of Spain following th\
e death of the childless Charles II;'
        ' The last of the Spanish Habsburgs?'
        ' The conflict arose from this question. Although\
 three principal claimants '
        'England, the Dutch Republic, and France signed a\
 treaty of partition in October 1698, it did'
        ' not last. A major conflict arose that centered \
around one question - who would occupy'
        ' most places. An anti-French alliance was formed\
 (September 7, 1701) by England, the Dutch Republic, and \
the emperor Leopold. '
        ' They were later joined by Prussia, Hanover, oth\
er German states, and Portugal.'
        ' On the other side, The electors of '
        ' Bavaria and Cologne and the dukes of Mantua and\
 Savoy allied themselves with France, although Savoy swit\
ched sides in 1703.'
        'John Churchill, duke of Marlborough, played the \
leading role in Queen Anne’s government and on the battle\
field until his fall in 1711. '
```

```
        ' He was ably supported on the battlefield by the\
 imperial general Prince Eugene of Savoy.'
        ' In Britain the enemies of Marlborough won influ\
ence with the queen and had him removed from command on D\
ecember 31, 1711. '
        ' With the collapse of the alliance, peace negoti\
ations began in 1712. '
        ' Because of the conflicts of interest between th\
e former allies, each dealt separately with France.',
    weapons: weapons,
    summary: 'Treaties of Utrecht marked the rise of the \
power of Britain',
),
SeventeenHundredWars(
    imageUrl:
        'https://cdn.pixabay.com/photo/2014/10/02/06/34/w\
ar-469503_960_720.jpg',
    centuries: 1700,
    name: 'Great Northern War 1717-1720.',
    place: 'Europe',
    howManyDied: 10000,
    description:
        'Lorem ipsum dolor sit amet, consectetur adipisci\
ng elit. Praesent vestibulum metus ac quam laoreet accums\
an. Sed quis ultrices massa, quis elementum nunc. Nam a m\
assa varius lacus suscipit fringilla.',
    weapons: weapons,
    summary: 'Lorem ipsum dolor sit amet, consectetur adi\
piscing elit.',
),
SeventeenHundredWars(
    imageUrl:
        'https://cdn.pixabay.com/photo/2017/11/08/12/04/w\
ar-2930223_960_720.jpg',
    centuries: 1700,
    name: 'War of the Austrian Succession 1740.',
    place: 'Europe',
    howManyDied: 15000,
    description:
        'Lorem ipsum dolor sit amet, consectetur adipisci\
ng elit. Nulla odio ipsum, rhoncus id libero id, pretium \
congue sem. Nunc vitae ultricies justo. In ac nunc a ligu\
la lobortis mattis sed ut ex. Etiam blandit ante sed lacu\
s ullamcorper pulvinar. Ut egestas massa a egestas accums\
an. Etiam eu velit ornare, consectetur urna quis, cursus \
ex. Suspendisse et ipsum mauris. Praesent vestibulum metu\
s ac quam laoreet accumsan. Sed quis ultrices massa, quis\
 elementum nunc. Nam a massa varius lacus suscipit fringi\
lla.'
        'Nulla ullamcorper euismod dui sit amet elementum\
. Suspendisse dapibus eu tellus eu placerat. Sed sit amet\
 nisi ac lectus maximus convallis. Class aptent taciti so\
ciosqu ad litora torquent per conubia nostra, per incepto\
s himenaeos. Aliquam erat volutpat. Praesent vitae lacus \
ac dui aliquam vehicula et sed diam. In eget interdum era\
t. Donec vel ex quis mi ornare vulputate ut sit amet puru\
s. Curabitur aliquet ornare turpis, sed luctus orci ultri\
ces nec. Mauris vestibulum euismod arcu, eu mollis nulla \
fermentum eu. Donec sit amet nulla leo. Nam vitae justo n\
ec magna egestas venenatis. Sed ut ipsum fermentum, ferme\
ntum est nec, laoreet mauris. Duis et suscipit libero, ne\
c porttitor erat. Proin vel sem sollicitudin, hendrerit m\
auris ac, accumsan eros.'
        'Proin non egestas velit, in pharetra neque. Etia\
m a sapien in nunc cursus pharetra. Integer mi nunc, inte\
rdum volutpat mi molestie, eleifend consequat lorem. Etia\
m mattis in libero eget fringilla. Mauris aliquet libero \
non massa blandit auctor. Duis vestibulum velit dignissim\
 nunc ullamcorper porttitor. Ut vestibulum, neque id blan\
dit eleifend, lectus turpis consectetur nunc, id viverra \
mauris lacus vitae ipsum. Aliquam quis finibus risus. ',
    weapons: weapons,
    summary: 'Lorem ipsum dolor sit amet, consectetur adi\
piscing elit.',
),
SeventeenHundredWars(
    imageUrl:
        'https://cdn.pixabay.com/photo/2017/11/24/05/38/j\
et-2974131_960_720.jpg',
    centuries: 1700,
    name: 'Anglo-Mysore Wars 1766–1799.',
    place: 'India',
    howManyDied: 20000,
    description:
        'Lorem ipsum dolor sit amet, consectetur adipisci\
ng elit. Nulla odio ipsum, rhoncus id libero id, pretium \
congue sem. Nunc vitae ultricies justo. In ac nunc a ligu\
la lobortis mattis sed ut ex. Etiam blandit ante sed lacu\
s ullamcorper pulvinar. Ut egestas massa a egestas accums\
an. Etiam eu velit ornare, consectetur urna quis, cursus \
ex. Suspendisse et ipsum mauris. Praesent vestibulum metu\
s ac quam laoreet accumsan. Sed quis ultrices massa, quis\
 elementum nunc. Nam a massa varius lacus suscipit fringi\
lla.'
        'Nulla ullamcorper euismod dui sit amet elementum\
. Suspendisse dapibus eu tellus eu placerat. Sed sit amet\
 nisi ac lectus maximus convallis. Class aptent taciti so\
ciosqu ad litora torquent per conubia nostra, per incepto\
s himenaeos. Aliquam erat volutpat. Praesent vitae lacus \
ac dui aliquam vehicula et sed diam. In eget interdum era\
t. Donec vel ex quis mi ornare vulputate ut sit amet puru\
s. Curabitur aliquet ornare turpis, sed luctus orci ultri\
ces nec. Mauris vestibulum euismod arcu, eu mollis nulla \
fermentum eu. Donec sit amet nulla leo. Nam vitae justo n\
ec magna egestas venenatis. Sed ut ipsum fermentum, ferme\
ntum est nec, laoreet mauris. Duis et suscipit libero, ne\
c porttitor erat. Proin vel sem sollicitudin, hendrerit m\
auris ac, accumsan eros.'
        'Proin non egestas velit, in pharetra neque. Etia\
m a sapien in nunc cursus pharetra. Integer mi nunc, inte\
rdum volutpat mi molestie, eleifend consequat lorem. Etia\
m mattis in libero eget fringilla. Mauris aliquet libero \
non massa blandit auctor. Duis vestibulum velit dignissim\
 nunc ullamcorper porttitor. Ut vestibulum, neque id blan\
dit eleifend, lectus turpis consectetur nunc, id viverra \
mauris lacus vitae ipsum. Aliquam quis finibus risus. ',
    weapons: weapons,
    summary: 'Lorem ipsum dolor sit amet, consectetur adi\
piscing elit.',
),
SeventeenHundredWars(
    imageUrl:
        'https://cdn.pixabay.com/photo/2014/06/21/21/57/a\
pocalyptic-374208_960_720.jpg',
    centuries: 1700,
    name: 'American Revolutionary War 1775-1783.',
    place: 'US',
    howManyDied: 50000,
    description:
        'Lorem ipsum dolor sit amet, consectetur adipisci\
ng elit. Nulla odio ipsum, rhoncus id libero id, pretium \
congue sem. Nunc vitae ultricies justo. In ac nunc a ligu\
la lobortis mattis sed ut ex. Etiam blandit ante sed lacu\
s ullamcorper pulvinar. Ut egestas massa a egestas accums\
an. Etiam eu velit ornare, consectetur urna quis, cursus \
ex. Suspendisse et ipsum mauris. Praesent vestibulum metu\
s ac quam laoreet accumsan. Sed quis ultrices massa, quis\
 elementum nunc. Nam a massa varius lacus suscipit fringi\
lla.'
        'Nulla ullamcorper euismod dui sit amet elementum\
. Suspendisse dapibus eu tellus eu placerat. Sed sit amet\
 nisi ac lectus maximus convallis. Class aptent taciti so\
ciosqu ad litora torquent per conubia nostra, per incepto\
s himenaeos. Aliquam erat volutpat. Praesent vitae lacus \
ac dui aliquam vehicula et sed diam. In eget interdum era\
t. Donec vel ex quis mi ornare vulputate ut sit amet puru\
s. Curabitur aliquet ornare turpis, sed luctus orci ultri\
ces nec. Mauris vestibulum euismod arcu, eu mollis nulla \
fermentum eu. Donec sit amet nulla leo. Nam vitae justo n\
ec magna egestas venenatis. Sed ut ipsum fermentum, ferme\
ntum est nec, laoreet mauris. Duis et suscipit libero, ne\
c porttitor erat. Proin vel sem sollicitudin, hendrerit m\
auris ac, accumsan eros.'
        'Proin non egestas velit, in pharetra neque. Etia\
m a sapien in nunc cursus pharetra. Integer mi nunc, inte\
rdum volutpat mi molestie, eleifend consequat lorem. Etia\
m mattis in libero eget fringilla. Mauris aliquet libero \
non massa blandit auctor. Duis vestibulum velit dignissim\
 nunc ullamcorper porttitor. Ut vestibulum, neque id blan\
dit eleifend, lectus turpis consectetur nunc, id viverra \
mauris lacus vitae ipsum. Aliquam quis finibus risus. ',
    weapons: weapons,
    summary: 'Lorem ipsum dolor sit amet, consectetur adi\
piscing elit.',
),
];
```

Now you can write the content or you can use the dummy content as we see above.

To complete please clone the GitHub repository, in your local machine, and develop.

At the end of the project you will learn how to design a Flutter UI.

- For full code snippet please visit the respective GitHub Repository - [32]

[32]https://github.com/sanjibsinha/no_war

10. How to build a Exchange Rate App

For loop Flutter: PriceTracker App – Step 1

Can we use for loop in Flutter? If we can, then how we can use for loop in Flutter? We're going to see that in a minute. In addition, we will build a Price Tracker Flutter App which will use the for loop.

Because of that reason, we will first check how we can use for loop in Flutter to build a custom Drop down menu Item.

Why do we use for loop?

The first and foremost reason is to to repeat a specific block of code a known number of times.

As a result, we can start from 0 and stop at 4. And, in addition, we can print each number.

Consider the code below.

```
void main() {
for (int i = 0; i < 5; i++) {
    print('hello ${i + 1}');
}
}

/**
hello 1
hello 2
hello 3
hello 4
hello 5
*
*
*/
```

But, in Dart, and Flutter for loop can play another role.

What is that role?

The role is to iterate over the list of items. For that reason, we can use the for-in loop.

For example, consider the code as follows where we have used the for-in loop to iterate over a list of items.

```
void main() {

List<String> firstNameList = ['John', 'Json', 'Dracula', \
'Othelo'];

List<String> secndNameList = ['Hamlet', 'Json', 'Hegemoto\
', 'Piku'];

checkNames(firstNameList); // You have 2 number of same n\
ames.
checkNames(secndNameList); // You have 1 number of same n\
ames.

}

List<String> listOfNames = ['Macbeth', 'Othelo', 'Hamlet'\
, 'John', 'Shakespeare'];

void checkNames(List<String> nameList) {

int name = 0;

for(String names in nameList){
    for(String matchingName in listOfNames) {
    if(names == matchingName){
        name++;
    }

    }
}

print('You have $name number of same names.');
}
```

The above code shows how we can read the items by using the for-in loop. After that, we can also check whether a common name exists or not.

The same way, we can create a drop down menu item widget using a constant list of items.

Since we are going to display the exchange rate of US dollar in different currency, we should display the list of currency first.

Let us see how it looks like.

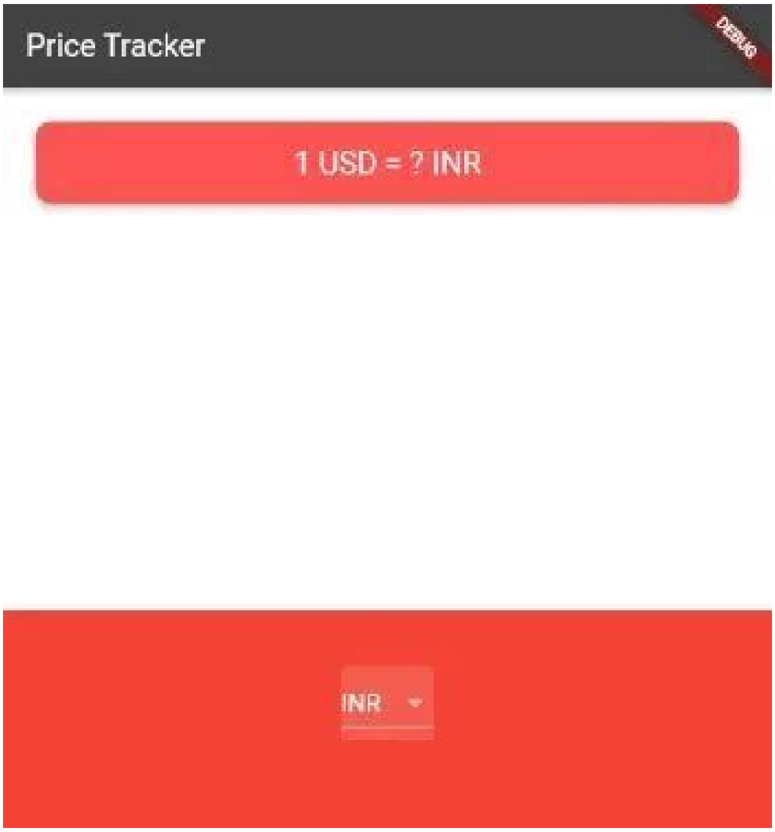


Figure 10.1 – For loop Flutter - PriceTracker App first

Firstly, to show a list of currency like above we should create a list of currencies first.

We will keep that file in our model folder.

```
const List<String> currencyList = [
'INR',
'JPY',
'MXN',
'NOK',
'NZD',
'PLN',
'RON',
'BRL',
'CAD',
'CNY',
'EUR',
'GBP',
'HKD',
'IDR',
'ILS',
'RUB',
];
```

After that, we will create a custom drop down button method.

Why?

Because this method will keep the selected currency in a setState() method as a value that we can display.

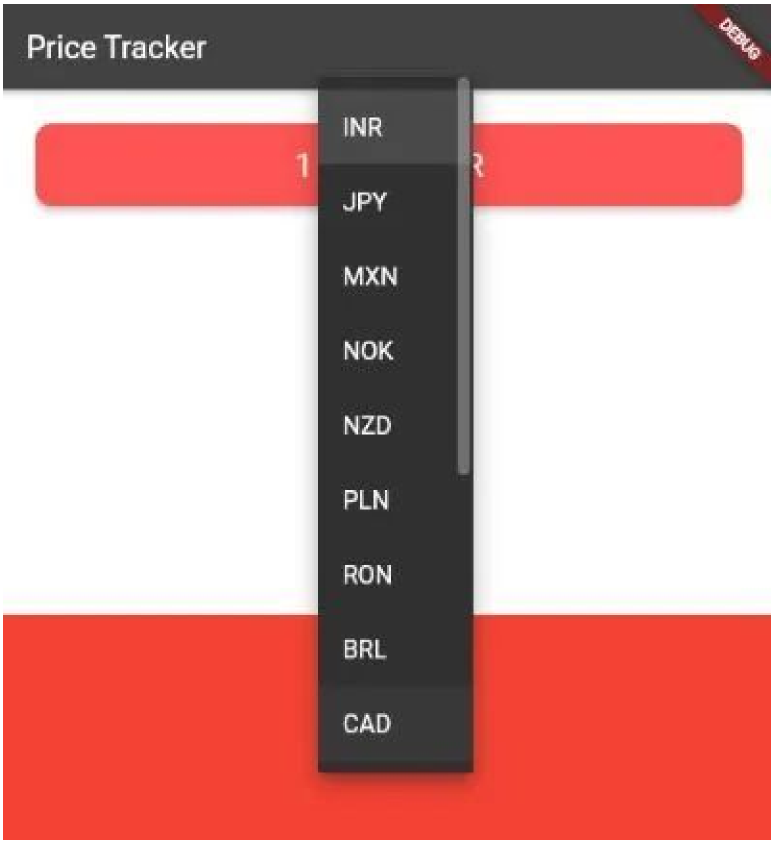


Figure 10.2 – For loop Flutter - PriceTracker App second

We can see that the list starts with INR or Indian Rupees.

But at the same time we can choose any currency from the above list.

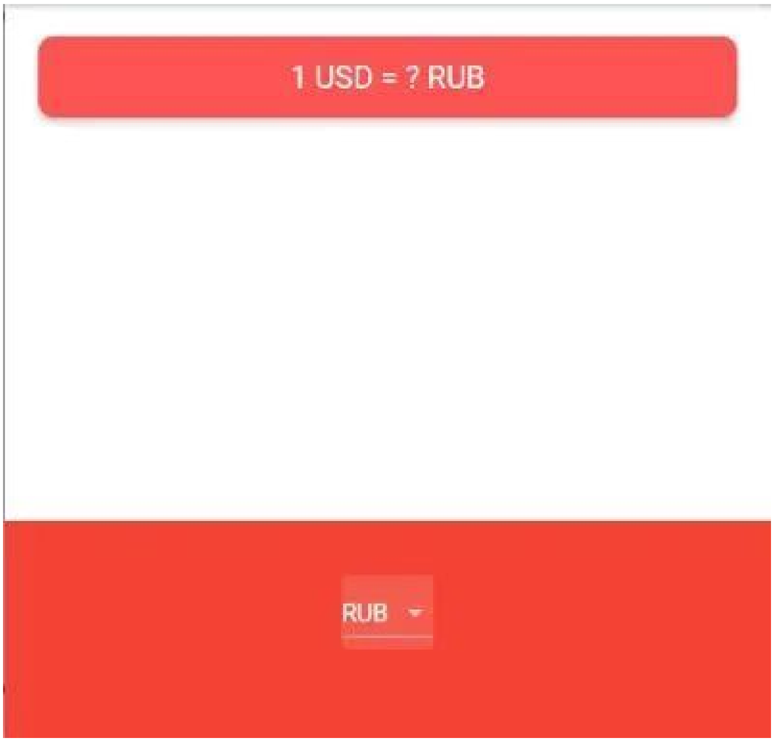


Figure 10.3 – For loop Flutter - PriceTracker App third

As the above image shows, we have chosen RUB or ruble.

How for loop in Flutter works?

It works just like any for-in loop. We iterate over the list of currencies that we have kept in the model folder.

```
DropdownButton<String> selectDropDownList() {
    List<DropdownMenuItem<String>> dropDownItems = [];
    for (String currency in currencyList) {
    var newItem = DropdownMenuItem(
        child: Text(currency),
        value: currency,
    );
    dropDownItems.add(newItem);
    }

    return DropdownButton<String>(
    value: selectedCurrency,
    items: dropDownItems,
    onChanged: (value) {
        setState(() {
        selectedCurrency = value!;
        });
    },
    );
}
```

And, lastly we call this method in the bottom Container Widget.

As a result, when we select any currency, it will display the exchange rate.

```
import 'package:flutter/material.dart';
import '../model/usd_data.dart';

class DisplayExchangerateInUSD extends StatefulWidget {
const DisplayExchangerateInUSD({Key? key}) : super(key: k\
ey);

@override
_DisplayExchangerateInUSDState createState() =>
    _DisplayExchangerateInUSDState();
```

```
}

class _DisplayExchangerateInUSDState extends State<Displa\
yExchangerateInUSD> {
String selectedCurrency = 'INR';

DropdownButton<String> selectDropDownList() {
    List<DropdownMenuItem<String>> dropDownItems = [];
    for (String currency in currencyList) {
    var newItem = DropdownMenuItem(
        child: Text(currency),
        value: currency,
    );
    dropDownItems.add(newItem);
    }

    return DropdownButton<String>(
    value: selectedCurrency,
    items: dropDownItems,
    onChanged: (value) {
        setState(() {
        selectedCurrency = value!;
        });
    },
    );
}

String? value = '?';


@override
Widget build(BuildContext context) {
    return Scaffold(
    appBar: AppBar(
        title: const Text('Price Tracker'),
    ),
    body: Column(
        mainAxisAlignment: MainAxisAlignment.spaceBetween,
        crossAxisAlignment: CrossAxisAlignment.stretch,
        children: <Widget>[
        Padding(
            padding: const EdgeInsets.fromLTRB(18.0, 18.0\
, 18.0, 0),
            child: Card(
            color: Colors.redAccent,
            elevation: 5.0,
            shape: RoundedRectangleBorder(
                borderRadius: BorderRadius.circular(10.0),
            ),
            child: Padding(
                padding: const EdgeInsets.symmetric(
                    vertical: 15.0, horizontal: 28.0),
                child: Text(
                '1 USD = $value $selectedCurrency',
                textAlign: TextAlign.center,
                style: const TextStyle(
                    fontSize: 20.0,
                    color: Colors.white,
                ),
                ),
            ),
            ),
        ),
        Container(
            height: 150.0,
            alignment: Alignment.center,
            padding: const EdgeInsets.only(bottom: 30.0),
            color: Colors.red,
            child: selectDropDownList(),
        ),
          ],
      ),
      );
}
}
```

Finally, our goal is to use the API of the https://exchangeratesapi.io/ to display the exchange rate.

In that case, the drop down menu item will track the price.

We will discuss that part in our next step.

By that time if you want to clone this step, please use this GitHub repository.

HTTP Request in Flutter: PriceTracker App Final step

What is HTTP request in Flutter? Is it different than other HTTP request? The answer is NO. Like any other application, Flutter App also perform HTTP request like GET and POST.

Flutter performs the same HTTP request. But it makes in a simple way.

Why?

Because Flutter uses the HTTP plugin. And this plugin makes the job simple.

In this section, we will see how we can perform HTTP request to an exchange rate website API. By using the API we will make the HTTP request and get the conversion rate.

In our PriceTracker App final step, we will see what is the exchange rate against 1 US dollar. By default, it shows the exchange rate of

Indian rupees. You can set the initial currency according to your country.

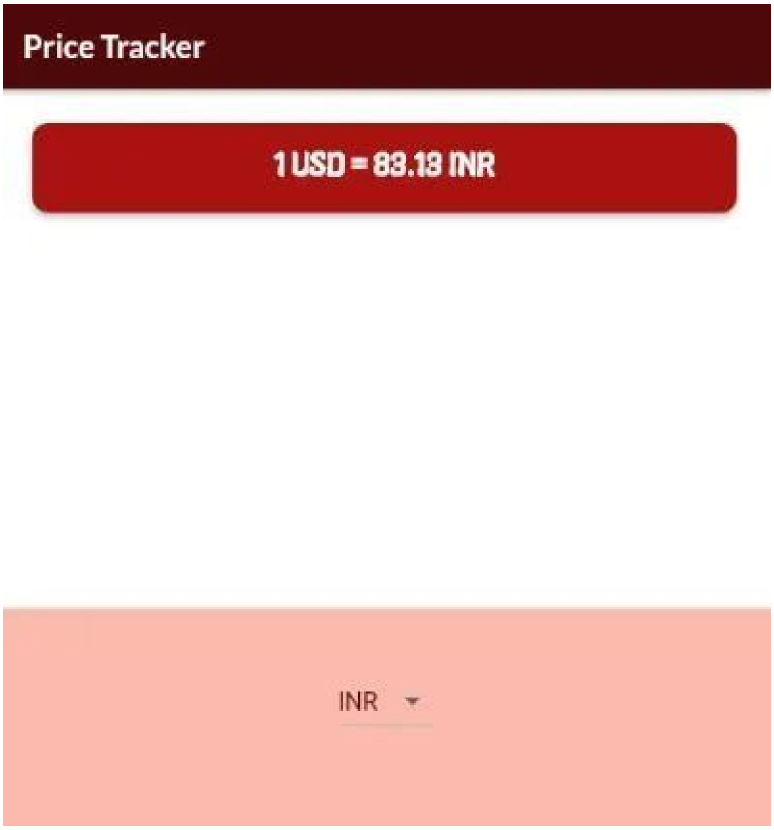


Figure 10.4 – HTTP request in Flutter first example

For example, a few hours back, our PriceTracker app displays that 1 dollar is equal to 1 euro. Actually, we had made a HTTP get request to the website.

At present, euro gets stronger and 1 dollar is equal to 0.92 euro. It always changes.

As a result, whenever you make a HTTP get request, the exchange rate may vary.

How do I send a HTTP request in flutter? It’s easy.

We use the HTTP plugin. Consequently, we add the dependency in our “pubspec.yaml” file.

```
dependencies:
flutter:
    sdk: flutter

cupertino_icons: ^1.0.2
http: ^0.13.4
google_fonts: ^2.3.1
```

We have also added the Google fonts plugin to add some more styling.

As we have seen earlier, to perform HTTP request and use the API, we need Future object.

Therefore, in our model folder, we need to use the API key. You can create a free API key, however, with that you may send 250 HTTP request.

If you want more, you need to use register to that site commercially.

Let us see the data model class first.

```
import 'dart:convert';
import 'package:http/http.dart' as http;

const List<String> currencyList = [
'INR',
'JPY',
'MXN',
'NOK',
'NZD',
'PLN',
'RON',
'BRL',
'CAD',
'CNY',
'EUR',
'GBP',
'HKD',
'IDR',
'ILS',
'RUB',
];

const String usd = 'USD';

const apiKey = 'Secret API Key';

class USDData {
Future getUSDDataMap(String selectedCurrency, String usd)\
 async {
    Map<String, String> convertedToSelectedCurrency = {};
    final httpUri = Uri.http('api.exchangeratesapi.io', '\
/v1/latest', {
    'access_key': apiKey,
    });

    var response = await http.get(httpUri);
    if (response.statusCode == 200) {
    var decodedData = jsonDecode(response.body);
    double price = decodedData['rates'][selectedCurrency];
    convertedToSelectedCurrency[usd] = price.toStringAsFi\
xed(2);
    } else {
    print(response.statusCode);
    throw 'Problem with the get request';
    }
    return convertedToSelectedCurrency;
}
}
```

Firstly, the class has a Future method that uses async and await keyword. The response data comes as a JSON map data.

As a result, we have to decode the JSON map data. And after that, we need to access the value by the key.

Secondly, this part is important.

```
var response = await http.get(httpUri);
    if (response.statusCode == 200) {
    var decodedData = jsonDecode(response.body);
    double price = decodedData['rates'][selectedCurrency];
    convertedToSelectedCurrency[usd] = price.toStringAsFi\
xed(2);
...
```

After that, we have to pass this Future data to the page where we will display the converted currency.

And, we can select the currency from the drop down menu item widget.

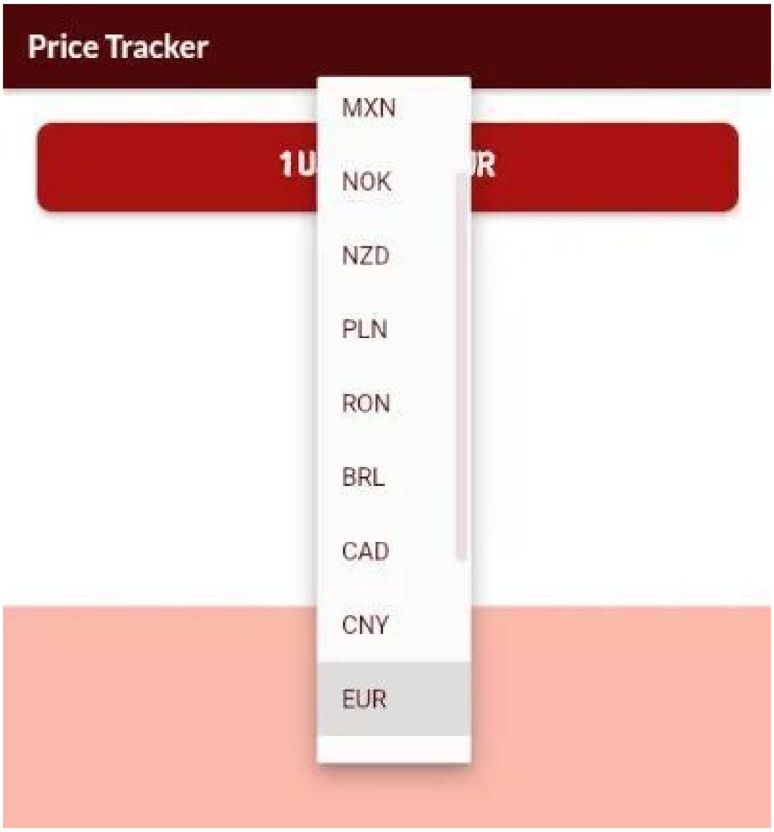


Figure 10.5 – HTTP request in Flutter second example

How do I get HTTP request flutter?

As we have sent the HTTP request, the website gets back to our app with a response.

According to that response, we send the data to the next page.

How do we get and display the data?

Let us see the full code.

```
import 'package:flutter/material.dart';
import 'package:price_tracker/model/happy_theme.dart';
import '../model/usd_data.dart';

class DisplayExchangerateInUSD extends StatefulWidget {
const DisplayExchangerateInUSD({Key? key}) : super(key: k\
ey);

@override
_DisplayExchangerateInUSDState createState() =>
    _DisplayExchangerateInUSDState();
}

class _DisplayExchangerateInUSDState extends State<Displa\
yExchangerateInUSD> {
String selectedCurrency = 'INR';

DropdownButton<String> selectDropDownList() {
    List<DropdownMenuItem<String>> dropDownItems = [];
    for (String currency in currencyList) {
    var newItem = DropdownMenuItem(
        child: Text(currency),
        value: currency,
    );
    dropDownItems.add(newItem);
    }

    return DropdownButton<String>(
    value: selectedCurrency,
    items: dropDownItems,
    onChanged: (value) {
        setState(() {
        selectedCurrency = value!;
        getConvertedValueInSelectedCurrency();
        });
    },
    );
}

String? value = '?';
String key = 'USD';

void getConvertedValueInSelectedCurrency() async {
    try {
    Map<String, String>? data =
        await USDData().getUSDDataMap(selectedCurrency, k\
ey);
    setState(() {
        value = data![key];
    });
    } catch (e) {
    throw '$e';
    }
}

@override
void initState() {
    super.initState();
    getConvertedValueInSelectedCurrency();
}

@override
Widget build(BuildContext context) {
    return Scaffold(
    appBar: AppBar(
        backgroundColor: HappyTheme.inactiveCoor,
        title: Text(
        'Exchange Rate',
        style: HappyTheme.appbarStyle,
        ),
    ),
    body: Column(
        mainAxisAlignment: MainAxisAlignment.spaceBetween,
        crossAxisAlignment: CrossAxisAlignment.stretch,
        children: <Widget>[
        Padding(
            padding: const EdgeInsets.fromLTRB(18.0, 18.0\
, 18.0, 0),
            child: Card(
            color: HappyTheme.shrineBrown600,
            elevation: 5.0,
            shape: RoundedRectangleBorder(
                borderRadius: BorderRadius.circular(10.0),
            ),
            child: Padding(
                padding: const EdgeInsets.symmetric(
                    vertical: 15.0, horizontal: 28.0),
                child: Text(
                '1 USD = $value $selectedCurrency',
                textAlign: TextAlign.center,
                style: HappyTheme.answerStyle,
                ),
            ),
            ),
        ),
        Container(
            height: 150.0,
            alignment: Alignment.center,
            padding: const EdgeInsets.only(bottom: 30.0),
            color: HappyTheme.shrinePink300,
            child: selectDropDownList(),
        ),
        ],
    ),
    );
```

```
}
}
```

The above code is large. However, the logic is simple.

Since we have got the response of HTTP request in JSON map data, we have used the key to grab the value.

Our code works as we have used the Stateful widget. Therefore, when we select the currency as EUR or euro, it stores the converted value and displays.

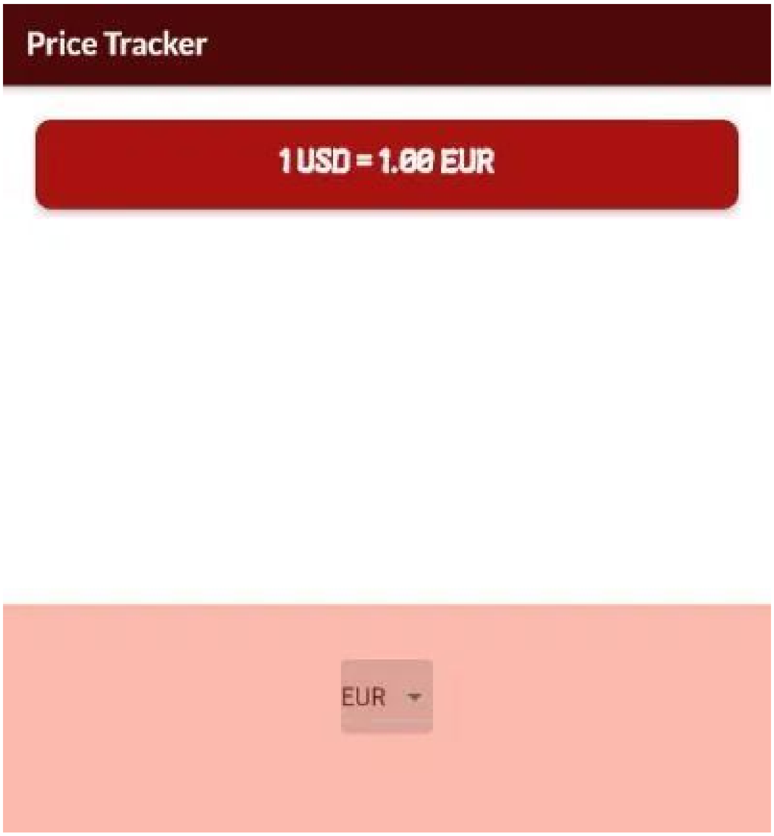


Figure 10.6 – HTTP request in Flutter third example

Certainly, you can clone the whole GitHub repository in your local machine to run the code.

But before running the code, you must create your own API key.

- For full code snippet please visit the respective GitHub Repository - [33]
- Read updated articles on Flutter, Dart, and Algorithm - [34]

[33]https://github.com/sanjibsinha/exchange_rate
[34]https://sanjibsinha.com

11. A Chat App with Firebase Authentication and Firestore Database

With Flutter Firebase we can build a Chat App. In this section, we will learn the rules and see how a user can register and login by email and password.

For example, to register and login process we use the Firebase authentication.

After that, we will use the Firestore database to save our Chat messages that user send to each other.

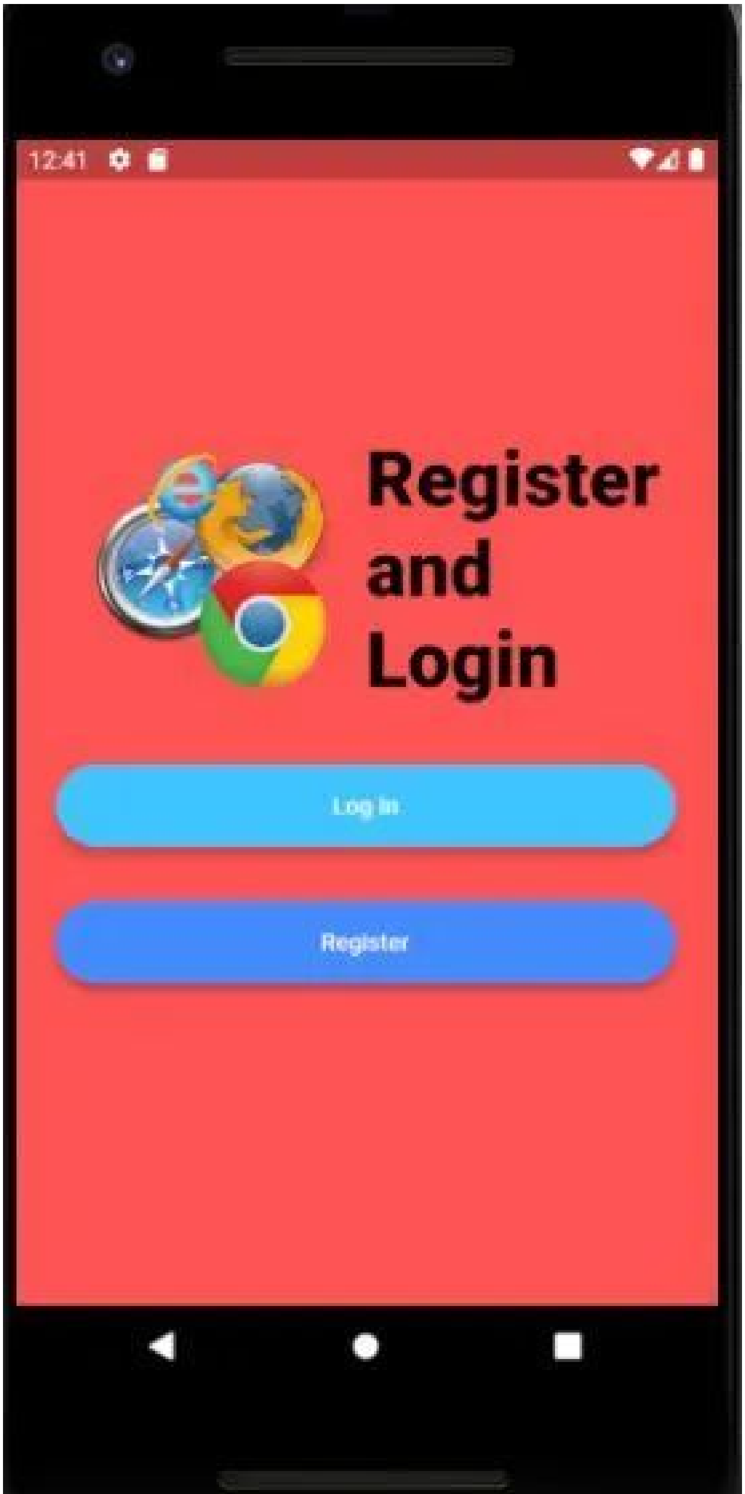
By the way, in previous sections, we have been building a News App where Flutter acts as the frontend. But the WordPress acts as the backend.

Although we have not finished that app yet in between we can take a break to build a small Chat app with Flutter Firebase.

Why we use the Firebase?

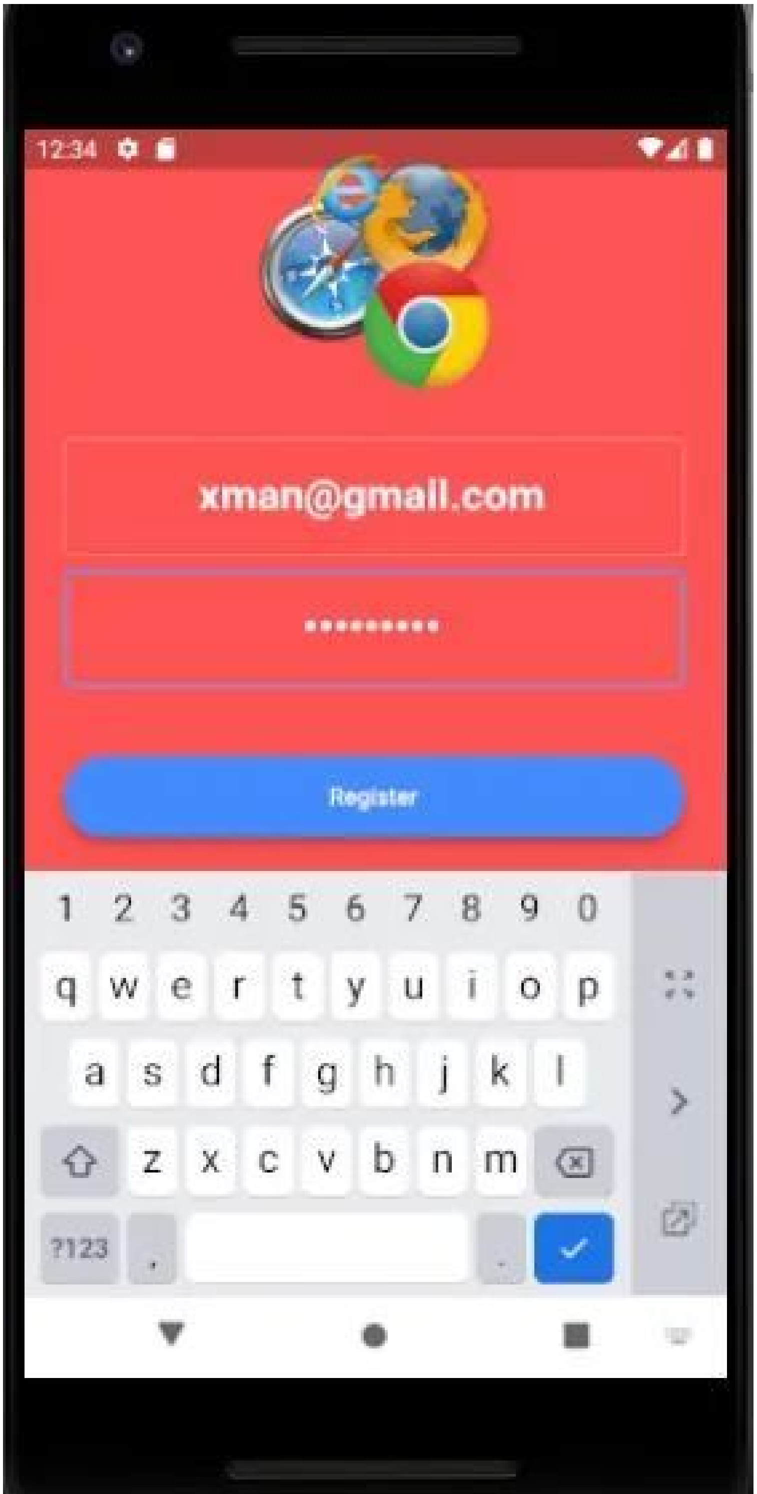
Because just like the WordPress we can also use Firebase as the backend to Flutter.

Before discussing the code and other rules of Firebase, let us see how we can authenticate users through the Flutter App.


12:41
Register
and
Login
Log In
Register

This is our welcome page where the new user can register. And the existing user can login to join the Chat page.

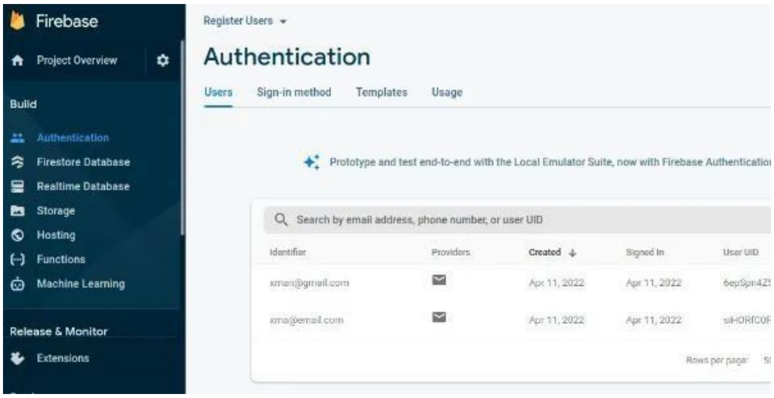
The registration page looks as follows.


12:34
xman@gmail.com
•••••••••
Register

We see that in the registration page a user is registering with email and password.

Once the user gets registered, the data of the user will be available in the Firebase authentication page.

Let’s see the screenshot. That will explain.



Identifier	Providers	Created	Signed In	User UID
xman@gmail.com	✉	Apr 11, 2022	Apr 11, 2022	6epSpn4Z
xma@email.com	✉	Apr 11, 2022	Apr 11, 2022	siHORfC0F

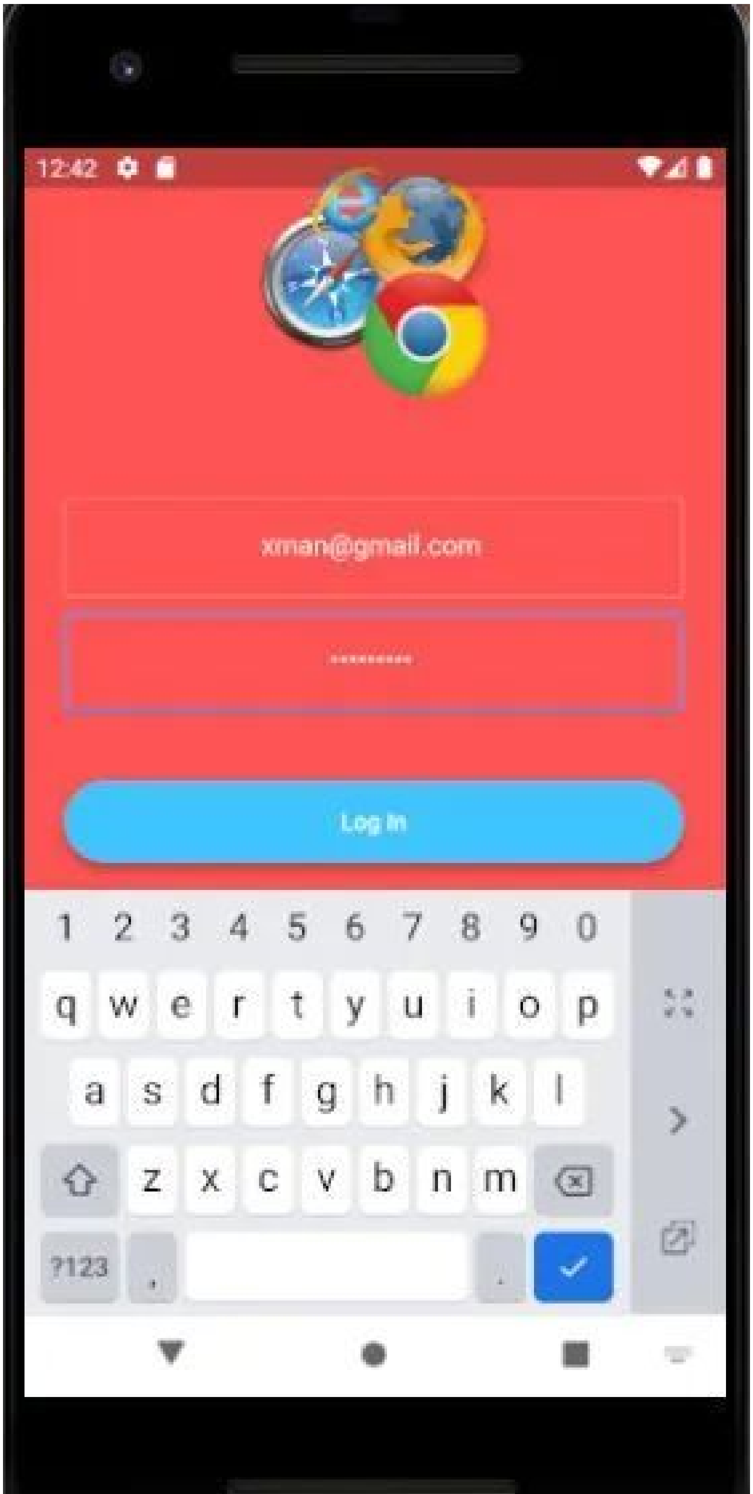
Figure 11.3 – Firebase authentication page shows list of users.

Just like the registration page, the login page also looks the same.

For instance, the login page takes inputs from the users. And as a result, the registered-user can type in the email and password.

The Firebase authentication instantly checks the credential and sends the user to the Chat page.

Let’s see how the login page looks like.


12:42
xman@gmail.com
••••••••••
Log In

Is Flutter and Firebase full stack?

Yes, Flutter and Firebase can work together to build a full stack App. In fact, the Chat App we are going to create will be an example of full stack app.

However, we must remember that Flutter is a frontend UI toolkit that wants a backend support to work with any database.

In that scenario, Firebase acts as the backend database.

In case of the News App, we have seen how WordPress gives the backend database support.

The same thing happens in case of Firebase.

Is Firebase easy to learn?

Certainly, for a beginner, using the Firebase as a backend to the Flutter may look tough.

But at one time, we all have started as a beginner. Right?

Therefore, just follow the instructions and code along with these steps. You will find that neither Firebase, nor the Firestore looks tough anymore.

So, first thing first.

We need to login to the Firebase home page with our Gmail account. After that we need to click the "Go to Console" tab.

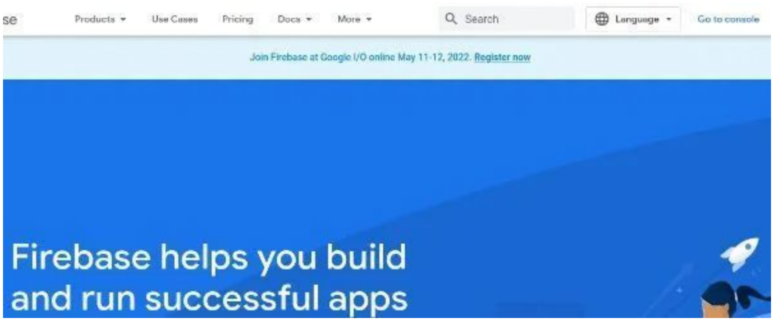

se
Products
Use Cases
Pricing
Docs
More
Search
Language
Go to console
Join Firebase at Google I/O online May 11-12, 2022. Register now
Firebase helps you build and run successful apps

Figure 11.5 – Firebase *Go to Console*

You will find it on the top right hand corner of the page.

Once you click the tab, it will take you to the “Add a Project” page.

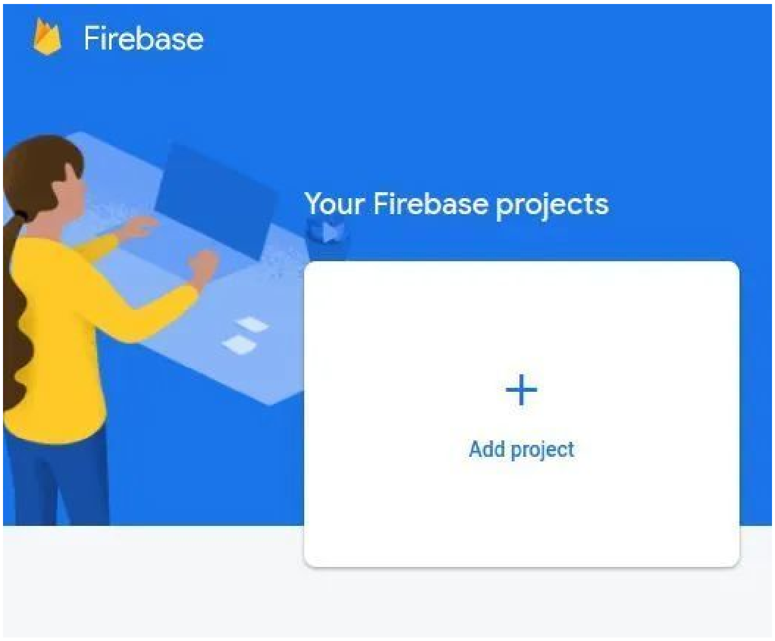


Figure 11.6 – Firebase Add Project

The Firebase Add project will guide you to do the rest of the things.

It will ask you to download a Google services JSON file. After downloading the file, we need to place it inside the project's "android/app" folder.

Meanwhile we should also add a few lines in our "android/app/build.gradle" file. Besides, in our "android/build.gradle" file we need to add a line.

From where we will get those lines?

Firebase instructs us to copy the line and place them where they are needed.

We also need to add a few necessary packages to our "pubspec.yaml" file dependency section.

dependencies: flutter: sdk: flutter

```
cupertino_icons: ^1.0.2
firebase_core: ^1.14.0
firebase_auth: ^3.3.13
cloud_firestore: ^3.1.11
```

We need to add the firebase_core, firebase_auth, and the cloud_firestore packages.

As we progress, we will see how these packages work together when we build the Chat app.

So stay tuned and we will meet again in the next section.

Flutter and Firebase: How to Initialise App and Avoid Errors

In the previous section we have learned that Flutter and Firebase can work together. In fact, Flutter works as the frontend. And Firebase acts as backend.

As a result, together they work as a full stack app.

However, before using the Firebase, we need to know a few basic rules.

For a small Flutter app, Firebase is free. In that case, we should limit ourselves within 100 connections.

For example, more than 100 users cannot use the Chat app that we've been building.

But for a big app where many users can participate, we can pay and increase our storage capacity.

What is Firebase in Flutter?

Beginners want to know the answer. So we need to clarify.

When we clarify, it makes sense. Right?

Firebase gives different types of backend services to Flutter.

What does the backend service mean? In fact, it means many things.

real time database cloud storage authentication crash reporting machine learning and many more…

Since Flutter is a frontend UI toolkit, we need a backend service.

In the previous sections we have seen how we have built a News App with WordPress as the backend.

Not only that, we have built a personal Blog App with SQLite database as our backend.

Therefore, we can do many things and use Firebase in the similar vein.

Initialise App and avoid errors

Firebase has changed its rules and developed gradually. Therefore, please read the documentation to get the updated news.

One of them is we need to initialise Firebase at the entry point of our Flutter app.

As an outcome, our top level main() function looks as follows.

void main() async { WidgetsFlutterBinding.ensureInitialized(); await Firebase.initializeApp(); runApp(const FirebaseLoginRegister()); }

But we need to import the "firebase_core" package which we've added as a dependency.

import 'package:firebase_core/firebase_core.dart';

We also need some other packages as well to run the Chat App. We have discussed that topic in our previous section.

dependencies: flutter: sdk: flutter

```
cupertino_icons: ^1.0.2
firebase_core: ^1.14.0
firebase_auth: ^3.3.13
cloud_firestore: ^3.1.11
```

Not only that, we also have to add another Firebase method before we initialise the State of the Widget.

void initState() { super.initState(); Firebase.initializeApp().whenComplete(() { setState(() {}); }); }

We'll discuss this topic in detail. With that we'll also discuss other important concepts.

Flutter and Firebase can act together. But we should know the rules to use them as the full stack.

So stay tuned. We'll meet in the next section.

Firebase and Flutter: Chat App authentication

In the previous sections we have learned how we can avoid errors while we use Firebase and Flutter.

Moreover, we have learned how to customise the button.

Now we'll finish the authentication section pf our Chat App. To do that we need Firebase and Flutter. Most importantly, they should work together.

To sum up, the Firebase gives different types of backend services to Flutter.

What does the backend service mean?

In fact, it means many things.

real time database cloud storage authentication crash reporting machine learning and many more…

Since Flutter is a frontend UI toolkit, therefore we need a backend service.

Certainly we can use other backend service. As we have built a News App with WordPress as the backend.

Let us see the top-level main function first.

import 'package:flutter/material.dart'; import 'package:firebase_-core/firebase_core.dart';

import 'view/chat.dart'; import 'view/login.dart'; import 'view/register.dart'; import 'view/welcome.dart';

void main() async { WidgetsFlutterBinding.ensureInitialized(); await Firebase.initializeApp(); runApp(const FirebaseLoginRegister()); }

class FirebaseLoginRegister extends StatelessWidget { const FirebaseLoginRegister({Key? key}) : super(key: key);

```
@override
Widget build(BuildContext context) {
  return MaterialApp(
    debugShowCheckedModeBanner: false,
    theme: ThemeData.dark().copyWith(
      textTheme: const TextTheme(
        bodyText1: TextStyle(color: Colors.black54),
      ),
    ),
    initialRoute: Welcome.id,
    routes: {
      Welcome.id: (context) => const Welcome(),
      Login.id: (context) => const Login(),
      Register.id: (context) => const Register(),
      Chat.id: (context) => const Chat(),
    },
  );
}
```

}

Firstly, we need to initialise the Firebase service. After that, we have used the"routes" property of the Material App Widget to define various routes.

We need a welcome page where the new user either register or the existing user will login so that they can chat.

How Firebase authentication works

In Firebase, there are several ways to authenticate users. The most popular one is, of course, the email authentication.

First we need to enable that service.

After that, in the welcome page we use two buttons so that users can either register, or login.

import ‘package:flutter/material.dart’; import ‘package:firebase_-core/firebase_core.dart’; import ‘../model/rounded_button.dart’; import ‘register.dart’;

import ‘login.dart’;

class Welcome extends StatefulWidget { const Welcome({Key? key}) : super(key: key); static const String id = ‘welcome’;

```
@override
_WelcomeState createState() => _WelcomeState();
```

}

class _WelcomeState extends State<Welcome> { @override void initState() { super.initState(); Firebase.initializeApp().whenComplete(() { setState(() {}); }); }

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Colors.redAccent,
    body: Padding(
      padding: const EdgeInsets.symmetric(horizontal: 24.\
0),
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        crossAxisAlignment: CrossAxisAlignment.stretch,
        children: <Widget>[
          Row(
            children: <Widget>[
              Expanded(
                child: Hero(
                  tag: 'logo',
                  child: Container(
                    padding: const EdgeInsets.all(8.0),
                    child: Image.network(
                        'https://cdn.pixabay.com/photo/20\
15/05/19/07/44/browser-773215_960_720.png'),
                    height: 160.0,
                    width: 160.0,
                  ),
                ),
              ),
              const Expanded(
                child: Text(
                  'Chatting Friends: Register or Login',
                  style: TextStyle(
                    fontSize: 45.0,
                    color: Colors.black,
                    fontWeight: FontWeight.w900,
                  ),
                ),
              ),
            ],
          ),
          const SizedBox(
            height: 18.0,
          ),
          RoundedButton(
            title: 'Log In',
            colour: Colors.black45,
            onPressed: () {
              Navigator.pushNamed(context, Login.id);
            },
          ),
          RoundedButton(
            title: 'Register',
            colour: Colors.black45,
            onPressed: () {
              Navigator.pushNamed(context, Register.id);
            },
            ),
          ],
        ),
      ),
    );
  }
```

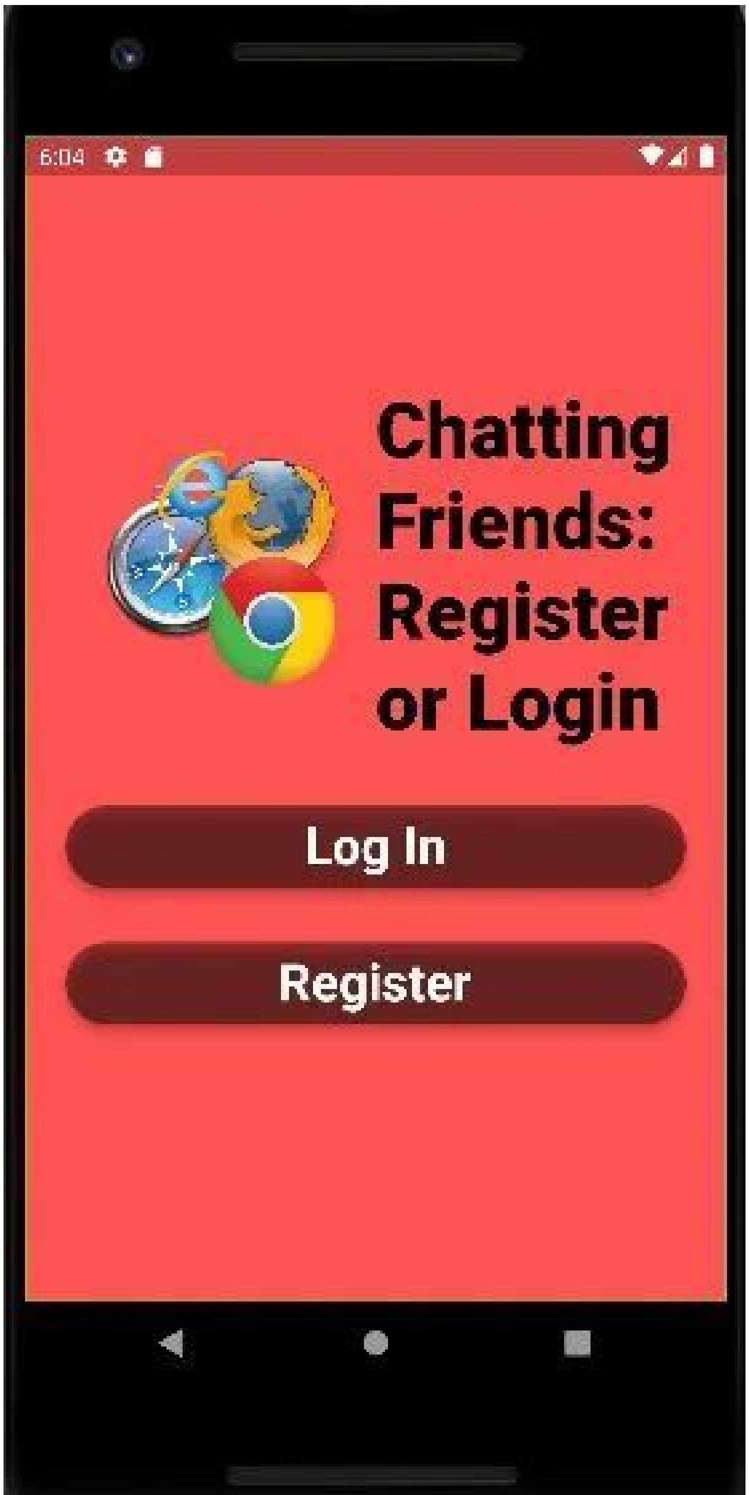
}

When Flutter initialises the State, we also initialise the Firebase service.

Certainly inside the setState() method, we can do some more tasks.

class _WelcomeState extends State<Welcome> { @override void initState() { super.initState(); Firebase.initializeApp().whenComplete(() { setState(() {}); }); } …

The welcome page looks as follows.


6:04
Chatting Friends: Register or Login
Log In
Register

In the register page we need two Firebase packages that we have already added in our “pubspec.yaml” file.

dependencies: flutter: sdk: flutter

```
cupertino_icons: ^1.0.2
firebase_core: ^1.14.0
firebase_auth: ^3.3.13
cloud_firestore: ^3.1.11
```

To clarify, the “firebase_auth” and the “firebase_core” packages help us to create the Firebase authentication instance.

final _auth = FirebaseAuth.instance; As a result, we can use that instance to create users email and password.

final user = await _auth.createUserWithEmailAndPassword(email: email!, password: password!,);

If we take a look at the register page, that will clarify how we use Firebase and Flutter.

It’s simple.

Above all, the Firebase packages take care of the authentication which we want. In addition, later the Flutter app authenticate users based on that email and password.

Let’s see the code of register page.

import ‘package:flutter/material.dart’; import ‘package:firebase_-auth/firebase_auth.dart’; import ‘package:firebase_core/firebase_-core.dart’;

import ‘chat.dart’;

import ‘../model/rounded_button.dart’;

class Register extends StatefulWidget { const Register({Key? key}) : super(key: key); static const String id = ‘register’;

```
@override
_RegisterState createState() => _RegisterState();
```

}

class _RegisterState extends State<Register> { final _auth = FirebaseAuth.instance;

```
String? email;
String? password;

User? loggedInUser;

@override
void initState() {
  super.initState();
  Firebase.initializeApp().whenComplete(() {
    setState(() {});
  });
  getUser();
}

void getUser() {
  try {
    final user = _auth.currentUser;
    if (user != null) {
      loggedInUser = user;
    }
  } catch (e) {
    throw e.toString();
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Colors.redAccent,
    body: Padding(
      padding: const EdgeInsets.symmetric(horizontal: 24.\
0),
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        crossAxisAlignment: CrossAxisAlignment.stretch,
        children: <Widget>[
          Flexible(
            child: Hero(
              tag: 'logo',
              child: Container(
                padding: const EdgeInsets.all(8.0),
                child: Image.network(
                    'https://cdn.pixabay.com/photo/2015/0\
5/19/07/44/browser-773215_960_720.png'),
                height: 160.0,
                width: 160.0,
              ),
            ),
          ),
          const SizedBox(
            height: 18.0,
          ),
          TextField(
            decoration: const InputDecoration(
              border: OutlineInputBorder(),
              hintText: 'Enter Email',
            ),
            keyboardType: TextInputType.emailAddress,
            textAlign: TextAlign.center,
            onChanged: (value) {
              email = value;
            },
            style: const TextStyle(
                fontSize: 35.0,
                fontWeight: FontWeight.bold,
                color: Colors.white,
              ),
            ),
            const SizedBox(
              height: 8.0,
            ),
            TextField(
              decoration: const InputDecoration(
                border: OutlineInputBorder(),
                hintText: 'Enter Password',
              ),
              obscureText: true,
              textAlign: TextAlign.center,
              onChanged: (value) {
                password = value;
              },
              style: const TextStyle(
                fontSize: 35.0,
                fontWeight: FontWeight.bold,
                color: Colors.white,
              ),
            ),
            const SizedBox(
              height: 24.0,
            ),
            RoundedButton(
              title: 'Register',
              colour: Colors.black45,
              onPressed: () async {
                setState(() {});
                try {
                  final user = await _auth.createUserWithEm\
ailAndPassword(
                    email: email!,
                    password: password!,
                  );
                  if (user != null) {
                    Navigator.pushNamed(context, Chat.id);
                  }

                  setState(() {});
                } catch (e) {
                  throw e.toString();
                }
              },
            ),
          ],
        ),
      ),
    );
  }
```

}

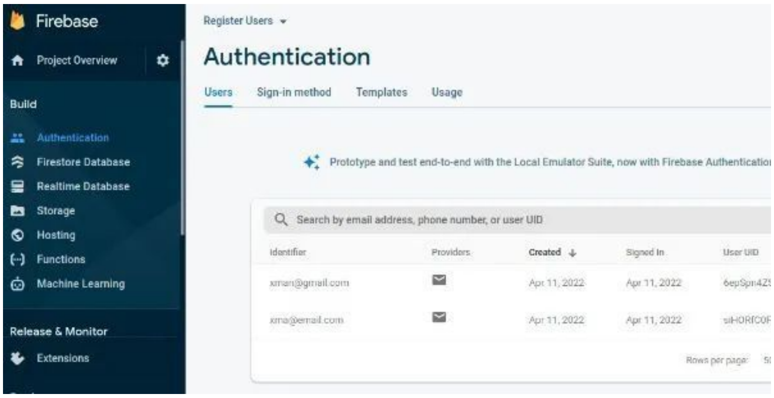
In the above code, the highlighted sections play the important role.

Is Firebase a backend or database?

As we have said earlier, Firebase is a backend service that supports many things. And, of course, NoSQL database is one of them.

Moreover, it keeps data in JSON format, and that makes it faster than other database.

When the user registers we can see the data in our Firebase console.



Prototype and test end-to-end with the Local Emulator Suite, now with Firebase Authenticatio

Search by email address, phone number, or user UID

Identifier	Providers	Created	Signed In	User UID
xman@gmail.com	✉	Apr 11, 2022	Apr 11, 2022	6epSpn4Z
xma@email.com	✉	Apr 11, 2022	Apr 11, 2022	siHORfC0F

Rows per page: 5

Figure 11.8 – Flutter Firebase users have been added

As an outcome now a registered user can login and take part in chatting with other registered users.

The code of login page is almost identical as the register page.

import 'package:flutter/material.dart'; import 'package:firebase_-core/firebase_core.dart'; import 'package:firebase_auth/firebase_-auth.dart'; import 'chat.dart';

import '../model/rounded_button.dart';

class Login extends StatefulWidget { const Login({Key? key}) : super(key: key); static const String id = 'login';

```
@override
_LoginState createState() => _LoginState();
```

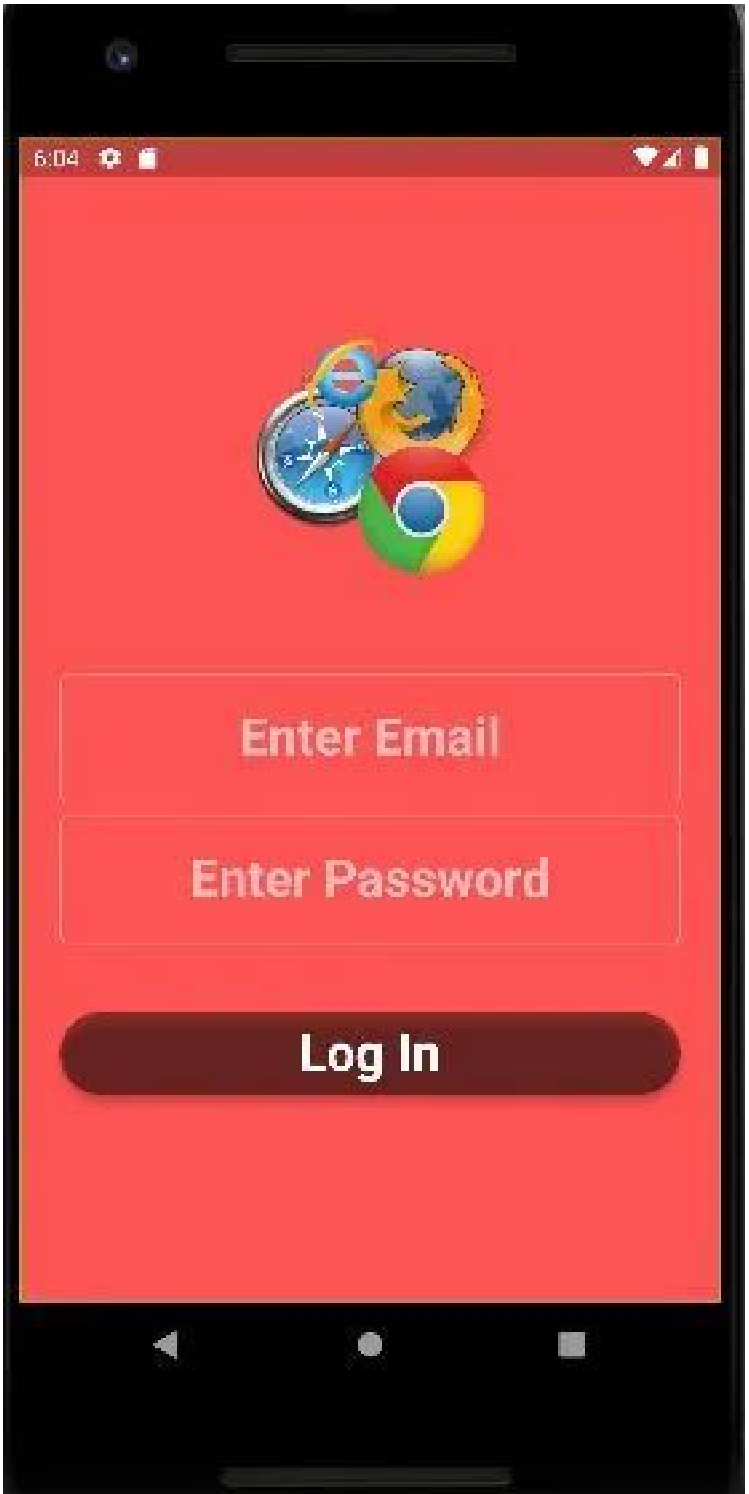
}

class _LoginState extends State<Login> { final _auth = FirebaseAuth.instance; String? email; String? password;

```
@override
void initState() {
  super.initState();
  Firebase.initializeApp().whenComplete(() {
    setState(() {});
  });
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Colors.redAccent,
    body: Padding(
      padding: const EdgeInsets.symmetric(horizontal: 24.\
0),
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        crossAxisAlignment: CrossAxisAlignment.stretch,
        children: <Widget>[
          Flexible(
            child: Hero(
              tag: 'logo',
              child: Container(
                padding: const EdgeInsets.all(8.0),
                child: Image.network(
                    'https://cdn.pixabay.com/photo/2015/0\
5/19/07/44/browser-773215_960_720.png'),
                height: 160.0,
                width: 160.0,
              ),
            ),
          ),
          const SizedBox(
            height: 48.0,
          ),
          TextField(
            decoration: const InputDecoration(
              border: OutlineInputBorder(),
              hintText: 'Enter Email',
            ),
            keyboardType: TextInputType.emailAddress,
            textAlign: TextAlign.center,
            onChanged: (value) {
              email = value;
            },
            style: const TextStyle(
              color: Colors.white,
              fontSize: 30.0,
              fontWeight: FontWeight.bold,
            ),
          ),
          const SizedBox(
            height: 8.0,
          ),
          TextField(
            decoration: const InputDecoration(
              border: OutlineInputBorder(),
              hintText: 'Enter Password',
            ),
            obscureText: true,
            textAlign: TextAlign.center,
            onChanged: (value) {
              password = value;
            },
            style: const TextStyle(
              color: Colors.white,
              fontSize: 30.0,
              fontWeight: FontWeight.bold,
            ),
          ),
            const SizedBox(
              height: 24.0,
            ),
            RoundedButton(
              title: 'Log In',
              colour: Colors.black45,
              onPressed: () async {
                setState(() {});
                try {
                  final user = await _auth.signInWithEmailA\
ndPassword(
                    email: email!,
                    password: password!,
                  );
                  if (user != null) {
                    Navigator.pushNamed(
                      context,
                      Chat.id,
                    );
                  }

                  setState(() {});
                } catch (e) {
                  throw e.toString();
                }
              },
            ),
          ],
        ),
      ),
    );
}

}
```

Flutter checks whether the user's email and password match with the Firebase authentication service.

If it matches, it takes the user to the Chat page.


6:04
Enter Email
Enter Password
Log In

For the Chat page we will use the Firestore database. In the next section, we will discuss that and see how Firestore and Flutter work together.

So stay tuned, and we will meet in the next section.

Flutter Firestore: Chat App Final Step

What is the relation between Flutter and Firestore? In this final section we'll discuss that.

Basically, when we use Firebase with Flutter, we use the database of Firestore as a service.

So Firebase and Firestore give us a few different type of backend services.

We need to understand it first.

Therefore, our target will be to connect our Flutter Chat app with the Firestore database. Right?

Why?

Because, authenticated users can can sign in and chat with other registered users.

Meanwhile we discuss this topic, we will also finish our chat app that we have been building for a while.

The previous section let's us know how to use authentication in Firebase.

Besides, we have learned how we can avoid errors while we use Firebase and Flutter. Moreover, we have also learned how to customise the button to give our Chat app a unique look.

As we have seen earlier, we need Firebase and Flutter.

Most importantly, they should work together.

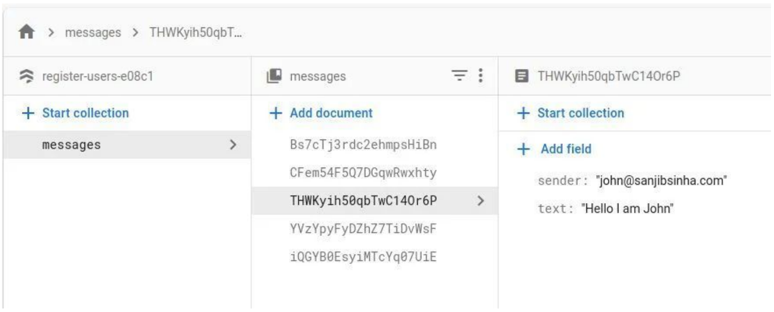
Certainly we can use other backend service for Flutter. As we have built a News App with WordPress as the backend.

How does Firestore work in Flutter? Firstly, we need a Firebase membership that we can get by signing in with the Gmail account.

Secondly, we should enable Email as our authentication service.

Finally, we can create a collection in Firestore. On the dashboard we can click the database option and create a collection.

It looks as follows.



```
Bs7cTj3rdc2ehmpsHiBn
CFem54F5Q7DGqwRwxhty
THWKyih50qbTwC14Or6P
YVzYpyFyDZhZ7TiDvWsF
iQGYB0EsyiMTcYq07UiE
```

```
sender: "john@sanjibsinha.com"
text: "Hello I am John"
```

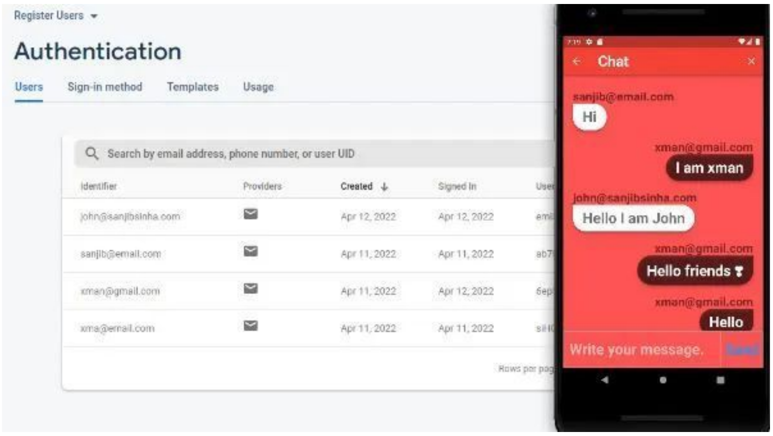
Figure 11.10 – Flutter Firestore collection

As we see the above collection requires only two fields. Both are texts. The first one represents the sender's name.

Who is the sender?

Any authenticated user who have either registered or signed in.

As a result, in our Chat page we see all the messages. Besides we can show the registered users on the Firebase page.



Identifier	Providers	Created	Signed In	User
john@sanjibsinha.com	✉	Apr 12, 2022	Apr 12, 2022	emi
sanjib@email.com	✉	Apr 11, 2022	Apr 11, 2022	ab7
xman@gmail.com	✉	Apr 11, 2022	Apr 12, 2022	6ep
xma@email.com	✉	Apr 11, 2022	Apr 11, 2022	siH

sanjib@email.com
Hi

xman@gmail.com
I am xman

john@sanjibsinha.com
Hello I am John

xman@gmail.com
Hello friends

xman@gmail.com
Hello

Figure 11.11 – Flutter Firestore Chat Page shows messages

Now our Chat App is perfectly working. As an outcome, new users can register, or the registered users can sign in.

After that, they can start chatting.

The chatting does not start automatically. To chat, we need to initialise the Firebase instance first.

@override void initState() { super.initState(); Firebase.initializeApp().whenComplete(() { setState(() {}); }); getCurrentUser(); }

After that, we can call the current user who can chat. How do we get the current user?

To get the current user we use the Firebase authentication property as follows.

final _auth = FirebaseAuth.instance;

As a result, we can define logged in user as the current user.

How do we get the logged in user?

It's simple.

final _firestore = FirebaseFirestore.instance; User? loggedInUser;

With the logged in user we declare the Firestore instance as global value.

What is the advantage?

For example, in the later part we can add messages to the Firestore database collection.

onPressed: () { messageTextController.clear(); _firestore.collection('messages').add({ 'text': messageText, 'sender': loggedInUser!.email, }); },

Consequently, when the user press the button, her message and email get added to the collection.

How do you fetch data from Firestore database in Flutter? We can create two separate Widgets to fetch data from Firestore database.

Firstly, we need a Widget that will display the data from the Firestore database. But it cannot display data if we don't have a Stream Builder widget as follows.

class MessageStreamBuilder extends StatelessWidget { const MessageStreamBuilder({Key? key}) : super(key: key);

```
@override
Widget build(BuildContext context) {
  return StreamBuilder<QuerySnapshot>(
    stream: _firestore.collection('messages').snapshots(),
    builder: (context, snapshot) {
      if (!snapshot.hasData) {
        return const Center(
          child: CircularProgressIndicator(
            backgroundColor: Colors.lightBlueAccent,
          ),
        );
      }
      final messages = snapshot.data!.docChanges.reversed;
      List<DisplayMessages> displayMessages = [];
```

```
        for (var message in messages) {
          final messageText = message.doc['text'];
          final messageSender = message.doc['sender'];

          final currentUser = loggedInUser!.email;

          final displayMessage = DisplayMessages(
            sender: messageSender,
            text: messageText,
            isMe: currentUser == messageSender,
          );

          displayMessages.add(displayMessage);
        }
        return Expanded(
          child: ListView(
            reverse: true,
            padding:
                const EdgeInsets.symmetric(horizontal: 10.0\
, vertical: 20.0),
            children: displayMessages,
          ),
        );
      },
    );
}
```

}

Only after that, we can change the style of the Container Widget that will display the data from Firestore.

Therefore, we can also define the widget which will display the message on the screen.

class DisplayMessages extends StatelessWidget { const DisplayMessages({ Key? key, required this.sender, required this.text, required this.isMe, }) : super(key: key);

```
final String sender;
final String text;
final bool isMe;

@override
Widget build(BuildContext context) {
  return Padding(
    padding: const EdgeInsets.all(10.0),
    child: Column(
      crossAxisAlignment:
          isMe ? CrossAxisAlignment.end : CrossAxisAlignm\
ent.start,
      children: <Widget>[
        Text(
          sender,
          style: const TextStyle(
            fontSize: 25.0,
            color: Colors.black54,
            fontWeight: FontWeight.bold,
          ),
        ),
        Material(
          borderRadius: isMe
              ? const BorderRadius.only(
                  topLeft: Radius.circular(30.0),
                  bottomLeft: Radius.circular(30.0),
                  bottomRight: Radius.circular(30.0))
              : const BorderRadius.only(
                  bottomLeft: Radius.circular(30.0),
                  bottomRight: Radius.circular(30.0),
                  topRight: Radius.circular(30.0),
                ),
          elevation: 5.0,
          color: isMe ? Colors.black54 : Colors.white,
          child: Padding(
            padding:
                const EdgeInsets.symmetric(vertical: 10.0\
, horizontal: 20.0),
            child: Text(
              text,
              style: TextStyle(
                color: isMe ? Colors.white : Colors.black\
54,
                fontSize: 30.0,
                fontWeight: FontWeight.bold,
              ),
            ),
          ),
        ),
      ],
    ),
  );
}
```

}

Once we have defined these two widgets which will display data, we can call them in the Chat widget as follows.

import ‘package:flutter/material.dart’; import ‘package:firebase_-auth/firebase_auth.dart’; import ‘package:cloud_firestore/cloud_-firestore.dart’; import ‘package:firebase_core/firebase_core.dart’;

final _firestore = FirebaseFirestore.instance; User? loggedInUser;

class Chat extends StatefulWidget { static const String id = ‘chat’;

```
const Chat({Key? key}) : super(key: key);
@override
_ChatState createState() => _ChatState();
```

}

class _ChatState extends State<Chat> { final messageTextController = TextEditingController(); final _auth = FirebaseAuth.instance;

```
String? messageText;

@override
void initState() {
  super.initState();
  Firebase.initializeApp().whenComplete(() {
    setState(() {});
  });
  getCurrentUser();
}

void getCurrentUser() {
  try {
    final user = _auth.currentUser;
    if (user != null) {
      loggedInUser = user;
    }
  } catch (e) {
    throw e.toString();
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Colors.redAccent,
    appBar: AppBar(
      leading: null,
      actions: <Widget>[
        IconButton(
            icon: const Icon(Icons.close),
            onPressed: () {
              _auth.signOut();
              Navigator.pop(context);
            }),
      ],
      title: const Text(
        'Chat',
        style: TextStyle(
          color: Colors.white,
          fontSize: 30.0,
          fontWeight: FontWeight.bold,
        ),
      ),
      backgroundColor: Colors.red,
    ),
    body: SafeArea(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.spaceBetween,
        crossAxisAlignment: CrossAxisAlignment.stretch,
        children: <Widget>[
          const MessageStreamBuilder(),
          Container(
            width: double.infinity,
            decoration: BoxDecoration(
              border: Border.all(
                color: Colors.grey,
              ),
            ),
            child: Row(
              crossAxisAlignment: CrossAxisAlignment.cent\
er,
              children: <Widget>[
                Expanded(
                  child: TextField(
                    controller: messageTextController,
                    onChanged: (value) {
                      messageText = value;
                    },
                    decoration: const InputDecoration(
                      border: OutlineInputBorder(),
                      hintText: 'Write your message.',
                    ),
                    style: const TextStyle(
                      fontSize: 30.0,
                      fontWeight: FontWeight.bold,
                    ),
                  ),
                ),
                TextButton(
                  onPressed: () {
                    messageTextController.clear();
                    _firestore.collection('messages').add\
({
                      'text': messageText,
                      'sender': loggedInUser!.email,
                    });
                  },
                  child: const Text(
                    'Send',
                    style: TextStyle(
                      fontSize: 30.0,
                      fontWeight: FontWeight.bold,
                    ),
                  ),
                ),
              ],
            ),
          ),
        ],
      ),
    ),
  );
```

```
}
```

}

Now users can write and send messages as well as they can view their messages.

We have highlighted some sections in the above code to understand the work flow. If you want to clone this repository and run in your local machine, please use this GitHub repository.

But we can certainly make this Chat app much better and faster with the help of the Provider package.

- Read updated articles on Flutter, Dart, and Algorithm - [35]

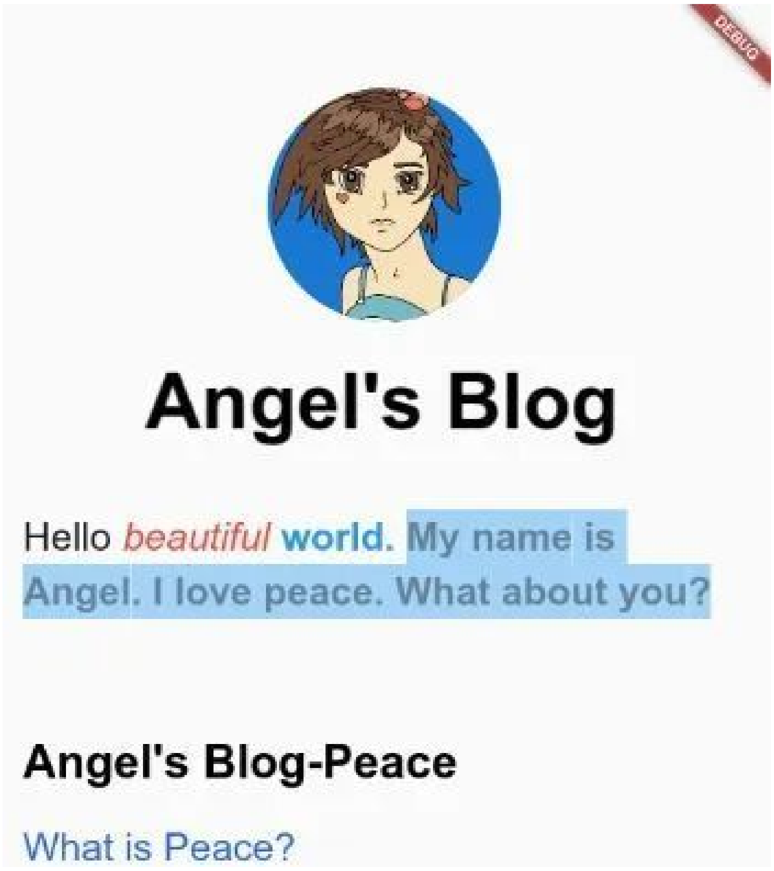
[35] https://sanjibsinha.com

12. A Blog App with Firebase Authentication and Firestore Database

As we were discussing flutter 3.0 web app, we had hard coded the initial blog data. There was no flutter sign in method so that the user could log in.

It's true that we had not introduced Firebase and Flutter sign in methods in our previous discussion. Still we had progressed a little bit.

Initially our Firebase, Provider web app looked like the following.



Angel's Blog

Hello *beautiful* world. My name is Angel. I love peace. What about you?

Angel's Blog-Peace

What is Peace?

Figure 12.1 – User can select a portion of text with selectable text in flutter

We have changed the look a little bit.

As a result, the web app now looks as follows.

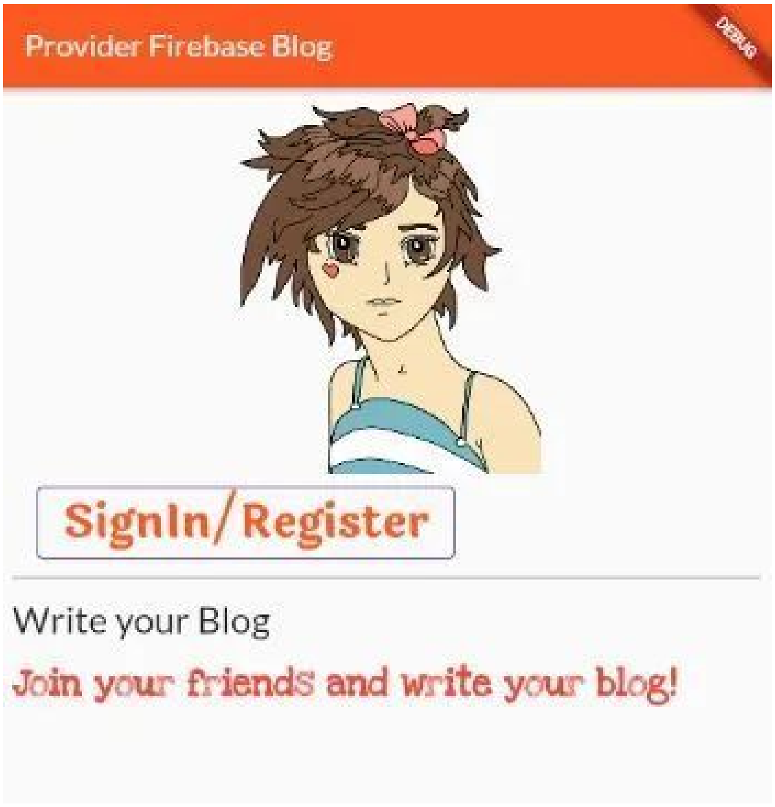


Figure 12.2 – Flutter sign in, home page

As a whole we are still not happy with the home page design, but we will work on it later.

Similarly, we will later adopt material 3 design that flutter 3.0 allows us to apply.

At present regarding this flutter sign in topic, let’s see how we can first write our business logic.

To begin with we need to go to the Firebase and create a project.

Secondly, we have discussed how to create a Firebase project before. Therefore, we’re not repeating the same procedure again.

Flutter sign in business logic

Instead let's concentrate on how we can write the business logic.

Certainly we have added all the dependencies to our pubspec.yaml file, all the same that was the beginning.

Real work starts from the Firebase email authentication page.

Why?

Because we have enabled the flutter sign in methods.

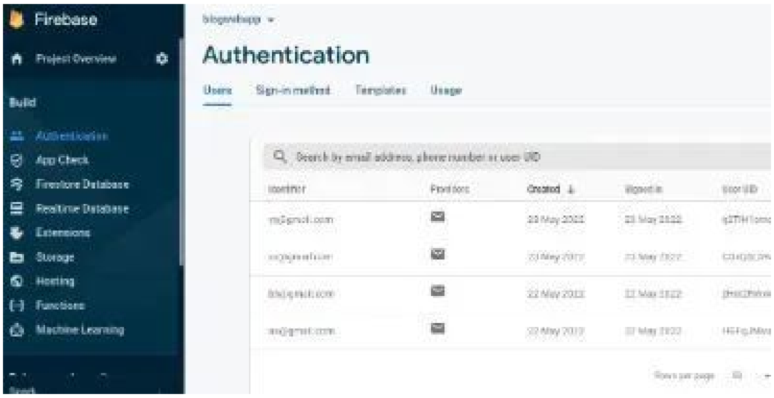


Figure 12.3 – Flutter sign in with Firebase

As a result on Firebase email authentication page we see a bunch of users' names.

In addition, in the pubspec.yaml file we have seen all the dependencies.

```
dependencies:
flutter:
    sdk: flutter


cupertino_icons: ^1.0.2
firebase_core: ^1.14.0
firebase_auth: ^3.3.13
cloud_firestore: ^3.1.11
google_fonts: ^2.3.1
provider: ^6.0.2
```

In the same vein, we can now write the flutter sign in business logic where we can test the current status of the user.

For that we have used enums. By the way, we have an article on enums earlier, you may check.

We have kept this flutter sign in logic in our controller folder.

```
import 'package:flutter/material.dart';
import 'package:google_fonts/google_fonts.dart';

import 'all_types_of_custom_forms.dart';
import 'all_widgets.dart';

enum UserStatus {
loggedOut,
emailAddress,
register,
password,
loggedIn,
}

class AuthenticationForFirebase extends StatelessWidget {
const AuthenticationForFirebase({
    required this.loginState,
    required this.email,
    required this.startLoginFlow,
    required this.verifyEmail,
    required this.signInWithEmailAndPassword,
    required this.cancelRegistration,
    required this.registerAccount,
    required this.signOut,
});

final UserStatus loginState;
final String? email;
final void Function() startLoginFlow;
final void Function(
    String email,
    void Function(Exception e) error,
) verifyEmail;
final void Function(
    String email,
    String password,
    void Function(Exception e) error,
) signInWithEmailAndPassword;
final void Function() cancelRegistration;
final void Function(
    String email,
    String displayName,
    String password,
    void Function(Exception e) error,
) registerAccount;
final void Function() signOut;

@override
Widget build(BuildContext context) {
    switch (loginState) {
    case UserStatus.loggedOut:
        return Row(
        children: [
            Padding(
            padding: const EdgeInsets.only(left: 24, bott\
om: 8),
            child: StyledButton(
                onPressed: () {
                startLoginFlow();
                },
                child: Text(
                'SignIn/Register',
                style: GoogleFonts.laila(
                    fontSize: 30.0,
                    fontWeight: FontWeight.bold,
                ),
                ),
            ),
            ),
        ],
        );
    case UserStatus.emailAddress:
        return CutomEmailForm(
            callback: (email) => verifyEmail(
                email, (e) => _showErrorDialog(context, '\
Invalid email', e)));
    case UserStatus.password:
        return CustomPasswordForm(
        email: email!,
        login: (email, password) {
            signInWithEmailAndPassword(email, password,
                (e) => _showErrorDialog(context, 'Failed \
to sign in', e));
        },
        );
    case UserStatus.register:
        return CustomRegistrationForm(
        email: email!,
        cancel: () {
            cancelRegistration();
        },
        registerAccount: (
            email,
            displayName,
            password,
        ) {
            registerAccount(
                email,
                displayName,
                password,
                (e) =>
                    _showErrorDialog(context, 'Failed to \
create account', e));
        },
        );
    case UserStatus.loggedIn:
        return Row(
        children: [
            Padding(
            padding: const EdgeInsets.only(left: 24, bott\
om: 8),
            child: StyledButton(
                onPressed: () {
                signOut();
                },
                child: const Text('LOGOUT'),
            ),
            ),
        ],
        );
    default:
            return Row(
            children: const [
                Text("Internal error, this shouldn't happen..\
.")，
            ],
            );
        }
}

void _showErrorDialog(BuildContext context, String title,\
 Exception e) {
    showDialog<void>(
    context: context,
    builder: (context) {
        return AlertDialog(
        title: Text(
            title,
            style: const TextStyle(fontSize: 24),
        ),
        content: SingleChildScrollView(
            child: ListBody(
            children: <Widget>[
                Text(
                '${(e as dynamic).message}',
                style: const TextStyle(fontSize: 18),
                ),
            ],
            ),
        ),
        actions: <Widget>[
            StyledButton(
            onPressed: () {
                Navigator.of(context).pop();
            },
            child: const Text(
                'OK',
                style: TextStyle(color: Colors.deepPurple\
),
            ),
            ),
          ],
          );
      },
    );
}
}
```

At any rate now our app will know the user’s status. That means, whether the user is a new user, or an existing user.

An existing user can sign in and start writing blogs.

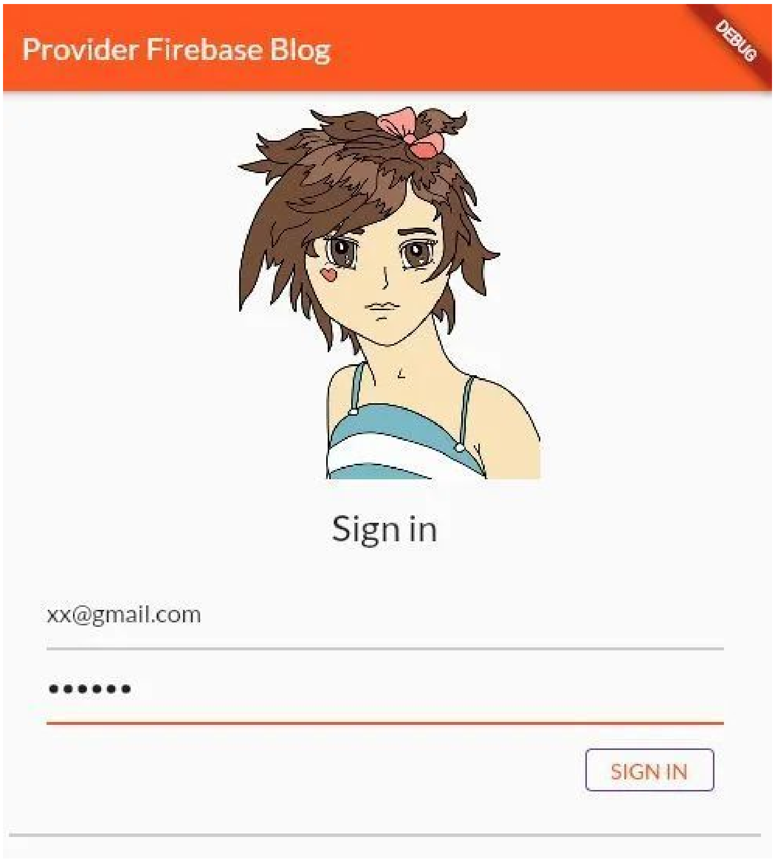


Figure 12.4 – Flutter sign in page

However, our app will check the application state and allow a new user to register.

In the next section, we will discuss that part in detail.

Multi Provider with Firebase

While building the Flutter web 3.0 blog app with Firebase and Provider, we have faced some challenges.

Firstly, we cannot hard code the blog posts anymore.

Secondly, we have to assure that only the signed-in visitors will post their blogs.

Finally, we will not use the multi provider technique. Instead we will use the ChangeNotifierProvider class.

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'model/state_of_application.dart';

import 'view/chat_app.dart';

/// moving to first branch

void main() {
runApp(
    ChangeNotifierProvider(
    create: (context) => StateOfApplication(),
    builder: (context, _) => const ChatApp(),
    ),
);
}
```

For instance, Flutter hard code is just like any other hard code principle that we cannot follow when we develop a dynamic flutter app. Right?

Why?

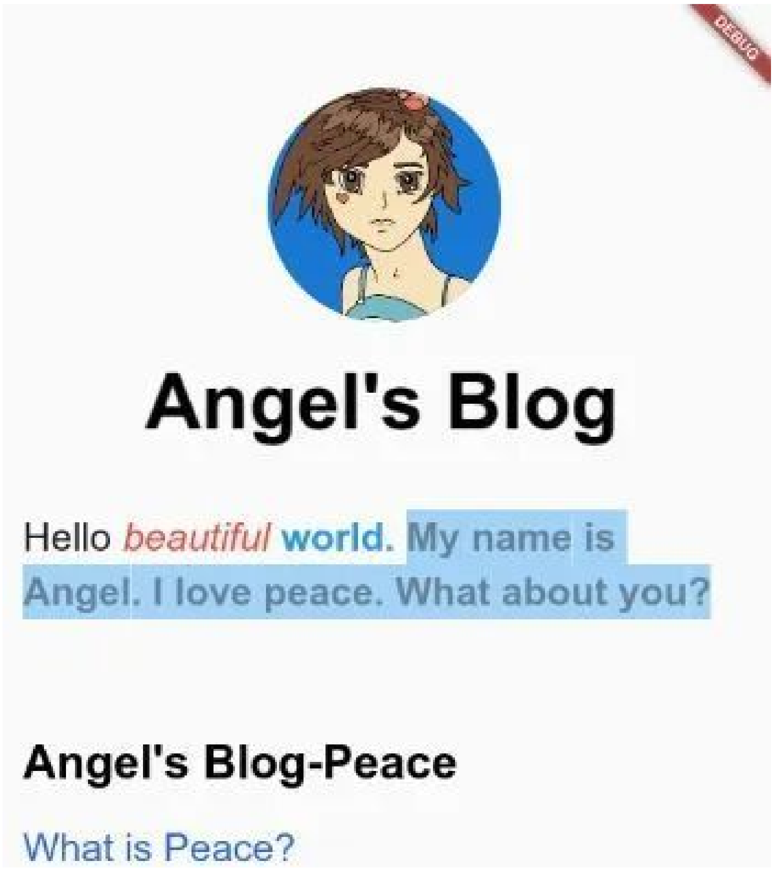
Because in our case, we're dealing with a remote database server Firebase and Firestore.

As a result, particularly at this stage, users will insert data, and retrieve data from Firebase.

On the contrary, we embed hard code into our source code. Neither it comes from any external source, nor we change the value on runtime. Right?

For example, we've seen the same instance in our Firebase, Firestore and Provider web app.

What do we see on the screen?Let's first see the home page of our flutter web app.



Angel's Blog

Hello *beautiful* world. My name is Angel. I love peace. What about you?

Angel's Blog-Peace

What is Peace?

Figure 12.5 – User can select a portion of text with selectable text in flutter

But after the initial change our current home page will look like the following.

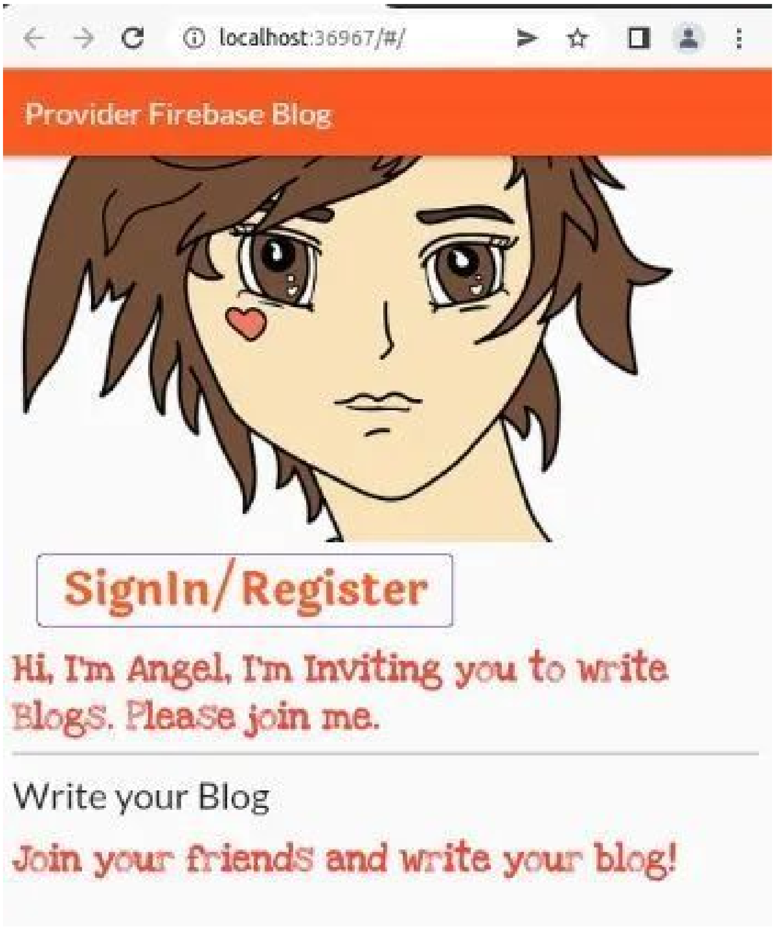

localhost:36967/#/
Provider Firebase Blog
SignIn/Register
Hi, I'm Angel, I'm Inviting you to write Blogs. Please join me.
Write your Blog
Join your friends and write your blog!

Figure 12.6 – Flutter web 3.0 homepage

As we see, here the first step is either you have to sign in, or you can register so that later you can sign in.

As an outcome, the Firebase authentication page will show the existing users who have registered already.

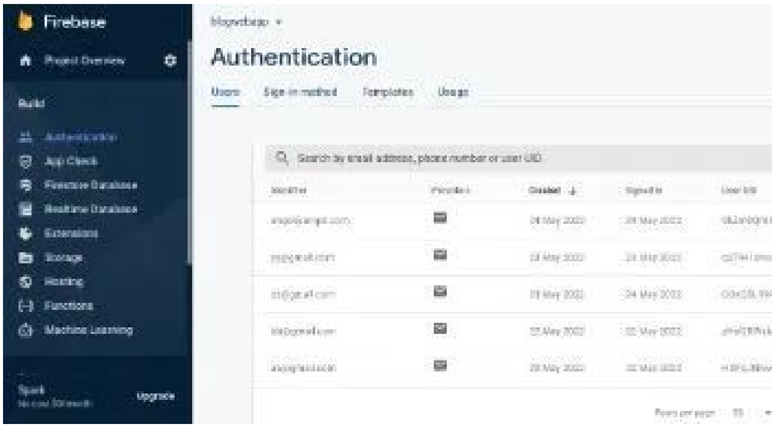


Figure 12.7 – Flutter web 3.0 Firebase authentication page

However, as we were discussing flutter 3.0 web app, initially it was different.

Why so? Because we had hard coded the initial blog data.

There was no flutter sign in method so that the user could log in.

It’s true that we had not introduced Firebase and Flutter sign in methods in our previous discussion.

How Flutter web 3.0 works with Firebase, and Provider While talking about Flutter 3.0, we have seen what are the primary changes.

Firstly, with reference to mobile application development, there has not been a great change. Structurally what we have been doing, will continue to do.

Secondly, we can work with Firebase and Provider just like before.

For that reason, we need to add the package dependencies first to the pubspec.yaml file.

```
dependencies:
flutter:
    sdk: flutter

cupertino_icons: ^1.0.2
firebase_core: ^1.14.0
firebase_auth: ^3.3.13
cloud_firestore: ^3.1.11
google_fonts: ^2.3.1
provider: ^6.0.2
```

As for the next move, we need to add the state management process in our model folder.

This class will extend the Changenotifier class and initialize the state.

Since our data source is external, we will use Future, async and awit.

In this class we will define different types of methods. It will check whether the user is new or existing.

Based on that, new users can register as follows.

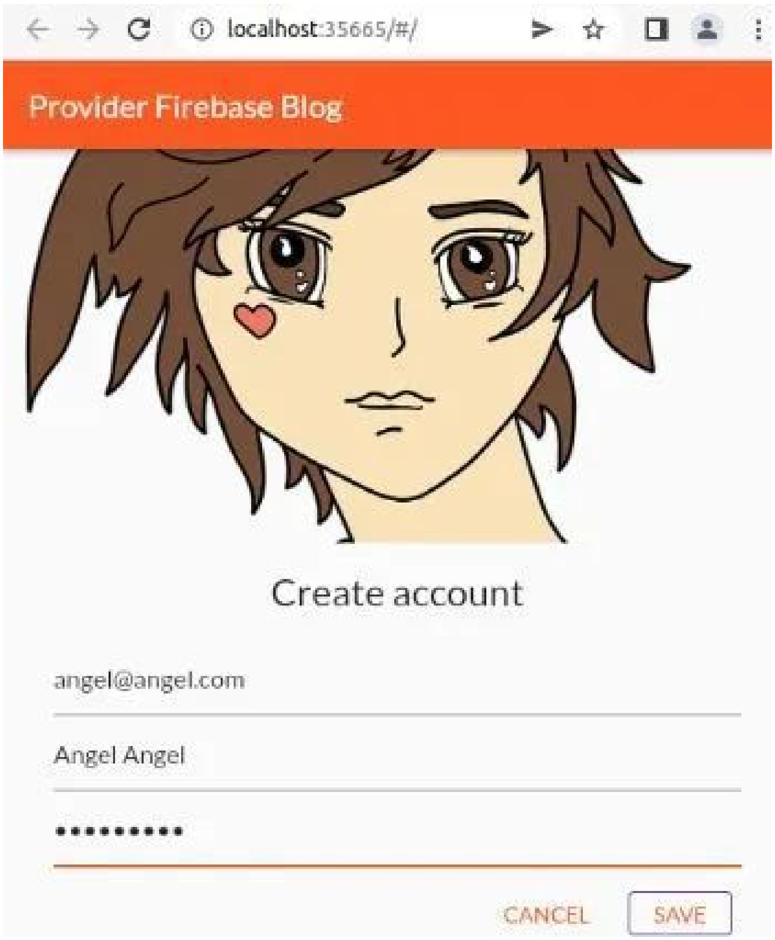


Figure 12.8 – Flutter web 3.0 sign in, register page

By the way, we can take a look at the code where it also checks whether the user is logged in or not.

On the other hand it will also add the blog posts to the Firebase Firestore collection.

```
import 'package:flutter/material.dart';
import 'dart:async';
import 'package:firebase_core/firebase_core.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';

import '../controller/authenticate_to_firebase.dart';
import '../firebase_options.dart';

import '../view/let_us_chat.dart';

class StateOfApplication extends ChangeNotifier {
StateOfApplication() {
    init();
}

Future<void> init() async {
    await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
    );

    FirebaseAuth.instance.userChanges().listen((user) {
    if (user != null) {
        _loginState = UserStatus.loggedIn;
        _chatBookSubscription = FirebaseFirestore.instance
            .collection('blog')
            .orderBy('timestamp', descending: true)
            .snapshots()
            .listen((snapshot) {
        _chatBookMessages = [];
        for (final document in snapshot.docs) {
            _chatBookMessages.add(
            LetUsChatMessage(
                name: document.data()['name'] as String,
                title: document.data()['title'] as String,
                body: document.data()['body'] as String,
            ),
            );
        }
        notifyListeners();
        });
    } else {
        _loginState = UserStatus.loggedOut;
        _chatBookMessages = [];
    }
    notifyListeners();
    });
}

UserStatus _loginState = UserStatus.loggedOut;
UserStatus get loginState => _loginState;

String? _email;
String? get email => _email;

StreamSubscription<QuerySnapshot>? _chatBookSubscription;
StreamSubscription<QuerySnapshot>? get chatBookSubscripti\
on =>
    _chatBookSubscription;
List<LetUsChatMessage> _chatBookMessages = [];
List<LetUsChatMessage> get chatBookMessages => _chatBookM\
essages;

void startLoginFlow() {
    _loginState = UserStatus.emailAddress;
    notifyListeners();
}

Future<void> verifyEmail(
    String email,
    void Function(FirebaseAuthException e) errorCallback,
) async {
    try {
    var methods =
        await FirebaseAuth.instance.fetchSignInMethodsFor\
Email(email);
    if (methods.contains('password')) {
        _loginState = UserStatus.password;
    } else {
        _loginState = UserStatus.register;
    }
    _email = email;
    notifyListeners();
    } on FirebaseAuthException catch (e) {
    errorCallback(e);
    }
}

Future<void> signInWithEmailAndPassword(
    String email,
    String password,
    void Function(FirebaseAuthException e) errorCallback,
) async {
    try {
    await FirebaseAuth.instance.signInWithEmailAndPasswor\
d(
        email: email,
        password: password,
    );
    } on FirebaseAuthException catch (e) {
    errorCallback(e);
    }
}

void cancelRegistration() {
    _loginState = UserStatus.emailAddress;
    notifyListeners();
}

Future<void> registerAccount(
    String email,
    String displayName,
    String password,
    void Function(FirebaseAuthException e) errorCallback)\
 async {
    try {
    var credential = await FirebaseAuth.instance
        .createUserWithEmailAndPassword(email: email, pas\
sword: password);
    await credential.user!.updateDisplayName(displayName);
    } on FirebaseAuthException catch (e) {
    errorCallback(e);
    }
}

void signOut() {
    FirebaseAuth.instance.signOut();
}

Future<DocumentReference> addMessageToChatBook(String tit\
le, String body) {
    if (_loginState != UserStatus.loggedIn) {
    throw Exception('Must be logged in');
    }

    return FirebaseFirestore.instance.collection('blog').\
add(<String, dynamic>{
    'title': title,
    'body': body,
    'timestamp': DateTime.now().millisecondsSinceEpoch,
    'name': FirebaseAuth.instance.currentUser!.displayNam\
e,
    'userId': FirebaseAuth.instance.currentUser!.uid,
    });
}
}
```

At the same time we also define the nature of the Flutter 3.0 application.

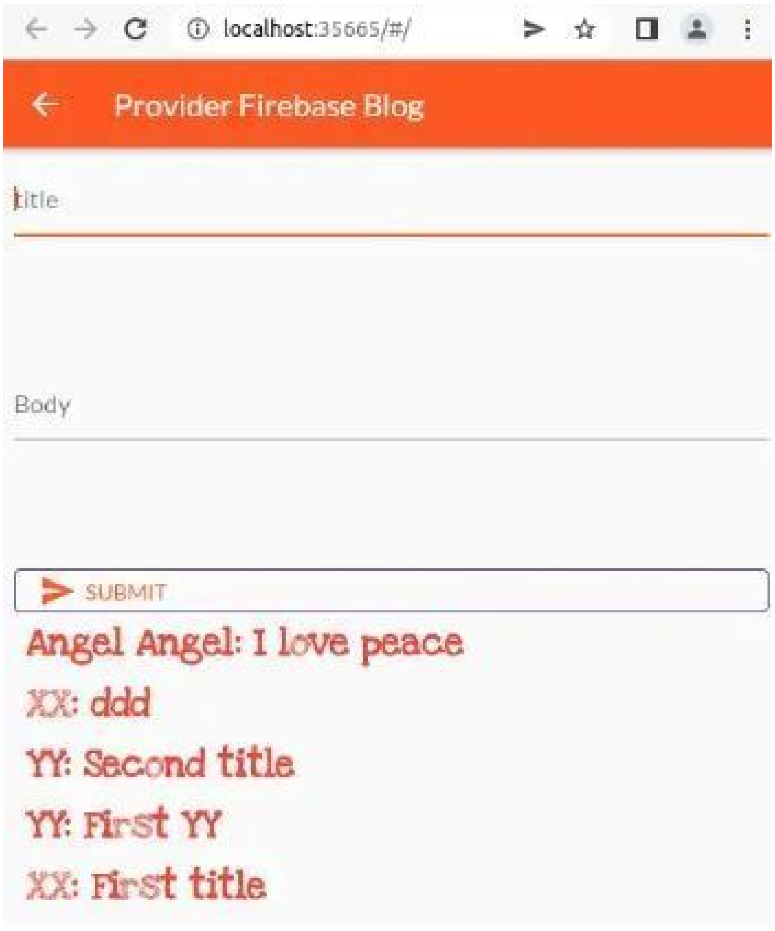
While adding the project in Firebase, we need to choose the platform. According to that it creates the API keywords.

Now in our code we can supply those keywords and connect with Firebase.

```
import 'package:firebase_core/firebase_core.dart' show Fi\
rebaseOptions;
import 'package:flutter/foundation.dart'
    show defaultTargetPlatform, kIsWeb, TargetPlatform;

/// we need to specify the associated values according to\
 the platform
/// we're using, like in this case, we have chosen web pl\
atform
/// in Firebase console
///
class DefaultFirebaseOptions {
static FirebaseOptions get currentPlatform {
    if (kIsWeb) {
    //return web;
    }
    // ignore: missing_enum_constant_in_switch
    switch (defaultTargetPlatform) {
    case TargetPlatform.android:
        return android;
    case TargetPlatform.iOS:
        return ios;
    case TargetPlatform.macOS:
        return macos;
    }

    throw UnsupportedError(
    'DefaultFirebaseOptions are not supported for this pl\
atform.',
    );
}

static const FirebaseOptions web = FirebaseOptions(
    apiKey: "****************************************",
    appId: "******************************************",
    messagingSenderId: "***********",
    projectId: "*********",
);

static const FirebaseOptions android = FirebaseOptions(
    apiKey: '',
    appId: '',
    messagingSenderId: '',
    projectId: '',
);

static const FirebaseOptions ios = FirebaseOptions(
    apiKey: '',
    appId: '',
    messagingSenderId: '',
    projectId: '',
);

static const FirebaseOptions macos = FirebaseOptions(
    apiKey: '',
    appId: '',
    messagingSenderId: '',
    projectId: '',
);
}
```

Certainly, in our case, we have chosen the web platform.

As a result, now we can see the titles of the blogs.



Angel Angel: I love peace
XX: ddd
YY: Second title
YY: First YY
XX: First title

Figure 12.9 – Flutter web 3.0 all posts displaying titles

Flutter web 3.0 and Firestore database

As our Flutter web 3.0 Firebase Provider blog app has built a connection with Firestore database, users can insert data.

Once the data is into the external database, it is not difficult to navigate to the blog detail page.

```
import 'package:flutter/material.dart';
import 'package:google_fonts/google_fonts.dart';
import 'package:provider/provider.dart';

import '../controller/all_widgets.dart';
import '../controller/authenticate_to_firebase.dart';
import '../model/state_of_application.dart';
import 'let_us_chat.dart';

class ChatHomePage extends StatelessWidget {
const ChatHomePage({Key? key}) : super(key: key);

@override
Widget build(BuildContext context) {
    return Scaffold(
    appBar: AppBar(
        title: const Text('Provider Firebase Blog'),
    ),
    body: ListView(
        children: <Widget>[
        Image.network(
            'https://cdn.pixabay.com/photo/2018/03/24/00/\
36/girl-3255402_960_720.png',
            width: 250,
            height: 250,
            fit: BoxFit.cover,
        ),
        const SizedBox(height: 8),
        Consumer<StateOfApplication>(
            builder: (context, appState, _) => Authentica\
tionForFirebase(
            email: appState.email,
            loginState: appState.loginState,
            startLoginFlow: appState.startLoginFlow,
            verifyEmail: appState.verifyEmail,
            signInWithEmailAndPassword: appState.signInWi\
thEmailAndPassword,
            cancelRegistration: appState.cancelRegistrati\
on,
            registerAccount: appState.registerAccount,
            signOut: appState.signOut,
            ),
        ),
        const Paragraph(
            'Hi, I\'m Angel, I\'m Inviting you to write B\
logs. Please join me.'),
        const Divider(
            height: 8,
            thickness: 1,
            indent: 8,
            endIndent: 8,
            color: Colors.grey,
        ),
        const Header('Write your Blog'),
        const Paragraph(
            'Join your friends and write your blog!',
        ),
        Consumer<StateOfApplication>(
            builder: (context, appState, _) => Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
                if (appState.loginState == UserStatus.log\
gedIn) ...[
                TextButton(
                    onPressed: () {
                    Navigator.push(
                        context,
                        MaterialPageRoute(
                        builder: (context) => LetUsChat(
                            addMessageOne: (title, body) \
=>
                                appState.addMessageToChat\
Book(title, body),
                            messages: appState.chatBookMe\
ssages,
                        ),
                        ),
                    );
                    },
                    child: Text(
                    'Let\'s Blog',
                    style: GoogleFonts.laila(
                        fontSize: 30.0,
                        fontWeight: FontWeight.bold,
                        color: Colors.yellow,
                        backgroundColor: Colors.red,
                    ),
                    ),
                ),
                ],
            ],
            ),
        ),
        ],
    ),
    );
}
}
```

Before we have done a lot of such things.

Sending data through the class constructor

But it would not be possible, if we had not used ChangeNotifierProvider, ChangeNotifier, and Consumer from the Provider package.

After all, we are sending data from the parent widget to the child widget. In addition, we need to manage the state of the application.

Incidentally the child widget will now consume the data and display the blog detail page.

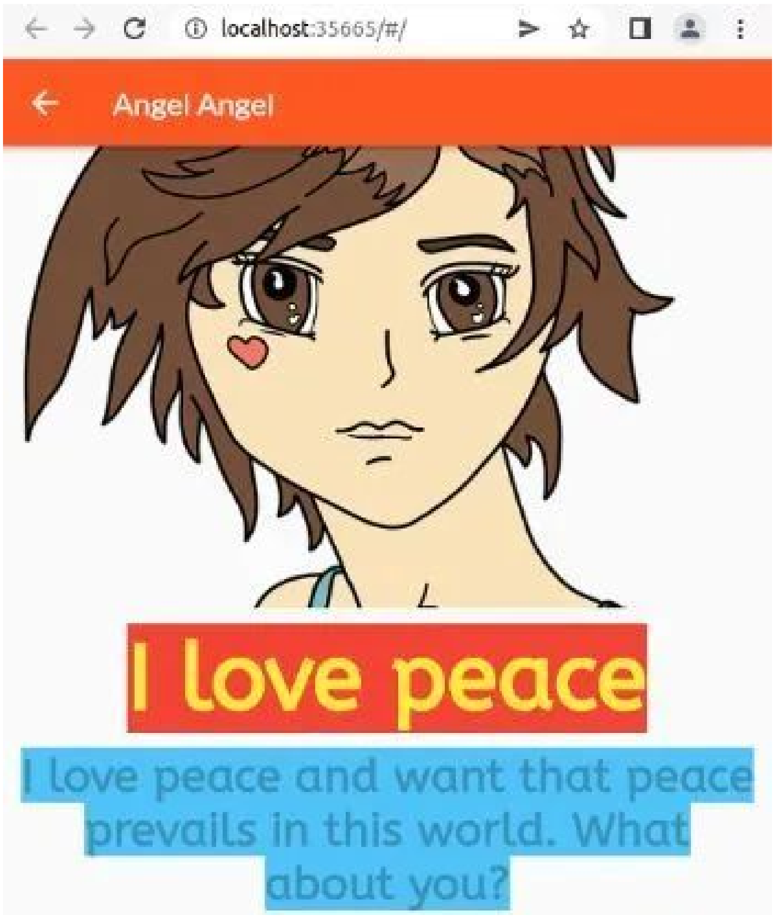
```
import 'dart:async';

import 'package:flutter/material.dart';
import 'package:google_fonts/google_fonts.dart';
import 'package:provider/provider.dart';

import '../controller/all_widgets.dart';
import '../controller/authenticate_to_firebase.dart';
import '../model/state_of_application.dart';

class LetUsChatMessage {
LetUsChatMessage({
    required this.name,
    required this.title,
    required this.body,
});
final String name;
final String title;
final String body;
}

class LetUsChat extends StatefulWidget {
```

```
const LetUsChat({
    required this.addMessageOne,
    required this.messages,
});
final FutureOr<void> Function(String messageOne, String m\
essageTwo)
    addMessageOne;
final List<LetUsChatMessage> messages;

@override
_LetUsChatState createState() => _LetUsChatState();
}

class _LetUsChatState extends State<LetUsChat> {
final _formKey = GlobalKey<FormState>(debugLabel: '_LetUs\
Blog');
final _controllerOne = TextEditingController();
final _controllerTwo = TextEditingController();

@override
Widget build(BuildContext context) {
    return Scaffold(
    appBar: AppBar(
        title: const Text('Provider Firebase Blog'),
    ),
    body: Padding(
        padding: const EdgeInsets.all(8.0),
        child: Form(
        key: _formKey,
        child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
            Expanded(
                child: TextFormField(
                controller: _controllerOne,
                decoration: const InputDecoration(
                    hintText: 'title',
                ),
                validator: (value) {
                    if (value == null || value.isEmpty) {
                    return 'Enter your message to continu\
e';
                    }
                    return null;
                },
                ),
            ),
            const SizedBox(width: 10),
            Expanded(
                child: TextFormField(
                controller: _controllerTwo,
                decoration: const InputDecoration(
                    hintText: 'Body',
                ),
                validator: (value) {
                    if (value == null || value.isEmpty) {
                    return 'Enter your message to continu\
e';
                    }
                    return null;
                },
                ),
            ),
            const SizedBox(width: 10),
            StyledButton(
                onPressed: () async {
                if (_formKey.currentState!.validate()) {
                    await widget.addMessageOne(
                        _controllerOne.text, _controllerT\
wo.text);
```

```
                    _controllerOne.clear();
                    _controllerTwo.clear();
                }
                },
                child: Row(
                children: const [
                    Icon(Icons.send),
                    SizedBox(width: 6),
                    Text('SUBMIT'),
                ],
                ),
            ),
            for (var message in widget.messages)
                GestureDetector(
                onTap: () {
                    Navigator.push(
                    context,
                    MaterialPageRoute(
                        builder: (context) => BlogDetailS\
creen(
                        name: message.name,
                        title: message.title,
                        body: message.body,
                        ),
                    ),
                    );
                },
                child: Paragraph('${message.name}: ${mess\
age.title}'),
                ),
            ],
        ),
        ),
    ),
    );
```

```
}
} // LetUsChat state ends

class BlogDetailScreen extends StatelessWidget {
// static const routename = '/product-detail';

const BlogDetailScreen({
    Key? key,
    required this.name,
    required this.title,
    required this.body,
}) : super(key: key);
final String name;
final String title;
final String body;

@override
Widget build(BuildContext context) {
    return Scaffold(
    appBar: AppBar(
        title: Text(name),
    ),
    body: SingleChildScrollView(
        child: Consumer<StateOfApplication>(
        builder: (context, appState, _) => Column(
            children: <Widget>[
            if (appState.loginState == UserStatus.loggedI\
n) ...[
                SizedBox(
                height: 300,
                width: double.infinity,
                child: Image.network(
                    'https://cdn.pixabay.com/photo/2018/0\
3/24/00/36/girl-3255402_960_720.png',
                    width: 250,
                    height: 250,
                    fit: BoxFit.cover,
                ),
                ),
                const SizedBox(height: 10),
                Text(
                title,
                style: GoogleFonts.aBeeZee(
                    fontSize: 60.0,
                    fontWeight: FontWeight.bold,
                    color: Colors.yellow,
                    backgroundColor: Colors.red,
                ),
                ),
                const SizedBox(
                height: 10,
                ),
                Container(
                padding: const EdgeInsets.symmetric(horiz\
ontal: 10),
                width: double.infinity,
                child: Text(
                    body,
                    textAlign: TextAlign.center,
                    softWrap: true,
                    style: GoogleFonts.aBeeZee(
                    fontSize: 30.0,
                    fontWeight: FontWeight.bold,
                    color: Colors.black26,
                    backgroundColor: Colors.lightBlue[300\
],
                    ),
                ),
                ),
                const SizedBox(
                height: 10,
                ),
              ],
            ],
          ),
        ),
      ),
    );
  }
}
```

Finally we can see the blog that Angel has just posted.



I love peace

I love peace and want that peace prevails in this world. What about you?

Figure 12.10 – Flutter web 3.0 blog detail page

At the same time the same blog post is present at the Firestore database collection.

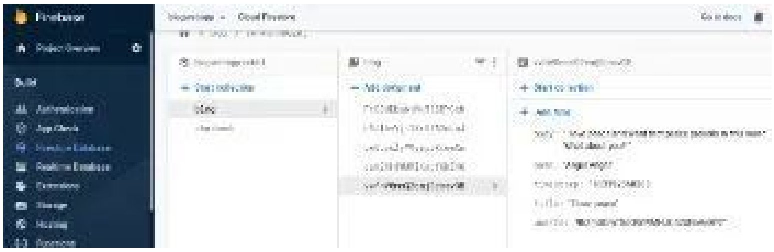


Figure 12.11 – Flutter web 3.0 Firebase Firestore database shows the blog posts

However, we are not happy with the design part. Therefore we will introduce Material Design 3 and change the entire look of the Flutter web 3.0 Firebase Provider blog app.

So stay tuned.

Text Form Field Flutter size, how to increase in web app

How do we increase the size of the text form field size in a Flutter app?

We need to increase the size because we've been building a Flutter Firebase, Provider web app where users write blogs.

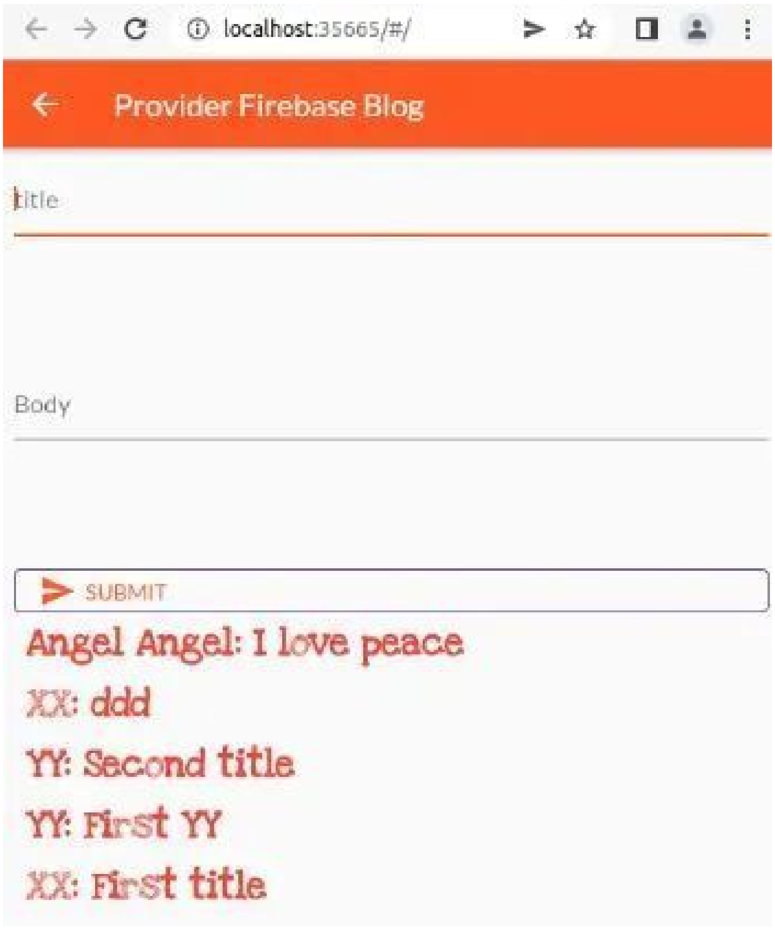
As a result, as an interface, on the screen users must give some inputs through the TextFormField.

However we're not happy with the default design of the TextFormField.

For that reason we need to tweak the code and find a solution where the size of the title field is small.

But the size of the TextFormField where users will write the content, will be bigger than the title field.

Initially it looked as follows.

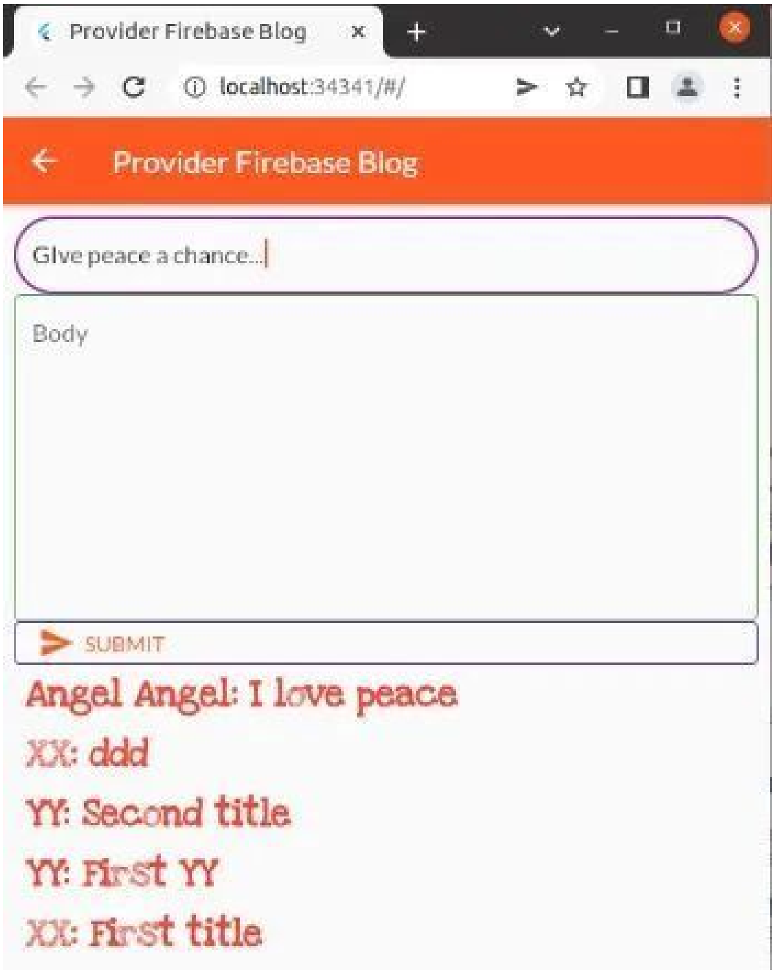


Angel Angel: I love peace
XX: ddd
YY: Second title
YY: First YY
XX: First title

Figure 12.12 – Flutter web 3.0 all posts displaying titles

As we see both the TextFields look the same. In addition, they are not user friendly.

Therefore we want to make them look as follows where the TextField for content will be bigger than the title field.



Angel Angel: I love peace
XX: ddd
YY: Second title
YY: First YY
XX: First title

Figure 12.13 – Text form field flutter size matters

Certainly it looks better than before, all the same we can make it more attractive.

The interesting part of this TextFormField is when users tap in, it changes its border color from green to blue.

Not only that, it also changes its shape.

As a whole it gives a rich user experience.Before discussing how we do that, let's see some key concepts of a TextFormField.

Does the text form field flutter size vary?

Can we vary the size of the text form field flutter?

Is there any kind of in-built properties that will help us in achieving this goal?

A TextFormField is a FormField that contains a TextField.

Why do we need this widget?

To give some inputs to the flutter app.

How does it work? What are the basic principles?

Firstly, it's a convenience widget that wraps a TextField widget in a FormField.

Although we don't always need a Form ancestor, in our case, we have a Form ancestor.

Why?

Because the Form ancestor makes it easier to save, reset, or validate multiple fields at once.

We have done exactly the same. Because we have multiple fields like title and body of a blog. Right?

Secondly, if you plan to use the TextFormField without the Form ancestor then you can pass a GlobalKey to the constructor and use GlobalKey.currentState to save or reset the form field.

Let's see our code that will explain this further.

```
import 'dart:async';

import 'package:flutter/material.dart';
import 'package:google_fonts/google_fonts.dart';
import 'package:provider/provider.dart';

import '../controller/all_widgets.dart';
import '../controller/authenticate_to_firebase.dart';
import '../model/state_of_application.dart';

class LetUsChatMessage {
LetUsChatMessage({
    required this.name,
    required this.title,
    required this.body,
});
final String name;
final String title;
final String body;
}

class LetUsChat extends StatefulWidget {
const LetUsChat({
    required this.addMessageOne,
    required this.messages,
});
final FutureOr<void> Function(String messageOne, String m\
essageTwo)
    addMessageOne;
final List<LetUsChatMessage> messages;

@override
State<LetUsChat> createState() => _LetUsChatState();
}

```

```
class _LetUsChatState extends State<LetUsChat> {
final _formKey = GlobalKey<FormState>(debugLabel: '_LetUs\
Blog');
final _controllerOne = TextEditingController();
final _controllerTwo = TextEditingController();

@override
Widget build(BuildContext context) {
    return Scaffold(
    appBar: AppBar(
        title: const Text('Provider Firebase Blog'),
    ),
    body: Padding(
        padding: const EdgeInsets.all(8.0),
        child: Form(
        key: _formKey,
        child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
            TextFormField(
                controller: _controllerOne,
                decoration: InputDecoration(
                hintText: 'Title',
                enabledBorder: OutlineInputBorder(
                    borderRadius: BorderRadius.circular(5\
),
                    borderSide: const BorderSide(
                    color: Colors.green,
                    width: 1.0,
                    ),
                ),
                focusedBorder: OutlineInputBorder(
                    borderRadius: BorderRadius.circular(3\
0),
                    borderSide: const BorderSide(
                    color: Colors.purple,
                    width: 2.0,
                    ),
                ),
                ),
                validator: (value) {
                if (value == null || value.isEmpty) {
                    return 'Enter your message to continu\
e';
                }
                return null;
                },
            ),
            Expanded(
                child: SizedBox(
                height: 150.0,
                child: TextFormField(
                    controller: _controllerTwo,
                    maxLines: 10,
                    decoration: InputDecoration(
                    hintText: 'Body',
                    enabledBorder: OutlineInputBorder(
                        borderRadius: BorderRadius.circul\
ar(5),
                        borderSide: const BorderSide(
                        color: Colors.green,
                        width: 1.0,
                        ),
                    ),
                    focusedBorder: OutlineInputBorder(
                        borderRadius: BorderRadius.circul\
ar(30),
                        borderSide: const BorderSide(
                        color: Colors.purple,
                        width: 2.0,
                      ),
                    ),
                    ),
                    validator: (value) {
                    if (value == null || value.isEmpty) {
                        return 'Enter your message to con\
tinue';
                    }
                    return null;
                    },
                  ),
                  ),
                ),
            const SizedBox(width: 10.0),
            StyledButton(
                onPressed: () async {
                if (_formKey.currentState!.validate()) {
                    await widget.addMessageOne(
                        _controllerOne.text, _controllerT\
wo.text);
                    _controllerOne.clear();
                    _controllerTwo.clear();
                }
                },
                child: Row(
                children: const [
                    Icon(Icons.send),
                    SizedBox(width: 6),
                    Text('SUBMIT'),
                ],
                ),
            ),
            for (var message in widget.messages)
                GestureDetector(
                onTap: () {
```

```
                              Navigator.push(
                              context,
                              MaterialPageRoute(
                                  builder: (context) => BlogDetailS\
creen(
                                  name: message.name,
                                  title: message.title,
                                  body: message.body,
                                  ),
                              ),
                              );
                        },
                        child: Paragraph('${message.name}: ${mess\
age.title}'),
                        ),
                  ],
            ),
            ),
      ),
      );
}
}
```

How did we increase the text form field in flutter? Simple. We've used a SizedBox Widget as its immediate parent and made the height property 150.

We've specified a TextEditingController that defines the initialValue.

In addition, the StatefulWidget as an ancestor manages the controller's lifetime.

In our case, the same thing happens.

We've called TextEditingController.dispose of the TextEditingController. Because we no longer need it once users have typed in and press the submit button.

This will ensure we discard any resources used by the object.

Let’s see the code.

```
StyledButton(
                onPressed: () async {
                if (_formKey.currentState!.validate()) {
                    await widget.addMessageOne(
                        _controllerOne.text, _controllerT\
wo.text);
                    _controllerOne.clear();
                    _controllerTwo.clear();
                }
                },
                child: Row(
                children: const [
                    Icon(Icons.send),
                    SizedBox(width: 6),
                    Text('SUBMIT'),
                ],
                ),
            ),
```

Apart from controlling the size, we can also decorate the text form field in flutter.

And that certainly adds a style statement to our Firebase, Provider web app.

We’ll discuss that topic in the next section.

Theme color Flutter, how to use in web app

In Flutter 3.0 theme color we’ll use Material design 3. Certainly we will adopt the same principle in our ongoing web app.

However, we need to understand how we can configure the overall visual Theme for a MaterialApp or a widget subtree within the app.

That's the first step.

In fact, this section will explain the basics. After that we will discuss how we can implement the same principle in our web app using Material design 3.

The theme property of the MaterialApp configures the appearance of the whole app.

As an outcome, we can implement the same pattern across the whole app. Most importantly it helps us to customise the color, font and many more.

First of all, we need to fill our ThemeData widget properties with custom colours.

In addition it will help us to maintain a custom theme. At the same time, we can also change colours at one place like the following:

```
ThemeData _customTheme() {
    return ThemeData(
    accentColor: Color(0xFF442B2D),
    primaryColor: Color(0xFFFEDBD0),
    buttonColor: Color(0xFFFEDBD0),
    scaffoldBackgroundColor: Colors.white,
    cardColor: Color(0xFF883B2D),
    textSelectionTheme: TextSelectionThemeData(
    selectionColor: Color(0xFFFEDBD0),
    ),
    errorColor: Colors.red,
    buttonTheme: ThemeData.light().buttonTheme.copyWith(
    buttonColor: Color(0xFFFEDBD0),
    colorScheme: ThemeData.light().colorScheme.copyWith(
    secondary: Color(0xFF442B2D),
    ),
    ),
    buttonBarTheme: ThemeData.light().buttonBarTheme.copy\
With(
    buttonTextTheme: ButtonTextTheme.accent,
    ),
    primaryIconTheme: ThemeData.light().primaryIconTheme.\
copyWith(
    color: Color(0xFF442B2D),
    ),
    );
    }
```

Wherever we keep this custom colours theme we need to call this method inside MaterialApp.

Because MaterialApp theme property returns ThemeData constructor, we can use the custom color theme method.

```
class MaterialDesignThemeControl extends StatelessWidget {
    const MaterialDesignThemeControl({Key? key}) : super(\
key: key);

    @override
    Widget build(BuildContext context) {
    // ignore: todo
    // TODO: building custom theme that'll control color \
and text
    return MaterialApp(
    title: 'Material Design Theme Control',
    home: MaterialDesignCustomTheme(),
    theme: _customTheme(),
    debugShowCheckedModeBanner: false,
    );
    }}
```

The following line is important here.

```
theme: _customTheme(),
```

As a result, we can have this output in our virtual device.

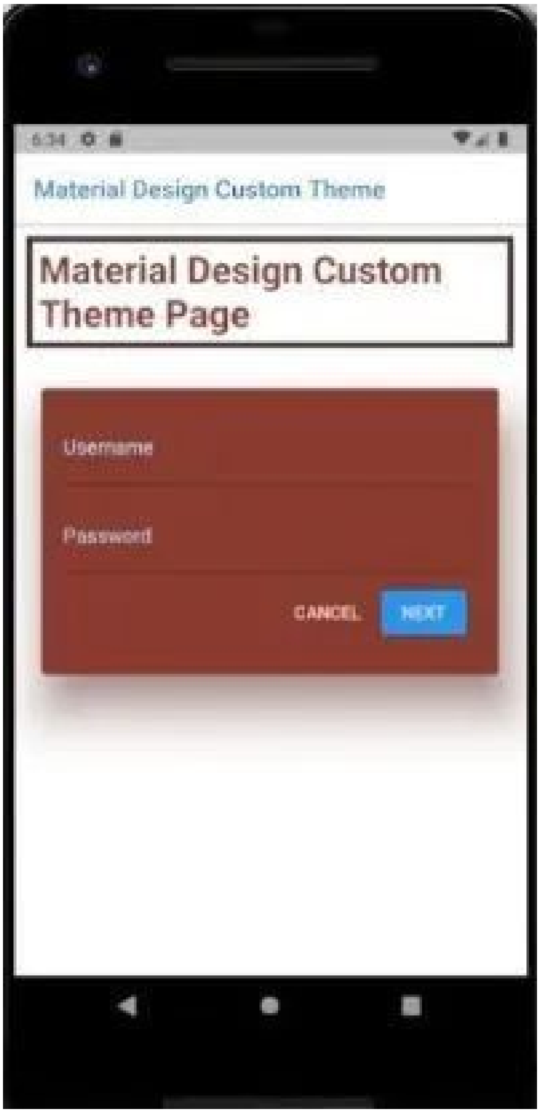

Material Design Custom Theme
Material Design Custom Theme Page
Username
Password
CANCEL
NEXT

Figure 12.14 – Theme color Flutter example

Consequently, changing colors in Flutter using a custom theme becomes easy now.

In one single file, you can add more functionalities.

We always want to make things simple. Isn't it?

Moreover, from MaterialApp, now you can control the color theme throughout the app.

Theme color across the flutter app

We can control AppBar color through Scaffold widget.

```
class MaterialDesignCustomTheme extends StatelessWidget {
    const MaterialDesignCustomTheme({Key? key}) : super(k\
ey: key);

    @override
    Widget build(BuildContext context) {
    return Scaffold(
    appBar: AppBar(
    title: Text(
    'Material Design Custom Theme',
    style: TextStyle(
    fontSize: 20,
    color: Theme.of(context).primaryColorDark,
    ),
    ),
    backgroundColor: Theme.of(context).scaffoldBackground\
Color,
    ),
    body: CustomPage(),
    );
    }
    }
```

After that, we can use a CustomPage widget where we can build the page using our custom theme.

```
class MaterialDesignCustomTheme extends StatelessWidget {
    const MaterialDesignCustomTheme({Key? key}) : super(k\
ey: key);

    @override
    Widget build(BuildContext context) {
    return Scaffold(
    appBar: AppBar(
    title: Text(
    'Material Design Custom Theme',
    style: TextStyle(
    fontSize: 20,
    color: Theme.of(context).primaryColorDark,
    ),
    ),
    backgroundColor: Theme.of(context).scaffoldBackground\
Color,
    ),
    body: CustomPage(),
    );
    }
    }

    class CustomPage extends StatelessWidget {
    const CustomPage({Key? key}) : super(key: key);

    @override
    Widget build(BuildContext context) {
    return ListView(
    children: [
    Container(
    margin: EdgeInsets.all(10),
    padding: EdgeInsets.all(5),
    decoration: BoxDecoration(
    border: Border.all(
    width: 5,
    color: Theme.of(context).accentColor,
    ),
    ),
    child: Text(
    'Material Design Custom Theme Page',
    style: TextStyle(
    fontSize: 30,
    fontWeight: FontWeight.bold,
    color: Theme.of(context).cardColor,
    ),
    ),
    ),
    Container(
    margin: EdgeInsets.all(10),
    padding: EdgeInsets.all(8),
    child: Card(
    elevation: 30,
    shadowColor: Theme.of(context).cardColor,
    child: Container(
    margin: EdgeInsets.all(10),
    padding: EdgeInsets.all(8),
    child: Column(
    children: [
    TextField(
    decoration: InputDecoration(
    labelText: 'Username',
    labelStyle: TextStyle(
    color: Theme.of(context).primaryColorLight,
    ),
    ),
    ),
    SizedBox(height: 12.0),
    TextField(
    decoration: InputDecoration(
    labelText: 'Password',
    labelStyle: TextStyle(
    color: Theme.of(context).primaryColorLight,
    ),
    ),
    obscureText: true,
    ),
    ButtonBar(
    children: <Widget>[
    TextButton(
    child: Text(
    'CANCEL',
    style: TextStyle(
    color: Theme.of(context).buttonColor,
    ),
    ),
    onPressed: () {},
    ),
    ElevatedButton(
    child: Text(
    'NEXT',
    style: TextStyle(
    color: Theme.of(context).buttonColor,
    ),
    ),
    onPressed: () {},
    ),
    ],
    ),
    ],
    ),
    ),
    ),
    )
    ],
    );
  }
}
```

We need to remember a few points here. The Theme.of method can help the other widgets to obtain the custom theme.

Material components typically depend exclusively on the colorScheme and textTheme.

We'll discuss that topic in the next section when we'll implement Material design 3.

Material design 3 Flutter : A Light Theme

What is Material design 3? How would we apply this theme to new Flutter 3.0? Well, let's try to answer the questions one after another.

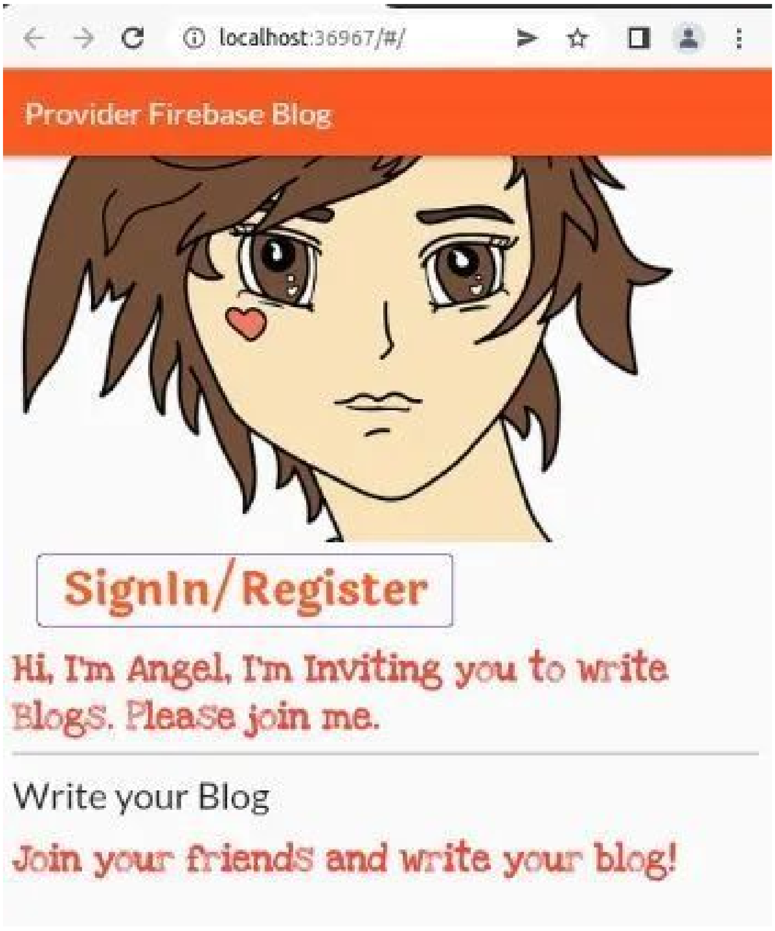
Firstly, Material design 3 is the next generation design language.

It's good news that Flutter has supported Material design 3 from the very beginning. As a result, what we have seen till now, designed in Material design 2.

We've been building a web app where users can write their blogs and share with other members. However, initially we have used the Material design 2 theme.

As a result, our old Firebase, Provider web app will have a different look.

Let's see the homepage with the old theme first.



Hi, I'm Angel, I'm Inviting you to write Blogs. Please join me.

Write your Blog

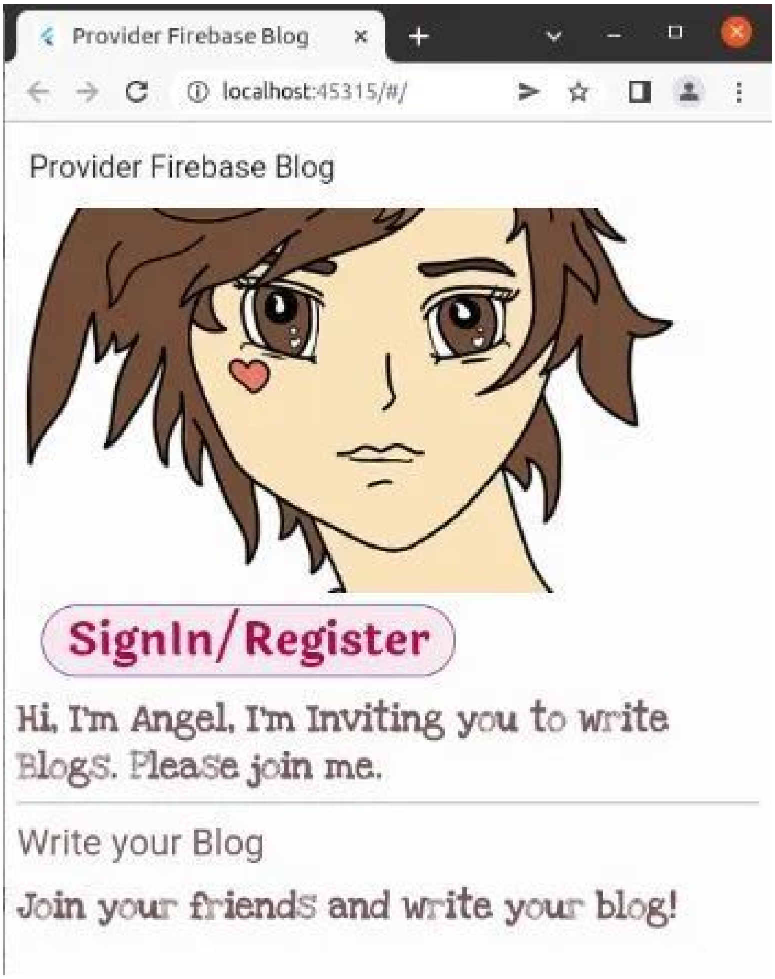
Join your friends and write your blog!

Figure 12.15 – Flutter web 3.0 homepage

To apply Material design 2 theme, we used the following code.

```
return MaterialApp(
    title: 'Provider Firebase Blog',
    debugShowCheckedModeBanner: false,
    theme: ThemeData(
        buttonTheme: Theme.of(context).buttonTheme.copyWi\
th(
            highlightColor: Colors.black45,
            ),
        primarySwatch: Colors.deepOrange,
        textTheme: GoogleFonts.latoTextTheme(
        Theme.of(context).textTheme,
        ),
        visualDensity: VisualDensity.adaptivePlatformDens\
ity,
    ),
    home: const ChatHomePage(),
    );
```

However, with the new Material design 3 theme, the homepage will look different now.



Hi, I'm Angel, I'm Inviting you to write Blogs. Please join me.

Write your Blog

Join your friends and write your blog!

Figure 12.16 – Flutter 3.0 Material 3 Theme

Not only on the home page, the new Material design 3 theme will reflect in each page, as an outcome.

We've chosen the light theme.

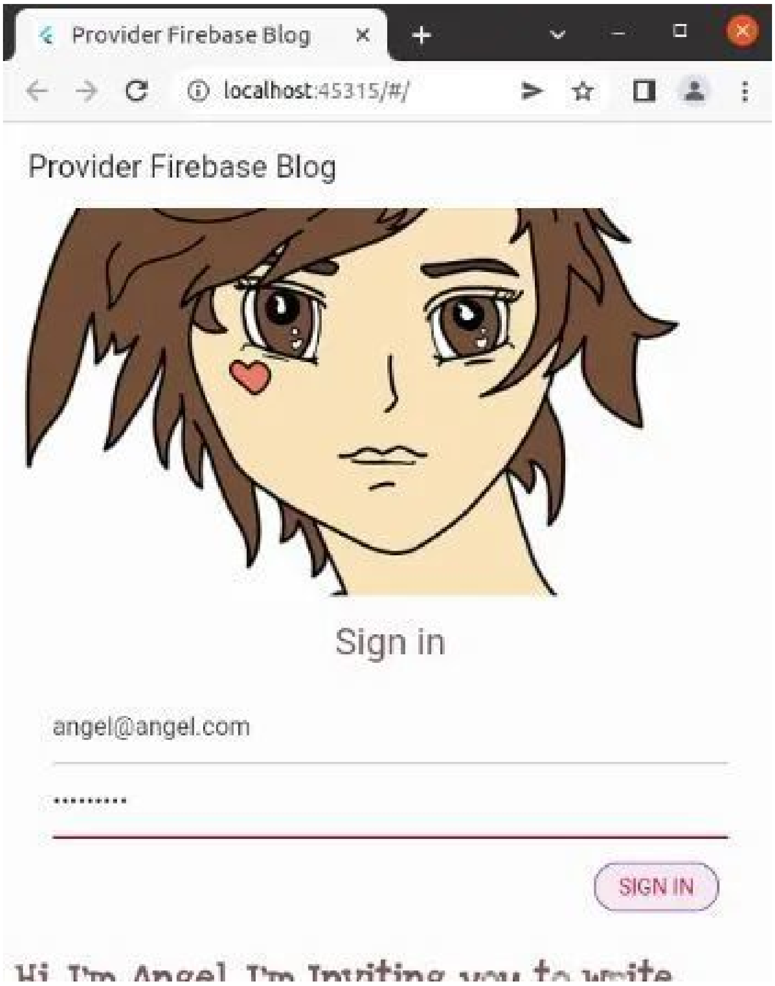


Figure 12.17 – Flutter 3.0 Material 3 Theme changes look

Finally, after signing in, users can click the “Let’s Blog” button and join other members.

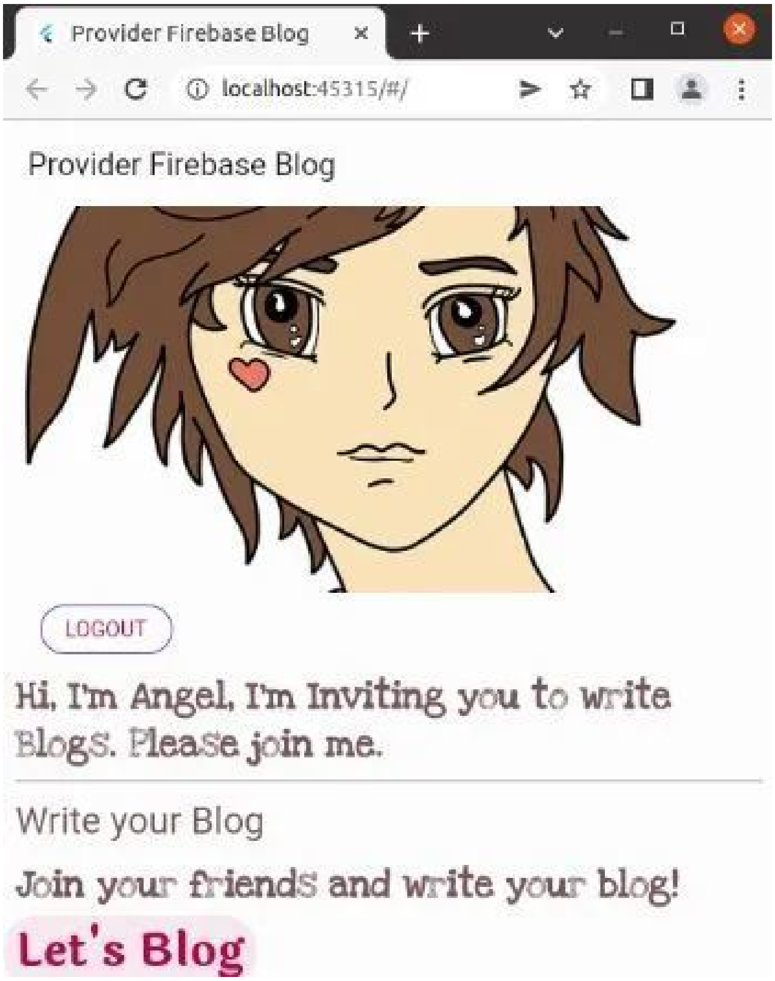

Provider Firebase Blog
localhost:45315/#/
Provider Firebase Blog
LOGOUT
Hi, I'm Angel, I'm Inviting you to write Blogs. Please join me.
Write your Blog
Join your friends and write your blog!
Let's Blog

Figure 12.18 – Flutter 3.0 Material 3 Theme design change

In the next section we'll look at how we have changed Material design from 2 to 3.

Moreover, we will also learn Material design 3 in Flutter 3.0 great detail.

So stay tuned.

Material 3 Flutter : A Dark Theme in Web App

We've been building a Firebase, Firestore and Provider based web app where we have already used Material design 3.

Firstly, what is Material design 3? It's the next generation design that will rule the cross platform application world.

Secondly, in our previous section we have changed the look of the existing web app. However, we used the light mode.

Finally, we want to change it to dark mode and see how it looks.

Besides, we want to show the code and the implementation.

However, it's always good to look back and recapitulate.

Therefore, we must remember why we use Flutter?

We use Flutter for building beautiful applications for mobile, web, desktop, and embedded devices from a single codebase. Right?

In addition, Flutter always implements the Material Design guidelines.

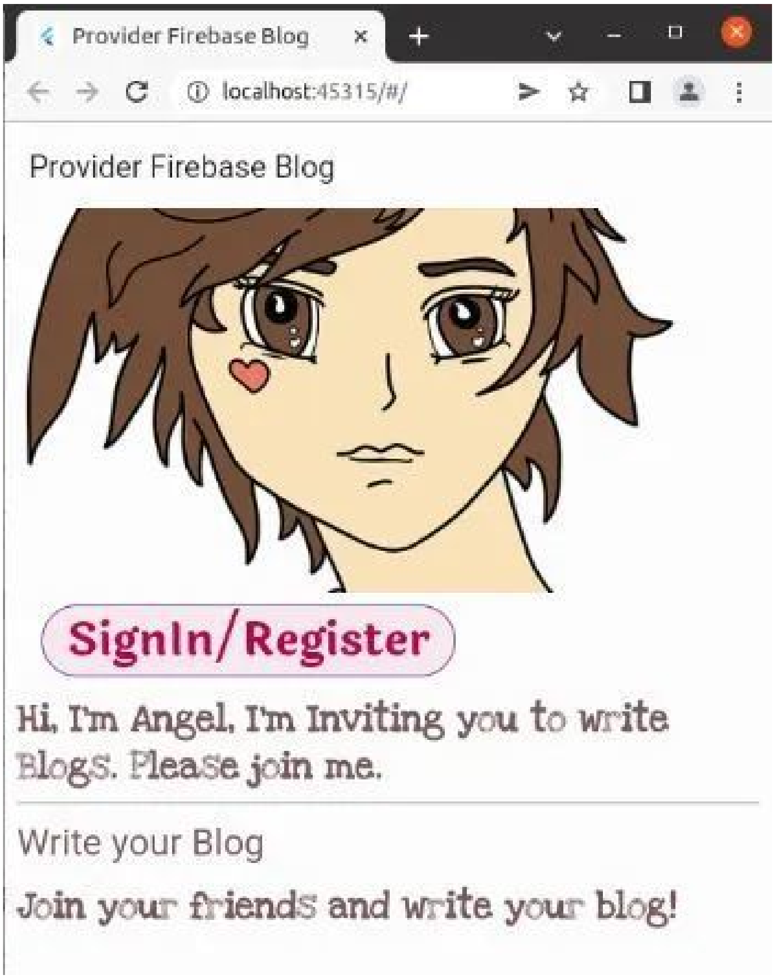
Moreover, there are lots of Material Widgets that implement these guidelines.

Let's take a quick look at the list of widgets that can implement Material 3.

```
App structure and navigation
Buttons
Input and selections
Dialogs, alerts, and panels
Information displays
Layout
```

You can get more widgets in the widget catalog.

Let’s take a look at the previous design which used the light mode.



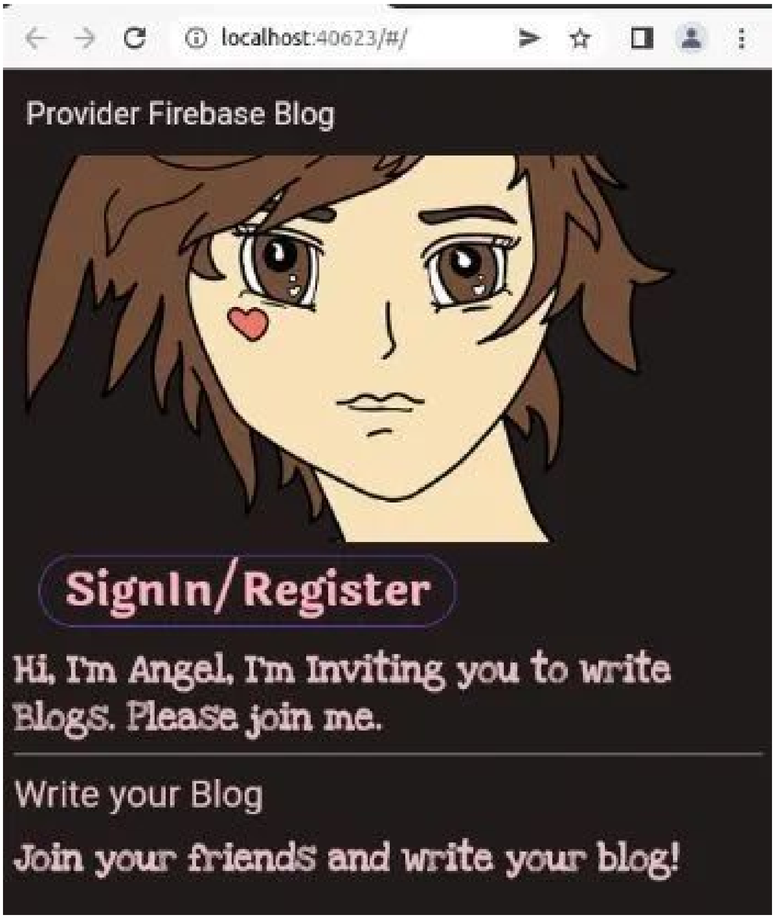
Hi, I'm Angel, I'm Inviting you to write Blogs. Please join me.

Write your Blog

Join your friends and write your blog!

Figure 12.19 – Flutter 3.0 Material 3 Theme

But the design changes when we use dark mode. It no longer remains the same.



Hi, I'm Angel, I'm Inviting you to write Blogs. Please join me.

Write your Blog

Join your friends and write your blog!

Figure 12.20 – Flutter Material 3 Web App

Firstly, we have used a Theme Provider to customize our theme.

Secondly, let's watch the code.

```
import 'dart:math';

import 'package:flutter/material.dart';
import 'package:material_color_utilities/material_color_u\
tilities.dart';

class ThemeSettingChange extends Notification {
ThemeSettingChange({required this.settings});
final ThemeSettings settings;
}

class ThemeProvider extends InheritedWidget {
const ThemeProvider(
    {super.key,
    required this.settings,
    required this.lightDynamic,
    required this.darkDynamic,
    required super.child});

final ValueNotifier<ThemeSettings> settings;
final ColorScheme? lightDynamic;
final ColorScheme? darkDynamic;

Color custom(CustomColor custom) {
    if (custom.blend) {
    return blend(custom.color);
    } else {
    return custom.color;
    }
}

Color blend(Color targetColor) {
    return Color(
        Blend.harmonize(targetColor.value, settings.value\
.sourceColor.value));
```

```
}

Color source(Color? target) {
    Color source = settings.value.sourceColor;
    if (target != null) {
    source = blend(target);
    }
    return source;
}

ColorScheme colors(Brightness brightness, Color? targetCo\
lor) {
    final dynamicPrimary = brightness == Brightness.light
        ? lightDynamic?.primary
        : darkDynamic?.primary;
    return ColorScheme.fromSeed(
    seedColor: dynamicPrimary ?? source(targetColor),
    brightness: brightness,
    );
}

ShapeBorder get shapeMedium => RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(8),
    );

CardTheme cardTheme() {
    return CardTheme(
    elevation: 0,
    shape: shapeMedium,
    clipBehavior: Clip.antiAlias,
    );
}

ListTileThemeData listTileTheme(ColorScheme colors) {
    return ListTileThemeData(
    shape: shapeMedium,
    selectedColor: colors.secondary,
    );
}

AppBarTheme appBarTheme(ColorScheme colors) {
    return AppBarTheme(
    elevation: 0,
    backgroundColor: colors.surface,
    foregroundColor: colors.onSurface,
    );
}

TabBarTheme tabBarTheme(ColorScheme colors) {
    return TabBarTheme(
    labelColor: colors.secondary,
    unselectedLabelColor: colors.onSurfaceVariant,
    indicator: BoxDecoration(
        border: Border(
        bottom: BorderSide(
            color: colors.secondary,
            width: 2,
        ),
        ),
    ),
    );
}

BottomAppBarTheme bottomAppBarTheme(ColorScheme colors) {
    return BottomAppBarTheme(
    color: colors.surface,
    elevation: 0,
    );
}

```

```
BottomNavigationBarThemeData bottomNavigationBarTheme(Col\
orScheme colors) {
    return BottomNavigationBarThemeData(
    type: BottomNavigationBarType.fixed,
    backgroundColor: colors.surfaceVariant,
    selectedItemColor: colors.onSurface,
    unselectedItemColor: colors.onSurfaceVariant,
    elevation: 0,
    landscapeLayout: BottomNavigationBarLandscapeLayout.c\
entered,
    );
}

NavigationRailThemeData navigationRailTheme(ColorScheme c\
olors) {
    return const NavigationRailThemeData();
}

DrawerThemeData drawerTheme(ColorScheme colors) {
    return DrawerThemeData(
    backgroundColor: colors.surface,
    );
}

ThemeData light([Color? targetColor]) {
    final colorScheme = colors(Brightness.light, targetCo\
lor);
    return ThemeData.light().copyWith(
    //pageTransitionsTheme: pageTransitionsTheme,
    colorScheme: colorScheme,
    appBarTheme: appBarTheme(colorScheme),
    cardTheme: cardTheme(),
    listTileTheme: listTileTheme(colorScheme),
    bottomAppBarTheme: bottomAppBarTheme(colorScheme),
    bottomNavigationBarTheme: bottomNavigationBarTheme(co\
lorScheme),
    navigationRailTheme: navigationRailTheme(colorScheme),
    tabBarTheme: tabBarTheme(colorScheme),
    drawerTheme: drawerTheme(colorScheme),
    scaffoldBackgroundColor: colorScheme.background,
    useMaterial3: true,
    );
}

ThemeData dark([Color? targetColor]) {
    final colorScheme = colors(Brightness.dark, targetCol\
or);
    return ThemeData.dark().copyWith(
    colorScheme: colorScheme,
    appBarTheme: appBarTheme(colorScheme),
    cardTheme: cardTheme(),
    listTileTheme: listTileTheme(colorScheme),
    bottomAppBarTheme: bottomAppBarTheme(colorScheme),
    bottomNavigationBarTheme: bottomNavigationBarTheme(co\
lorScheme),
    navigationRailTheme: navigationRailTheme(colorScheme),
    tabBarTheme: tabBarTheme(colorScheme),
    drawerTheme: drawerTheme(colorScheme),
    scaffoldBackgroundColor: colorScheme.background,
    useMaterial3: true,
    );
}

ThemeMode themeMode() {
    return settings.value.themeMode;
}

ThemeData theme(BuildContext context, [Color? targetColor\
]) {
    final brightness = MediaQuery.of(context).platformBri\
ghtness;
    return brightness == Brightness.light
        ? light(targetColor)
        : dark(targetColor);
}

static ThemeProvider of(BuildContext context) {
    return context.dependOnInheritedWidgetOfExactType<The\
meProvider>()!;
}

@override
bool updateShouldNotify(covariant ThemeProvider oldWidget\
) {
    return oldWidget.settings != settings;
}
}

class ThemeSettings {
ThemeSettings({
    required this.sourceColor,
    required this.themeMode,
});

final Color sourceColor;
final ThemeMode themeMode;
}

Color randomColor() {
return Color(Random().nextInt(0xFFFFFFFF));
}

// Custom Colors
const linkColor = CustomColor(
name: 'Link Color',
color: Color(0xFF00B0FF),
);

class CustomColor {
const CustomColor({
    required this.name,
    required this.color,
    this.blend = true,
});

final String name;
final Color color;
final bool blend;

Color value(ThemeProvider provider) {
    return provider.custom(this);
}
}
```

By the way, we have highlighted the dark ThemeData section so that you can follow along with the code.

But, how will we use the custom Material 3 and implement the design to our web app?

Material 3 Flutter : Parent and Child

Probably you may have noticed how we have extended the Theme Provider class.

Because it extends an Inherited Widget.

```
class ThemeProvider extends InheritedWidget {
const ThemeProvider(
    {super.key,
    required this.settings,
    required this.lightDynamic,
    required this.darkDynamic,
    required super.child});

final ValueNotifier<ThemeSettings> settings;
...
```

As a result, any child theme can use the theme. To do that we will create an instance at the very beginning and pass it to the scoped theme object in MaterialApp.

Now any nested child widget can inherit the theme. Right?

```
import 'package:dynamic_color/dynamic_color.dart';
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'model/state_of_application.dart';

import 'model/theme.dart';
import 'view/chat_app.dart';

/// moving to second branch
final settings = ValueNotifier(ThemeSettings(
sourceColor: Colors.pink,
themeMode: ThemeMode.system,
));
void main() {
runApp(
    ChangeNotifierProvider(
    create: (context) => StateOfApplication(),
    builder: (context, _) => DynamicColorBuilder(
        builder: (lightDynamic, darkDynamic) => ThemeProv\
ider(
        lightDynamic: lightDynamic,
        darkDynamic: darkDynamic,
        settings: settings,
        child: ChatApp(),
        ),
    ),
    ),
);
}
//
```

By the way, to make this happen, we have used a package – dynamic color.

After that we have added the dependency to our “pubspec.yaml” file.

```
dependencies:
flutter:
    sdk: flutter


cupertino_icons: ^1.0.2
firebase_core: ^1.14.0
firebase_auth: ^3.3.13
cloud_firestore: ^3.1.11
google_fonts: ^2.3.1
provider: ^6.0.2
material_color_utilities: ^0.1.4
dynamic_color: ^1.1.2
```

Now we can pass the ThemeData to the MaterialApp widget and set the theme mode to dark.

```
import 'package:flutter/material.dart';
//import 'package:dynamic_color/dynamic_color.dart';

//import 'package:google_fonts/google_fonts.dart';

import '../main.dart';
import 'chat_home_page.dart';
import '../model/theme.dart';

class ChatApp extends StatelessWidget {
ChatApp({Key? key}) : super(key: key);

@override
Widget build(BuildContext context) {
    final theme = ThemeProvider.of(context);
    return MaterialApp(
    title: 'Provider Firebase Blog',
    debugShowCheckedModeBanner: false,
    theme: theme.dark(settings.value.sourceColor),
    home: const ChatHomePage(),
    );
}
}
```

You may have noticed that we have used Pink as the source color.

Certainly you can change it in your main file, where we have defined the source color.

```
final settings = ValueNotifier(ThemeSettings(
sourceColor: Colors.pink,
themeMode: ThemeMode.system,
));
void main() {
runApp(
    ChangeNotifierProvider(
....
```

Of course we will discuss and learn more about using the Material 3 Flutter design.For this step you may clone the GitHub repository.

Material 3 : Flutter Firebase, Provider Blog App Final

Material 3, which is a short form of Material Design 3, is the next generation theme for Flutter apps.

According to the Google team, it is the most expressive and adaptable design system yet.

How will you adopt Material 3, and apply a custom theme across your Flutter app, depends on you.

In other words, the Flutter 3.0 theme color we'll take a new style using Material design 3. Certainly we will adopt the same principle in our ongoing Firebase, Provider Flutter app.

However, we need to understand how we can configure the overall visual Theme for a MaterialApp or a widget subtree within the app.

We've been building a Firebase, Firestore and Provider based web app where we have already used Material design 3.

Above all, we have reached the final stage.

Let us explain how we have progressed step by step.

Firstly, we have added the dependencies to the “pubspec.yaml” file.

```
dependencies:
flutter:
    sdk: flutter


cupertino_icons: ^1.0.2
firebase_core: ^1.14.0
firebase_auth: ^3.3.13
cloud_firestore: ^3.1.11
google_fonts: ^2.3.1
provider: ^6.0.2
material_color_utilities: ^0.1.4
dynamic_color: ^1.1.2
```

The second step involves the most important step. Providing the values to the Firebase options constructor.

The API key, App ID, project ID, etc.

Since our Flutter app is cross-platform, we can provide multiple values as we have defined in our class.

```
import 'package:firebase_core/firebase_core.dart' show Fi\
rebaseOptions;
import 'package:flutter/foundation.dart'
    show defaultTargetPlatform, kIsWeb, TargetPlatform;

/// we need to specify the associated values according to\
 the platform
/// we're using, like in this case, we have chosen web pl\
atform
/// in Firebase console
///
class DefaultFirebaseOptions {
```

```
static FirebaseOptions get currentPlatform {
    if (kIsWeb) {
    return web;
    }
    // ignore: missing_enum_constant_in_switch
    switch (defaultTargetPlatform) {
    case TargetPlatform.android:
        return android;
    case TargetPlatform.iOS:
        return ios;
    case TargetPlatform.macOS:
        return macos;
    }

    throw UnsupportedError(
    'DefaultFirebaseOptions are not supported for this pl\
atform.',
    );
}

static const FirebaseOptions web = FirebaseOptions(
    apiKey: "*****************************************",
    appId: "********************************************",
    messagingSenderId: "***********",
    projectId: "*********",
);

static const FirebaseOptions android = FirebaseOptions(
    apiKey: "*****************************************",
    appId: "********************************************",
    messagingSenderId: "***********",
    projectId: "*********",
);

static const FirebaseOptions ios = FirebaseOptions(
    apiKey: '',
    appId: '',
    messagingSenderId: '',
    projectId: '',
);

static const FirebaseOptions macos = FirebaseOptions(
    apiKey: '',
    appId: '',
    messagingSenderId: '',
    projectId: '',
);
}
```

Next, we'll use Material 3. Therefore, we will write a custom theme provider class that extends Inherited widget.

Why have we used the Inherited Widget?

Because we can apply the same theme across the whole widget tree. Right?

As a result, we have kept the custom theme class in our Model folder.

```
import 'dart:math';

import 'package:flutter/material.dart';
import 'package:material_color_utilities/material_color_u\
tilities.dart';

class ThemeSettingChange extends Notification {
ThemeSettingChange({required this.settings});
final ThemeSettings settings;
}

class ThemeProvider extends InheritedWidget {
```

```
const ThemeProvider(
    {super.key,
    required this.settings,
    required this.lightDynamic,
    required this.darkDynamic,
    required super.child});

final ValueNotifier<ThemeSettings> settings;
final ColorScheme? lightDynamic;
final ColorScheme? darkDynamic;

Color custom(CustomColor custom) {
    if (custom.blend) {
    return blend(custom.color);
    } else {
    return custom.color;
    }
}

Color blend(Color targetColor) {
    return Color(
        Blend.harmonize(targetColor.value, settings.value\
.sourceColor.value));
}

Color source(Color? target) {
    Color source = settings.value.sourceColor;
    if (target != null) {
    source = blend(target);
    }
    return source;
}

ColorScheme colors(Brightness brightness, Color? targetCo\
lor) {
    final dynamicPrimary = brightness == Brightness.light
        ? lightDynamic?.primary
        : darkDynamic?.primary;
    return ColorScheme.fromSeed(
    seedColor: dynamicPrimary ?? source(targetColor),
    brightness: brightness,
    );
}

ShapeBorder get shapeMedium => RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(8),
    );

CardTheme cardTheme() {
    return CardTheme(
    elevation: 0,
    shape: shapeMedium,
    clipBehavior: Clip.antiAlias,
    );
}

ListTileThemeData listTileTheme(ColorScheme colors) {
    return ListTileThemeData(
    shape: shapeMedium,
    selectedColor: colors.secondary,
    );
}

AppBarTheme appBarTheme(ColorScheme colors) {
    return AppBarTheme(
    elevation: 0,
    backgroundColor: colors.surface,
    foregroundColor: colors.onSurface,
    );
}
```

```

TabBarTheme tabBarTheme(ColorScheme colors) {
    return TabBarTheme(
    labelColor: colors.secondary,
    unselectedLabelColor: colors.onSurfaceVariant,
    indicator: BoxDecoration(
        border: Border(
        bottom: BorderSide(
            color: colors.secondary,
            width: 2,
        ),
        ),
    ),
    );
}

BottomAppBarTheme bottomAppBarTheme(ColorScheme colors) {
    return BottomAppBarTheme(
    color: colors.surface,
    elevation: 0,
    );
}

BottomNavigationBarThemeData bottomNavigationBarTheme(Col\
orScheme colors) {
    return BottomNavigationBarThemeData(
    type: BottomNavigationBarType.fixed,
    backgroundColor: colors.surfaceVariant,
    selectedItemColor: colors.onSurface,
    unselectedItemColor: colors.onSurfaceVariant,
    elevation: 0,
    landscapeLayout: BottomNavigationBarLandscapeLayout.c\
entered,
    );
}
```

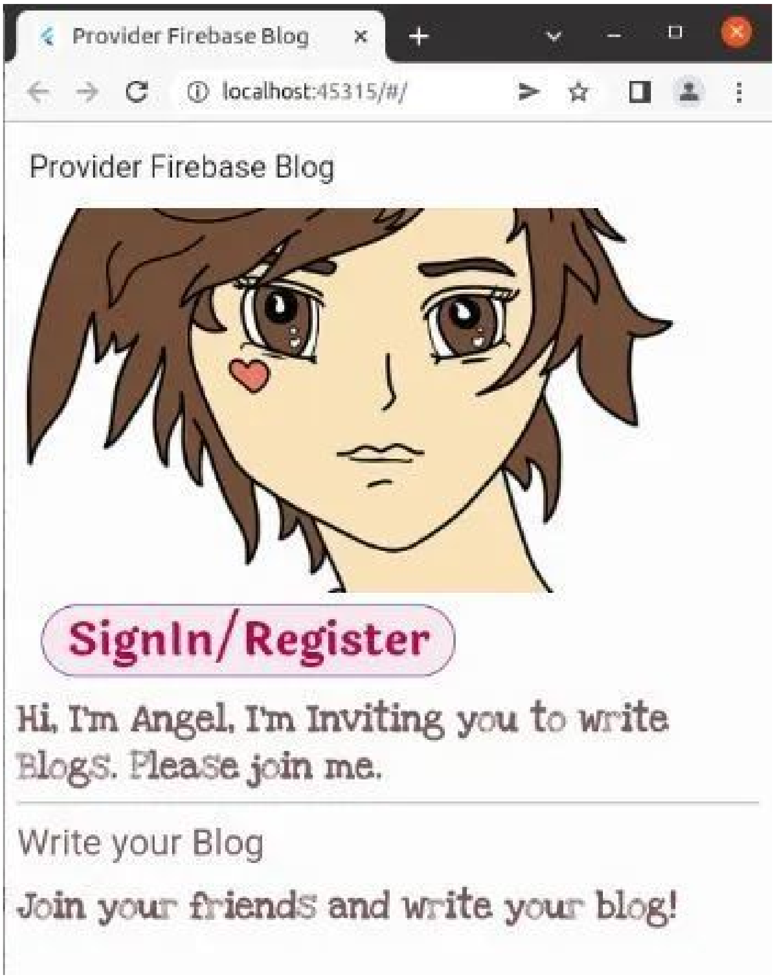
```

NavigationRailThemeData navigationRailTheme(ColorScheme c\
olors) {
    return const NavigationRailThemeData();
}

DrawerThemeData drawerTheme(ColorScheme colors) {
    return DrawerThemeData(
    backgroundColor: colors.surface,
    );
}

ThemeData light([Color? targetColor]) {
    final colorScheme = colors(Brightness.light, targetCo\
lor);
    return ThemeData.light().copyWith(
    //pageTransitionsTheme: pageTransitionsTheme,
    colorScheme: colorScheme,
    appBarTheme: appBarTheme(colorScheme),
    cardTheme: cardTheme(),
    listTileTheme: listTileTheme(colorScheme),
    bottomAppBarTheme: bottomAppBarTheme(colorScheme),
    bottomNavigationBarTheme: bottomNavigationBarTheme(co\
lorScheme),
    navigationRailTheme: navigationRailTheme(colorScheme),
    tabBarTheme: tabBarTheme(colorScheme),
    drawerTheme: drawerTheme(colorScheme),
    scaffoldBackgroundColor: colorScheme.background,
    useMaterial3: true,
    );
}

ThemeData dark([Color? targetColor]) {
    final colorScheme = colors(Brightness.dark, targetCol\
or);
```

```
    return ThemeData.dark().copyWith(
    colorScheme: colorScheme,
    appBarTheme: appBarTheme(colorScheme),
    cardTheme: cardTheme(),
    listTileTheme: listTileTheme(colorScheme),
    bottomAppBarTheme: bottomAppBarTheme(colorScheme),
    bottomNavigationBarTheme: bottomNavigationBarTheme(co\
lorScheme),
    navigationRailTheme: navigationRailTheme(colorScheme),
    tabBarTheme: tabBarTheme(colorScheme),
    drawerTheme: drawerTheme(colorScheme),
    scaffoldBackgroundColor: colorScheme.background,
    useMaterial3: true,
    );
}

ThemeMode themeMode() {
    return settings.value.themeMode;
}

ThemeData theme(BuildContext context, [Color? targetColor\
]) {
    final brightness = MediaQuery.of(context).platformBri\
ghtness;
    return brightness == Brightness.light
        ? light(targetColor)
        : dark(targetColor);
}

static ThemeProvider of(BuildContext context) {
    return context.dependOnInheritedWidgetOfExactType<The\
meProvider>()!;
}

@override
```

```
bool updateShouldNotify(covariant ThemeProvider oldWidget\
) {
    return oldWidget.settings != settings;
}
}

class ThemeSettings {
ThemeSettings({
    required this.sourceColor,
    required this.themeMode,
});

final Color sourceColor;
final ThemeMode themeMode;
}

Color randomColor() {
return Color(Random().nextInt(0xFFFFFFFF));
}

// Custom Colors
const linkColor = CustomColor(
name: 'Link Color',
color: Color(0xFF00B0FF),
);

class CustomColor {
const CustomColor({
    required this.name,
    required this.color,
    this.blend = true,
});

final String name;
final Color color;
final bool blend;

Color value(ThemeProvider provider) {
    return provider.custom(this);
}
}
```

As we have set two settings – light and dark, we can use any one of them.

In the beginning, we have seen how we can use the light theme.



Hi, I'm Angel, I'm Inviting you to write Blogs. Please join me.

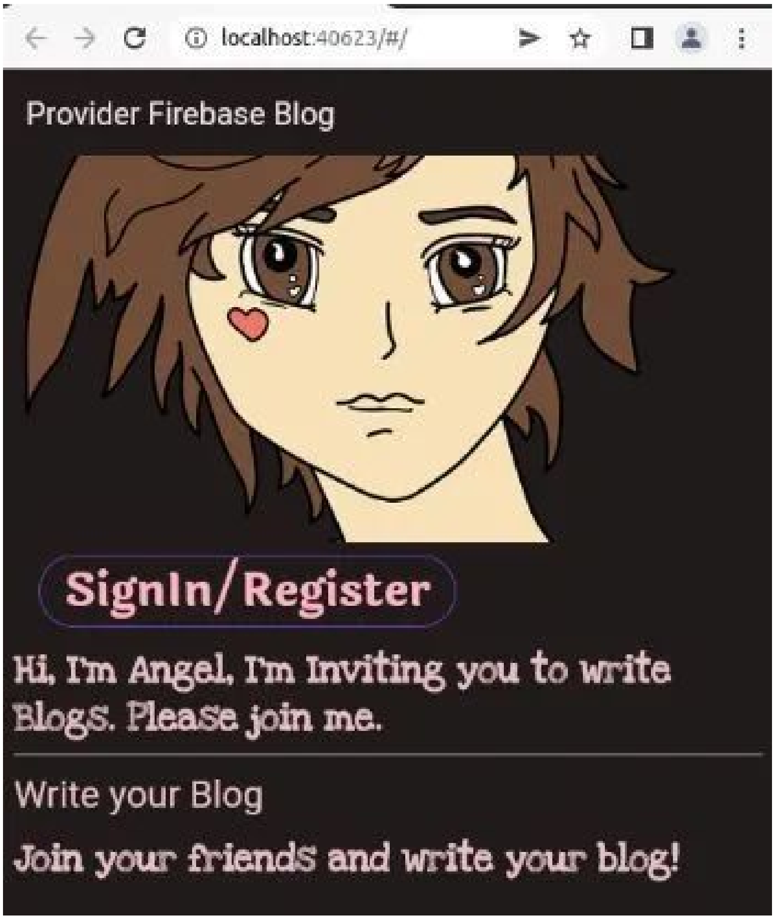
Write your Blog

Join your friends and write your blog!

Figure 12.21 – Flutter 3.0 Material 3 Theme

But we can easily change the theme to the dark mode in our MaterialApp.

```
import 'package:flutter/material.dart';
//import 'package:dynamic_color/dynamic_color.dart';

//import 'package:google_fonts/google_fonts.dart';

import '../main.dart';
import 'chat_home_page.dart';
import '../model/theme.dart';

class ChatApp extends StatelessWidget {
ChatApp({Key? key}) : super(key: key);

@override
Widget build(BuildContext context) {
    final theme = ThemeProvider.of(context);
    return MaterialApp(
    title: 'Provider Firebase Blog',
    debugShowCheckedModeBanner: false,
    theme: theme.dark(settings.value.sourceColor),
    home: const ChatHomePage(),
    );
}
}
```



Hi, I'm Angel, I'm Inviting you to write Blogs. Please join me.

Join your friends and write your blog!

Figure 12.22 – Flutter Material 3 Web App

Material 3 and Flutter 3.0

Flutter supports Material design from the very beginning. However, Material 3 is the next level design. As a result, Flutter also launches the 3.0 version which is much faster than the previous versions.

Most importantly, it adopts Material Design 3.

Therefore we have successfully adopted Material 3. Now, we would like to give it a final touch.

Previously we have seen that once the user posts a blog, it shows on the same page.

We've changed that part.

For example, we have changed the code a little bit. We're no longer showing the blog titles on the same page.

On the contrary, we have used a Text Button and navigated to another page.

```
import 'dart:async';

import 'package:blog_web_app_with_firebase/view/all_blogs\
.dart';
import 'package:flutter/material.dart';
import 'package:google_fonts/google_fonts.dart';
import 'package:provider/provider.dart';

import '../controller/all_widgets.dart';
import '../controller/authenticate_to_firebase.dart';
import '../model/state_of_application.dart';

class LetUsChatMessage {
LetUsChatMessage({
    required this.name,
    required this.title,
    required this.body,
});
final String name;
final String title;
final String body;
}
```

```

class LetUsChat extends StatefulWidget {
const LetUsChat({
    required this.addMessageOne,
    required this.messages,
});
final FutureOr<void> Function(String messageOne, String m\
essageTwo)
    addMessageOne;
final List<LetUsChatMessage> messages;

@override
State<LetUsChat> createState() => _LetUsChatState();
}

class _LetUsChatState extends State<LetUsChat> {
final _formKey = GlobalKey<FormState>(debugLabel: '_LetUs\
Blog');
final _controllerOne = TextEditingController();
final _controllerTwo = TextEditingController();

@override
Widget build(BuildContext context) {
    return Scaffold(
    appBar: AppBar(
        title: Text(
        'Provider Firebase Blog',
        style: TextStyle(
            color: Theme.of(context).appBarTheme.foregrou\
ndColor,
        ),
        ),
    ),
    body: Padding(
        padding: const EdgeInsets.all(8.0),
        child: Form(
        key: _formKey,
        child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
            TextFormField(
                controller: _controllerOne,
                decoration: InputDecoration(
                hintText: 'Title',
                enabledBorder: OutlineInputBorder(
                    borderRadius: BorderRadius.circular(5\
),
                    borderSide: BorderSide(
                    color: Theme.of(context).highlightCol\
or,
                    width: 1.0,
                    ),
                ),
                focusedBorder: OutlineInputBorder(
                    borderRadius: BorderRadius.circular(3\
0),
                    borderSide: const BorderSide(
                    color: Colors.purple,
                    width: 2.0,
                    ),
                ),
                ),
                validator: (value) {
                if (value == null || value.isEmpty) {
                    return 'Enter your message to continu\
e';
                }
                return null;
                },
            ),
            Expanded(
                child: SizedBox(
                height: 150.0,
                child: TextFormField(
                    controller: _controllerTwo,
                    maxLines: 10,
                    decoration: InputDecoration(
                    hintText: 'Body',
                    enabledBorder: OutlineInputBorder(
                        borderRadius: BorderRadius.circul\
ar(5),
                        borderSide: BorderSide(
                        color: Theme.of(context).highligh\
tColor,
                        width: 1.0,
                        ),
                    ),
                    focusedBorder: OutlineInputBorder(
                        borderRadius: BorderRadius.circul\
ar(30),
                        borderSide: const BorderSide(
                        color: Colors.purple,
                        width: 2.0,
                        ),
                    ),
                    ),
                    validator: (value) {
                    if (value == null || value.isEmpty) {
                        return 'Enter your message to con\
tinue';
                    }
                    return null;
                    },
                ),
                ),
            ),
            const SizedBox(width: 10.0),
            StyledButton(
                onPressed: () async {
                if (_formKey.currentState!.validate()) {
                    await widget.addMessageOne(
                        _controllerOne.text, _controllerT\
wo.text);
                    _controllerOne.clear();
                    _controllerTwo.clear();
                }
                },
                child: Row(
                children: const [
                    Icon(Icons.send),
                    SizedBox(width: 6),
                    Text('SUBMIT'),
                ],
                ),
            ),
            const SizedBox(
                height: 20.0,
            ),
            TextButton(
                onPressed: () {
                    Navigator.push(
                    context,
                    MaterialPageRoute(
                        builder: (context) =>
                            AllBlogs(messages: widget.mes\
sages),
                    ),
                    );
                },
                child: Text(
                    'All Titles',
                    style: GoogleFonts.aBeeZee(
                    fontSize: 60.0,
                    fontWeight: FontWeight.bold,
                    ),
                ))
            ],
        ),
        ),
    ),
    );
}
} // LetUsChat state ends
```

As an outcome, we press the button, and reach a new page where we can see all the blog titles.

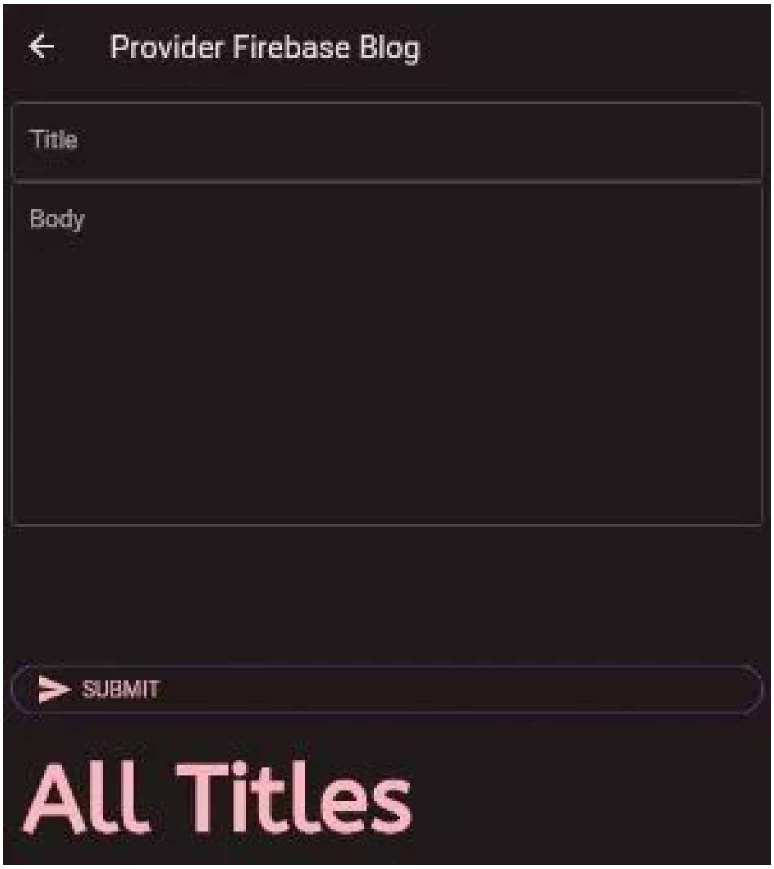


Figure 12.23 – Material 3 and Flutter Firebase Provider Blog

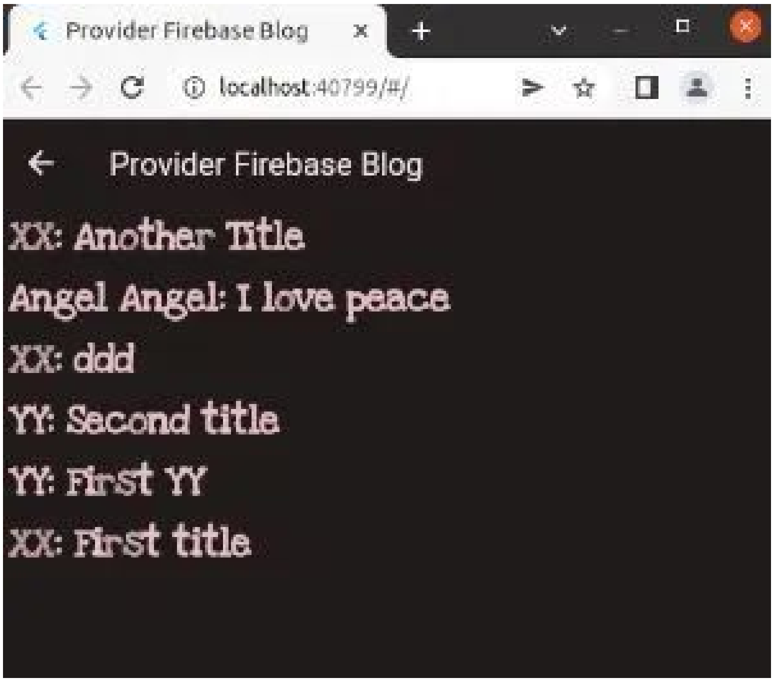
Showing all titles is not difficult. We have passed the blog collection which we retrieve from the Firestore database.

In the above code watch this part in particular.

```
final List<LetUsChatMessage> messages;
...
TextButton(
                onPressed: () {
                    Navigator.push(
                    context,
                    MaterialPageRoute(
                        builder: (context) =>
                            AllBlogs(messages: widget.mes\
sages),
                    ),
                    );
                },
                child: Text(
                    'All Titles',
                    style: GoogleFonts.aBeeZee(
                    fontSize: 60.0,
                    fontWeight: FontWeight.bold,
                    ),
                ))
```

Meanwhile we can click any title and read the entire blog.

But before that we need to show them. Right?



XX: Another Title
Angel Angel: I love peace
XX: ddd
YY: Second title
YY: First YY
XX: First title

Figure 12.24 – Material 3 and Flutter Firebase Provider Blog all titles

Let's take a look at the full code. That'll give you an idea of how we can use the for loop to select each title, and after that we have used a Gesture Detector Widget so that we can navigate and reach individual posts.

```
import 'dart:async';

import 'package:flutter/material.dart';
import 'package:google_fonts/google_fonts.dart';
import 'package:provider/provider.dart';

import '../controller/all_widgets.dart';
import '../controller/authenticate_to_firebase.dart';
import '../model/state_of_application.dart';

```

```
class AllBlogs extends StatefulWidget {
const AllBlogs({
    required this.messages,
});

final messages;

@override
State<AllBlogs> createState() => _AllBlogsState();
}

class _AllBlogsState extends State<AllBlogs> {
@override
Widget build(BuildContext context) {
    return Scaffold(
        appBar: AppBar(
        title: Text(
            'Provider Firebase Blog',
            style: TextStyle(
            color: Theme.of(context).appBarTheme.foregrou\
ndColor,
            ),
        ),
        ),
        body: ListView(
        children: [
            for (var message in widget.messages)
            GestureDetector(
                onTap: () {
                Navigator.push(
                    context,
                    MaterialPageRoute(
                    builder: (context) => BlogDetailScree\
n(
                        name: message.name,
                        title: message.title,
                        body: message.body,
                    ),
                    ),
                );
                },
                child: Paragraph('${message.name}: ${mess\
age.title}'),
            ),
        ],
        ));
}
} // AllBlogs state ends

class BlogDetailScreen extends StatelessWidget {
// static const routename = '/product-detail';

const BlogDetailScreen({
    Key? key,
    required this.name,
    required this.title,
    required this.body,
}) : super(key: key);
final String name;
final String title;
final String body;

@override
Widget build(BuildContext context) {
    return Scaffold(
    appBar: AppBar(
        title: Text(name),
    ),
    body: SingleChildScrollView(
        child: Consumer<StateOfApplication>(
        builder: (context, appState, _) => Column(
            children: <Widget>[
            if (appState.loginState == UserStatus.loggedI\
n) ...[
                SizedBox(
                height: 300,
                width: double.infinity,
                child: Image.network(
                    'https://cdn.pixabay.com/photo/2018/0\
3/24/00/36/girl-3255402_960_720.png',
                    width: 250,
                    height: 250,
                    fit: BoxFit.cover,
                ),
                ),
                const SizedBox(height: 10),
                Text(
                title,
                style: GoogleFonts.aBeeZee(
                    fontSize: 60.0,
                    fontWeight: FontWeight.bold,
                ),
                ),
                const SizedBox(
                height: 10,
                ),
                Container(
                padding: const EdgeInsets.symmetric(horiz\
ontal: 10),
                width: double.infinity,
                child: Text(
                    body,
                    textAlign: TextAlign.center,
                    softWrap: true,
                    style: GoogleFonts.aBeeZee(
                    fontSize: 30.0,
                    fontWeight: FontWeight.bold,
                    ),
                ),
                ),
                const SizedBox(
                height: 10,
                ),
            ],
            ],
        ),
        ),
    ),
    );
}
}
```

Most importantly, each post shows the user's name on the AppBar Widget.

And in the body section we see the title and content.

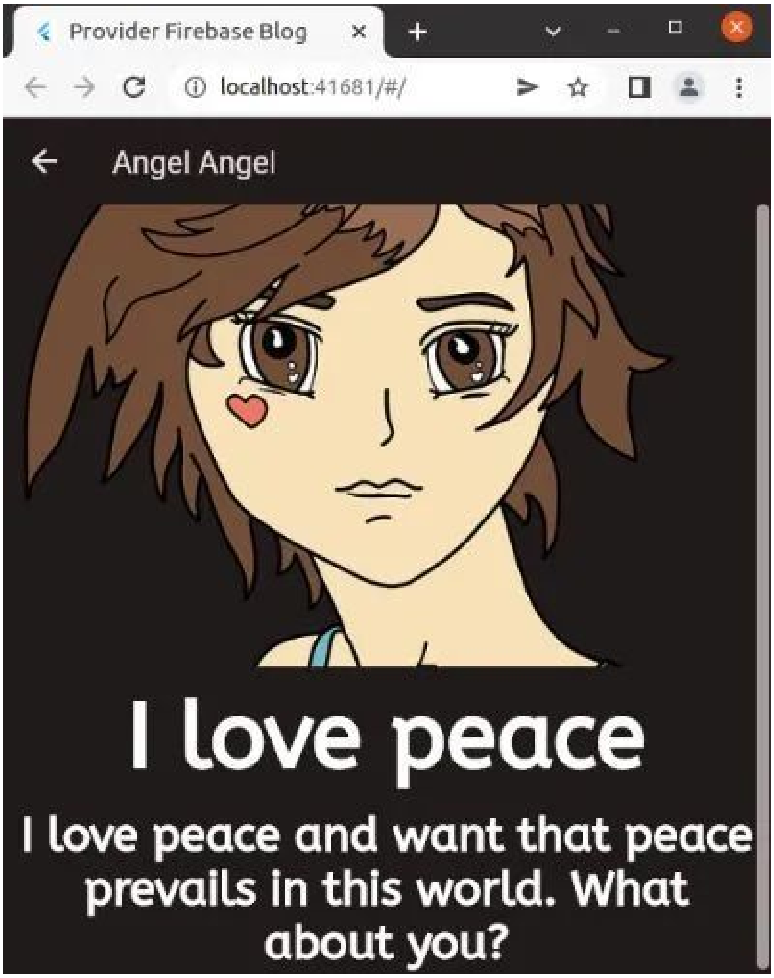

Provider Firebase Blog
localhost:41681/#/
Angel Angel
I love peace
I love peace and want that peace prevails in this world. What about you?

Figure 12.25 – Material 3 and Flutter Firebase Provider Blog individual post page

Do you want to clone the entire project and want to modify the code?

Please clone this GitHub Repository.

For more Flutter related Articles and Resources[36]

The code repositories for this book[37]

[36]https://sanjibsinha.com/blog_web_app_with_firebase
[37]https://github.com/sanjibsinha/

13. A Complete News App - Using WordPress as the backend

Can we integrate a WordPress app to our Flutter App? Certainly we can. It's a challenge that we are going to discuss in this section.

For example we can use WordPress as the backend to to Flutter App.

As a result it cuts short our tasks.

The question is how we can shorten this task?

How we will shorten this task? Before we answer, let's see how WordPress and Flutter work together.

How Flutter and WordPress work together?

Flutter is a cross-platform UI toolkit. On the other hand, WordPress manages the backend with MySQL database.

To make them work together we need HTTP plugin that connects Flutter and WordPress.

So, we can manage the front with Flutter and manage the backend database with WordPress.

Consequently, they work together.

Meanwhile, to manage the backend with Flutter is not easy. WordPress makes it easy.

How does the WordPress make it easy?

Let’s see the image first. An image speaks thousand words.

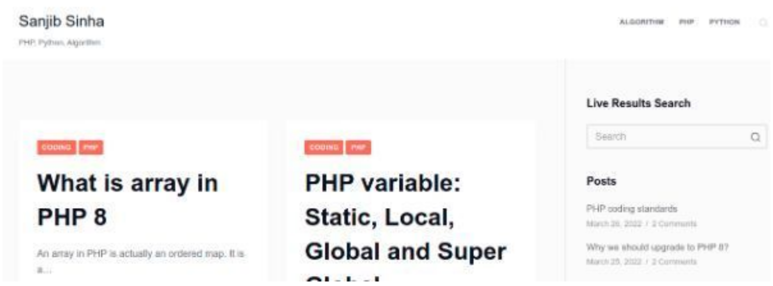
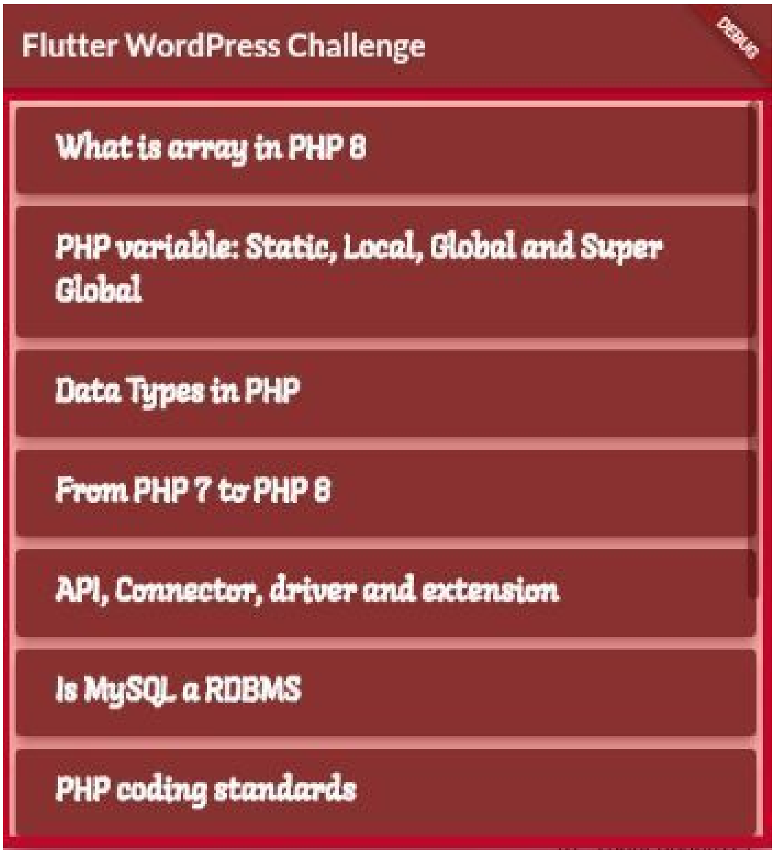

Sanjib Sinha
PHP, Python, Algorithm
ALGORITHM
PHP
PYTHON
Live Results Search
Search
CODING
PHP
What is array in PHP 8
An array in PHP is actually an ordered map. It is a...
CODING
PHP
PHP variable: Static, Local, Global and Super
Posts
PHP coding standards
March 28, 2022 / 2 Comments
Why we should upgrade to PHP 8?
March 25, 2022 / 2 Comments

Figure 13.1 – Flutter WordPress challenge first example

For example, the above image shows the WordPress Web App where I write on PHP,Python and Algorithm.

How we can display the same post on our Flutter App?

Let’s see the second image where we have displayed the same titles on the Flutter App.



What is array in PHP 8

PHP variable: Static, Local, Global and Super Global

Data Types in PHP

From PHP 7 to PHP 8

API, Connector, driver and extension

Is MySQL a RDBMS

PHP coding standards

Figure 13.2 – Flutter WordPress challenge second example

We have made our Flutter App work together with the WordPress App so readers can read the same posts.

To read the post, they will just tap the title and read the full article.

How do I link my WordPress to Flutter?

If we want to link the WordPress to Flutter, we need the HTTP Package. In addition, we will add the Google fonts package to add some styling.

Now we are ready to go.

Firstly, to use that package we add the dependency to the "pubspec.yaml" file.

```
dependencies:
flutter:
    sdk: flutter

cupertino_icons: ^1.0.2
http: ^0.13.4
google_fonts: ^2.3.1
```

Secondly, we will use the HTTP package in our model folder where we're sending the request to the WordPress app.

In that way, we can link the Flutter mobile App to the WordPress web App.

```
import 'dart:convert';
import 'package:http/http.dart' as http;

class Post {
String baseURL =
    'http://algorithm.sanjibsinha.com/wp-json/wp/v2/posts\
?_embed';

Future<List> getAllPosts() async {
    try {
    var response = await http.get(Uri.parse(baseURL));
    if (response.statusCode == 200) {
        return jsonDecode(response.body);
    } else {
        return Future.error('Server Error');
    }
    } catch (e) {
    throw '$e';
    }
}
}
```

Most importantly, we will make the Type of the method Future whose Type will be a List.

Why?

Because we want to get the posts in a List so that we can iterate over them and display on the screen. Right?

Moreover, we get the posts from the base URL.

```
String baseURL =
    'http://algorithm.sanjibsinha.com/wp-json/wp/v2/posts\
?_embed';
```

Meanwhile, we get the response with the help of the HTTP package.

The response is in JSON Map format. So we need to decode the data.

Once we have decoded the data, we can use them in our page.

Can we convert the WordPress app into the Flutter app?

That is the real challenge. Because we are going to convert the existing WordPress web app into a Flutter App.

To convert we need the Future Builder Widget that will use the List Type.

So the future property of the Future Builder Widget will use the Future method we have already defined in our model folder.

After that, the builder property of the Future Builder widget will return a ListView builder constructor which will display the post as the List items.

Let’s see the code.

```
import 'package:flutter/material.dart';
import 'package:flutter_wordpress_challenge/model/happy_t\
heme.dart';

import '../model/post.dart';

class FlutterWordPressHomePage extends StatefulWidget {
const FlutterWordPressHomePage({Key? key, required this.t\
itle})
    : super(key: key);

final String title;

@override
State<FlutterWordPressHomePage> createState() =>
    _FlutterWordPressHomePageState();
}

```

```
class _FlutterWordPressHomePageState extends State<Flutte\
rWordPressHomePage> {
@override
void initState() {
    super.initState();
    Post().getAllPosts();
}

@override
Widget build(BuildContext context) {
    Post posts = Post();
    return Scaffold(
    backgroundColor: HappyTheme.shrineErrorRed,
    appBar: AppBar(
        backgroundColor: HappyTheme.shrineBrown600,
        title: Text(
        widget.title,
        style: HappyTheme.appbarStyle,
        ),
    ),
    body: Center(
        child: Container(
        color: HappyTheme.shrinePink300,
        margin: const EdgeInsets.all(8.0),
        child: FutureBuilder<List>(
            future: posts.getAllPosts(),
            builder: (context, snapshot) {
            if (snapshot.hasData) {
                if (snapshot.data!.isEmpty) {
                return const Center(
                    child: Text('No Post available.'),
                );
                }
                return ListView.builder(
                itemCount: snapshot.data?.length,
                itemBuilder: (context, index) {
                    return Card(
                    elevation: 10.0,
                    shadowColor: HappyTheme.shrineErrorRe\
d,
                    color: HappyTheme.inactiveCoor,
                    child: ListTile(
                        title: Row(
                        children: [
                            /* Expanded(
                            child: Image.network(snapshot\
.data![index]
                                    ['_embeded']['wp:feat\
uredmedia'][0]
                                ['source_url']),
                            ), */
                            Expanded(
                            child: Container(
                                padding: const EdgeInsets\
.all(10.0),
                                child: Text(
                                snapshot.data![index]['ti\
tle']['rendered'],
                                style: HappyTheme.answerS\
tyle,
                                ),
                            ),
                            )
                        ],
                        ),
                    ),
                    );
                },
                );
            } else if (snapshot.hasError) {
                return Center(
                child: Text(snapshot.error.toString()),
                );
            } else {
                return const Center(
                child: CircularProgressIndicator(),
                );
            }
            },
        ),
        ),
    ),
    // This trailing comma makes auto-formatting nicer fo\
r build methods.
    );
}
}
```

We have commented out the image part because the WordPress web App has no featured image.

However, we have fetched the post and display them on the Flutter App.

If you want to clone this step, you can use this GitHub repository.

- For full code snippet please visit the respective GitHub Repository - [38]

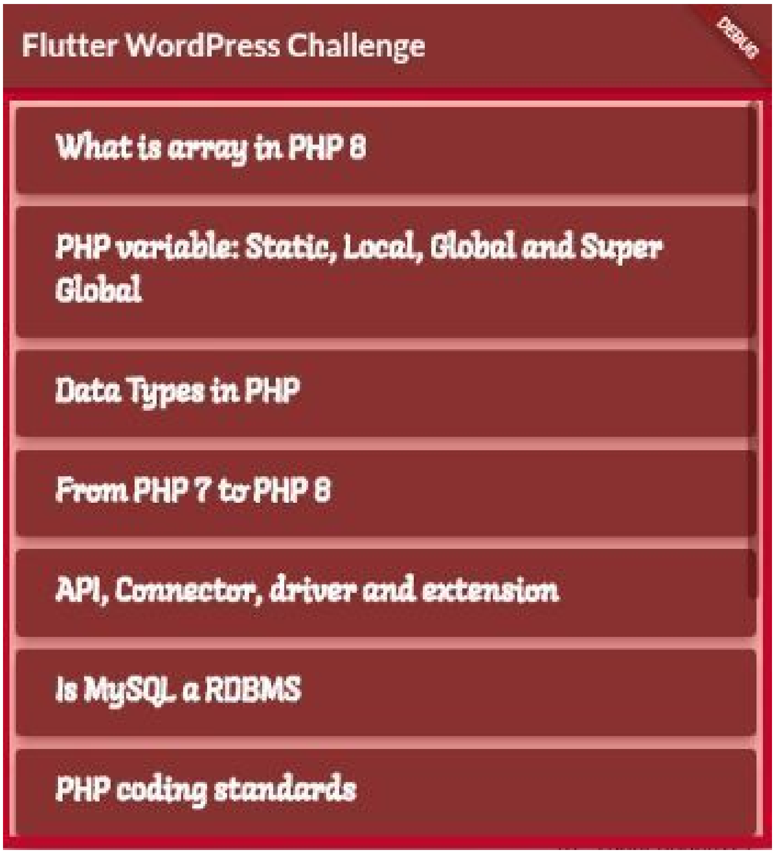
Create, retrieve, update, and delete

In the previous section, we have seen how we can use WordPress as the backend of the Flutter app. Flutter and WordPress can work together.

[38]https://github.com/sanjibsinha/flutter_wordpress_challenge/tree/first-step

We can create, retrieve, update, and delete our posts in the WordPress web app. And, as an effect, that will reflect on the Flutter App.

For example, we had created a minimalist App which looks as follows.


Flutter WordPress Challenge

DEBUG

What is array in PHP 8

PHP variable: Static, Local, Global and Super Global

Data Types in PHP

From PHP 7 to PHP 8

API, Connector, driver and extension

Is MySQL a RDBMS

PHP coding standards

Figure 13.3 – Flutter WordPress challenge second example

From the WordPress database, it retrieves the titles and display them on our Flutter App.

However, we cannot just display the title only. On the contrary, we should display the posts, images also.

How to create a NewsApp in WordPress?

All we need a basic understanding of WordPress. That will help us to install a ready made theme that comes with demo content.

Therefore, it does not take much time to create a NewsApp in WordPress.

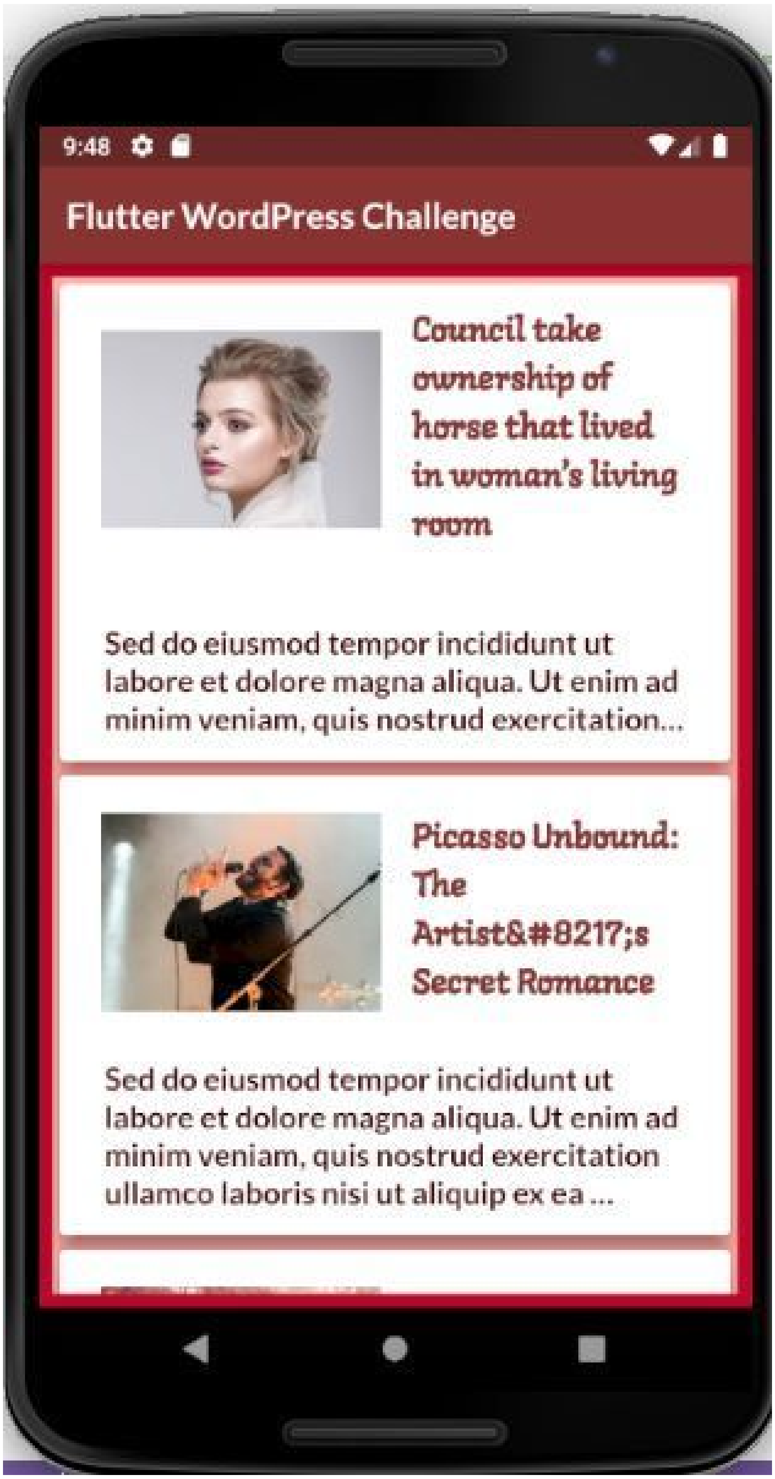
But to use that NewsApp as a backend for Flutter, we need a package first.

In our earlier sections, we have discussed how to use HTTP package. We will use the same package here.

As a result, in our model folder, we have a Post class.

```
import 'dart:convert';
import 'package:http/http.dart' as http;

/// fourth test to test latest post by category ID

class Post {
String baseURLForAllPosts =
    'http://news.sanjibsinha.com/wp-json/wp/v2/posts?_emb\
ed';

String latestPostsByCategoryID =
    'http://news.sanjibsinha.com/wp-json/wp/v2/latest-pos\
ts/';

Future<List> getAllPosts() async {
    try {
    var response = await http.get(Uri.parse(baseURLForAll\
Posts));
    if (response.statusCode == 200) {
        return jsonDecode(response.body);
    } else {
        return Future.error('Server Error');
    }
    } catch (e) {
    throw '$e';
    }
}

Future<List> getPostsByCategoryID(int id) async {
    String latestPosts = '$latestPostsByCategoryID/$id';
    try {
    var response = await http.get(Uri.parse(latestPosts));
    if (response.statusCode == 200) {
        return jsonDecode(response.body);
    } else {
        return Future.error('Server Error');
    }
    } catch (e) {
    throw '$e';
    }
}
}
```

Firstly, we will display all the posts on our home page of the Flutter App.


9:48

Flutter WordPress Challenge

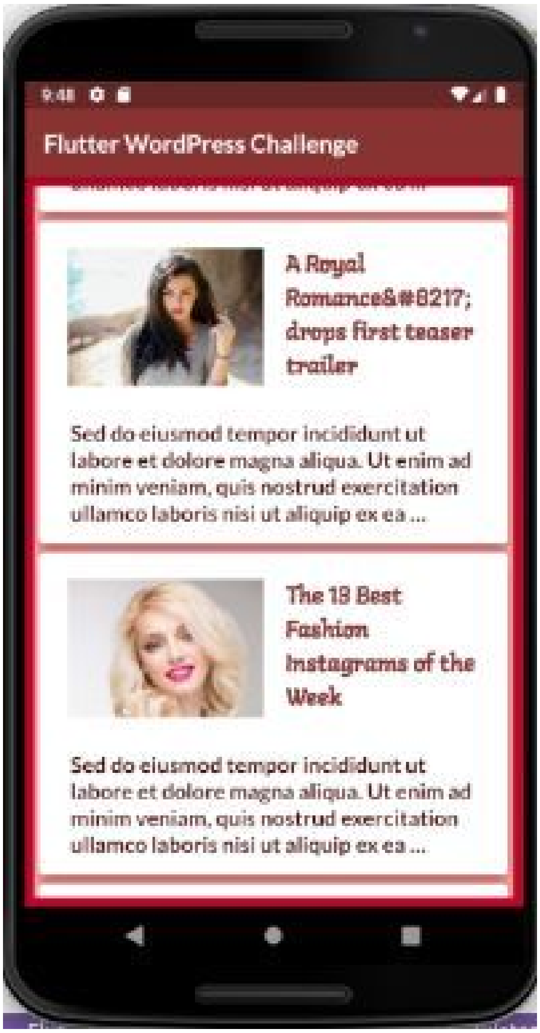
Council take ownership of horse that lived in woman's living room

Sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation...

Picasso Unbound: The Artist&#8217;s Secret Romance

Sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea ...

Our home page displays all the posts. We can scroll down so we can view other posts also.



A Royal Romance&#8217; drops first teaser trailer

Sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea ...

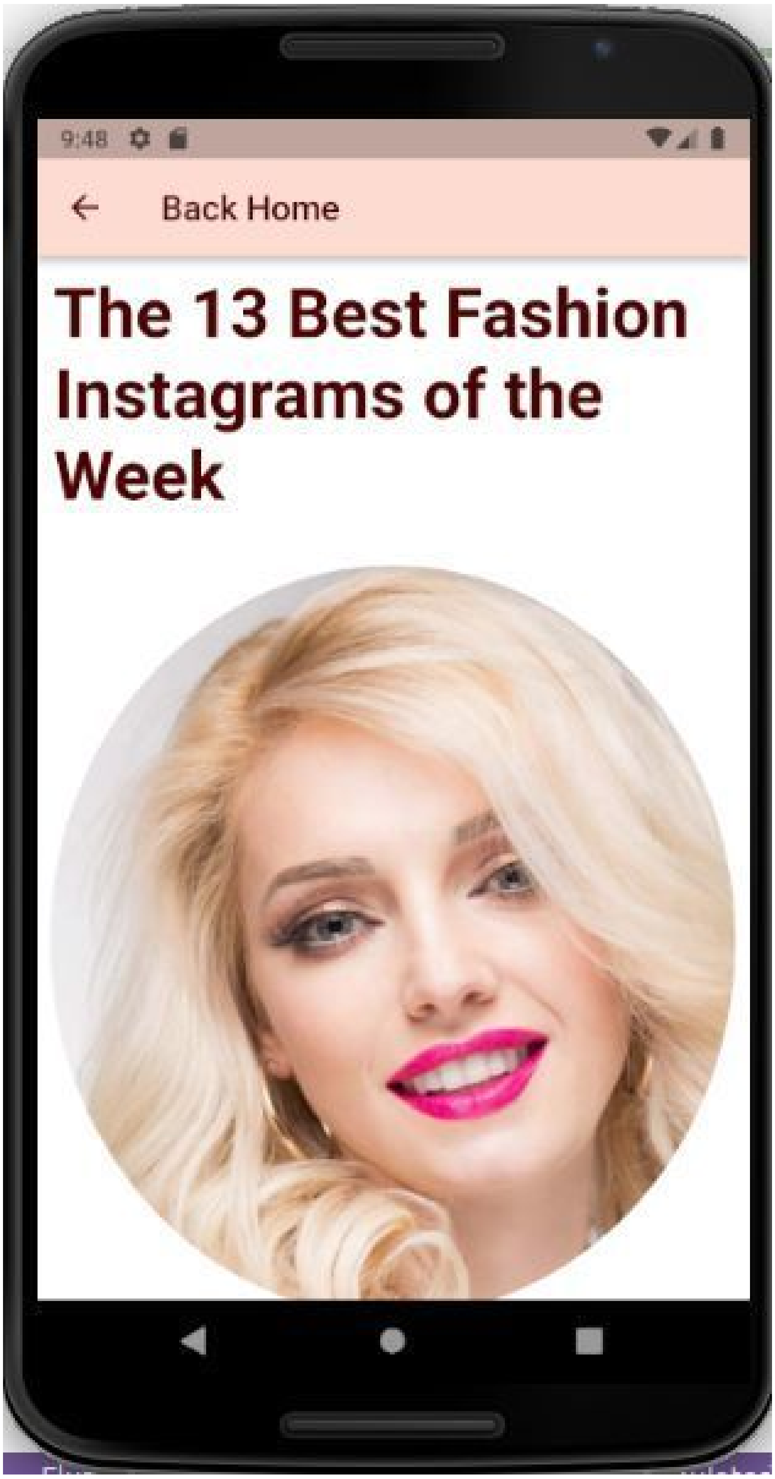
The 19 Best Fashion Instagrams of the Week

Sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea ...

Figure 13.5 – Flutter backend WordPress second

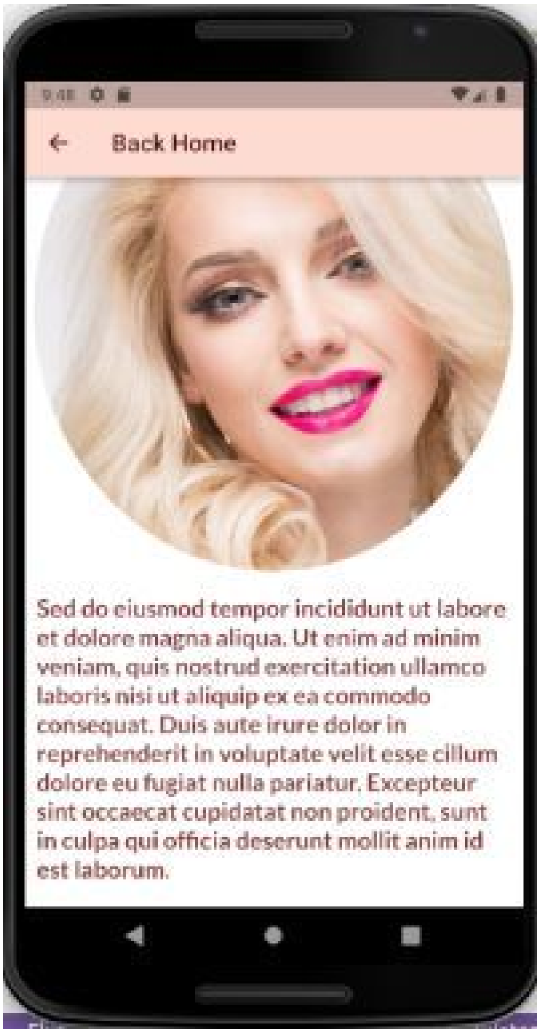
Each post comes with the title, featured image and the post. Although the post shows a few lines only.

Meanwhile, to read the full post, we can click any post on the home page. That will take us to the destination page.



The 13 Best Fashion Instagrams of the Week

For instance, just like the home page, we can scroll down the single post to read the full post.



Sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Figure 13.7 – Flutter backend WordPress fourth

In addition, we can press the back button to get back to the home page again.

As we add the new post, that will reflect on our Flutter home page. How does it happen? Let's see.

How Can I use Flutter with WordPress?

Firstly, we have added the dependency in our "pubspec.yaml" file. Remember, we need the HTTP package to send the request and receive the response.

After that, based on that response, we retrieve the content. In other words, the HTTP package plays the most important role.

```
dependencies:
flutter:
    sdk: flutter

cupertino_icons: ^1.0.2
http: ^0.13.4
google_fonts: ^2.3.1
```

On the other hand, the Google fonts package helps us to enhance the styling. For example we can use any fancy font, if we wish.

But, we should always use the fonts that are easy to read.

Secondly, in the home page we will use the Future Builder widget that has a property future which get all posts.

Besides, the builder property will return the ListView builder constructor that will display all the posts with title, image and content.

```
import 'package:flutter/material.dart';
import 'package:flutter_wordpress_challenge/model/happy_t\
heme.dart';

import '../model/post.dart';
import 'post_detail.dart';

/// added content
/// adding the navigation

class FlutterWordPressHomePage extends StatefulWidget {
const FlutterWordPressHomePage({Key? key, required this.t\
itle})
    : super(key: key);

final String title;

@override
State<FlutterWordPressHomePage> createState() =>
    _FlutterWordPressHomePageState();
}

class _FlutterWordPressHomePageState extends State<Flutte\
rWordPressHomePage> {
@override
void initState() {
    super.initState();
    Post().getAllPosts();
}

@override
Widget build(BuildContext context) {
    Post posts = Post();
    return Scaffold(
    backgroundColor: HappyTheme.shrineErrorRed,
    appBar: AppBar(
        backgroundColor: HappyTheme.shrineBrown600,
        title: Text(
        widget.title,
        style: HappyTheme.appbarStyle,
        ),
    ),
    body: Center(
        child: Container(
        color: HappyTheme.shrinePink300,
        margin: const EdgeInsets.all(8.0),
        child: FutureBuilder<List>(
            future: posts.getAllPosts(),
            builder: (context, snapshot) {
            if (snapshot.hasData) {
                if (snapshot.data!.isEmpty) {
                return const Center(
                    child: Text('No Post available.'),
                );
                }
                return ListView.builder(
                itemCount: snapshot.data?.length,
                itemBuilder: (context, index) {
                    return Card(
                    elevation: 10.0,
                    shadowColor: HappyTheme.shrineErrorRe\
d,
                    child: ListTile(
                        title: Row(
                        children: [
                            Expanded(
                            child: Container(
                                padding: const EdgeInsets\
.all(8.0),
                                width: 150,
                                height: 150,
                                child: Image.network(snap\
shot.data![index]
                                        ['_embedded']['wp\
:featuredmedia'][0]
                                    ['source_url']),
                            ),
                            ),
                            Expanded(
                            child: Container(
                                padding: const EdgeInsets\
.all(10.0),

                                child: Text(
                                snapshot.data![index]['ti\
tle']['rendered'],

                                style: HappyTheme.titleSt\
yle,
                                ),
                            ),
                            ),
                        ],
                        ),
                        subtitle: Container(
                        padding: const EdgeInsets.all(10.\
0),

                        child: Text(
                            snapshot.data![index]['conten\
t']['rendered']
                                .toString()
                                .replaceAll('<p>', '')
                                .replaceAll('</p>', '')
                                .replaceAll('<strong>', '\
')
                                .replaceAll('</strong>', \
''),
                            maxLines: 4,
                            overflow: TextOverflow.ellips\
is,
                            style: HappyTheme.contentStyl\
e,
                        ),
                        ),
                        onTap: () {
                        Navigator.push(
                            context,
                            MaterialPageRoute(
                            builder: (context) => PostDet\
ail(
                                data: snapshot.data![inde\
x],
                            ),
                            ),
                        );
                        },
                    ),
                    );
                },
                );
            } else if (snapshot.hasError) {
                return Center(
                child: Text(snapshot.error.toString()),
                );
            } else {
                return const Center(
                child: CircularProgressIndicator(),
                );
            }
            },
        ),
        ),
    ),
    // This trailing comma makes auto-formatting nicer fo\
r build methods.
    );
}
}
```

Finally, we should be careful about getting the data in the right manner.

Why?

Because the data comes in a JSON format. Therefore, we have to extract the key to get the value.

Sending data to the single post page We have discussed how to send data from one page to another page. In addition,we have the data as a single List of various maps and lists inside.

Above all, every post has the index associated with it.

```
import 'package:flutter/material.dart';
import 'package:flutter_wordpress_challenge/model/happy_t\
heme.dart';

class PostDetail extends StatelessWidget {
const PostDetail({
    Key? key,
    required this.data,
}) : super(key: key);
final Map<dynamic, dynamic> data;

@override
Widget build(BuildContext context) {
    return Scaffold(
    appBar: AppBar(
        title: const Text('Back Home'),
    ),
    body: ListView(
        children: [
        Container(
            margin: const EdgeInsets.all(10.0),
            child: Text(
            data['title']['rendered'],
            style: const TextStyle(
                fontSize: 40.0,
                fontWeight: FontWeight.bold,
            ),
            ),
        ),
        const SizedBox(
            height: 10.0,
        ),
        Container(
            padding: const EdgeInsets.all(8.0),
            width: 450,
            height: 450,
            child: ClipOval(
            child: Image.network(
                data['_embedded']['wp:featuredmedia'][0][\
'source_url'],
                fit: BoxFit.cover,
            ),
            ),
        ),
        Container(
            padding: const EdgeInsets.all(10.0),
            child: Text(
            data['content']['rendered']
                .toString()
                .replaceAll('<p>', '')
                .replaceAll('</p>', '')
```

```
                    .replaceAll('<strong>', '')
                    .replaceAll('</strong>', ''),
                style: HappyTheme.postContentStyle,
                ),
            ),
            ],
        ),
        );
}
}
```

Now we can display the single post as we have used the Navigator push method.

As a result, it pushes the data through the class constructor.

If you want to clone this step please visit the respective branch of the GitHub repository.

- For full code snippet please visit the respective GitHub Repository - [39]

How to show Categories in the NewsApp

In this section we will learn how to use Tab in Flutter to show the Categories. We have been building the NewsApp.

So far we have progressed a little bit. In the previous sections, we have shown how to use the WordPress as the backend.

We have also been able to display the latest posts on the home page. But this is not enough. For example, a post may belong to one or many categories.

[39]https://github.com/sanjibsinha/flutter_wordpress_challenge/tree/fourth-step

Therefore, in this section we will try to show each category on the upper part of our Flutter App. In addition, we can also allow users to click any category to see all posts that belong to that category.

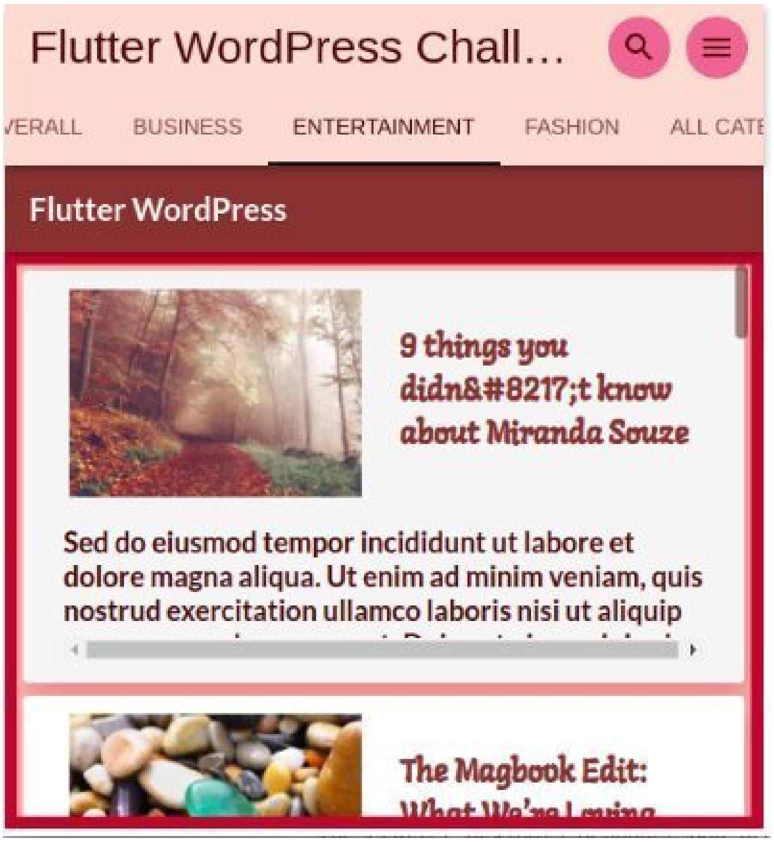
Finally, we will also retrieve every category on a single page.

Why?

Because users can click any category to view all the posts belong to that category.

There are lot of things to do.

To get an idea, we may take a look at the categories.



Sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip

Figure 13.8 – Flutter backend WordPress fourth

In the above image we have clicked the category entertainment. And, as a result, we have displayed all the posts belonging to that category.

The next image will show you how we can retrieve all the categories in one page.

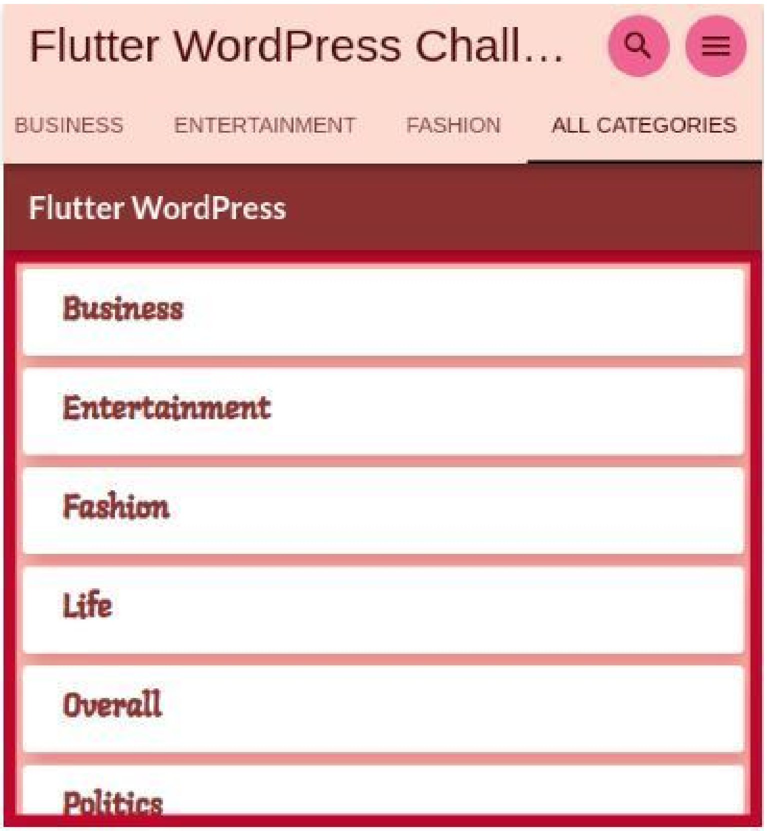


Figure 13.9 – Tab Flutter second example

How to use Tab in Flutter?

To make it happen we have used the Tab in Flutter. In any Flutter App, the tab layout is the part of the material library.

We can display all the categories in the Tab layout.

To do this we follow the following steps.

```
Firstly, Create a TabController.
Secondly, Create the tabs.
Finally, Create content for each tab.
```

After that, we have created a Tab Controller in the Dashboard controller. And in the Tab controller we have mentioned name of the each category.

Let us see the full code first. Meanwhile we can discuss the code later.

```
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';

import '../view/categories/posts_by_category_id_one.dart';

import 'dashboard_drawer.dart';
import '../view/latest_posts.dart';
import '../view/categories/posts_by_category_id_two.dart';
import '../view/categories/posts_by_category_id_three.dar\
t';
import '../view/categories/posts_by_category_id_four.dart\
';
import '../view/categories/test_page.dart';

class DashBoardHome extends StatefulWidget {
const DashBoardHome({
    Key? key,
}) : super(key: key);

@override
State<DashBoardHome> createState() => _DashBoardHomeState\
();
}

class _DashBoardHomeState extends State<DashBoardHome>
```

```
    with SingleTickerProviderStateMixin {
TabController? _tabController;

final List<Tab> topTabs = <Tab>[
    const Tab(child: Text('LATEST')),
    const Tab(child: Text('OVERALL')),
    const Tab(child: Text('BUSINESS')),
    const Tab(child: Text('ENTERTAINMENT')),
    const Tab(child: Text('FASHION')),
    const Tab(child: Text('ALL CATEGORIES')),
];

@override
void initState() {
    _tabController =
        TabController(length: topTabs.length, initialInde\
x: 0, vsync: this)
        ..addListener(() {
            setState(() {});
        });

    super.initState();
}

Future<bool> _onWillPop() async {
    if (_tabController?.index == 0) {
    await SystemNavigator.pop();
    }

    Future.delayed(const Duration(microseconds: 100), () {
    _tabController?.index = 0;
    });

    return _tabController?.index == 0;
}
```

```

final _scaffoldKey = GlobalKey<ScaffoldState>();

@override
Widget build(BuildContext context) {
    return Container(
    margin: const EdgeInsets.all(5.0),
    child: WillPopScope(
        onWillPop: _onWillPop,
        child: Scaffold(
        key: _scaffoldKey,
        appBar: AppBar(
            title: const Text(
            'Flutter WordPress Challenge',
            style: TextStyle(
                fontSize: 30,
            ),
            ),
            actions: [
            Container(
                child: IconButton(
                icon: const Icon(Icons.search),
                splashColor: Colors.transparent,
                highlightColor: Colors.transparent,
                onPressed: () {},
                ),
                decoration: BoxDecoration(
                shape: BoxShape.circle,
                color: Colors.pink[300],
                ),
            ),
            Container(
                margin: const EdgeInsets.symmetric(horizo\
ntal: 10.0),
                child: IconButton(
                icon: const Icon(Icons.menu),
                splashColor: Colors.transparent,
                highlightColor: Colors.transparent,
                onPressed: () => _scaffoldKey.currentStat\
e!.openEndDrawer(),
                ),
                decoration: BoxDecoration(
                shape: BoxShape.circle,
                color: Colors.pink[300],
                ),
            ),
            ],
            bottom: TabBar(
            controller: _tabController,
            indicatorColor: Colors.black,
            tabs: topTabs,
            isScrollable: true,
            ),
        ),
        endDrawer: Container(
            padding: const EdgeInsets.all(5.0),
            child: const DashBoardDrawer(),
        ),
        body: TabBarView(
            controller: _tabController,
            children: const [
            /// all categories displayed on tabs
            ///
            LatestPosts(),
            PostsByCategoryIDOne(),
            PostsByCategoryIDTwo(),
            PostsByCategoryIDThree(),
            PostsByCategoryIDFour(),
            TestPage(),
            ],
          ),
          ),
      ),
      );
}
}
```

How Tab Controller works in Flutter

First thing first.

We have created a Tab Controller and a List of Type Tab.

```
TabController? _tabController;

final List<Tab> topTabs = <Tab>[
    const Tab(child: Text('LATEST')),
    const Tab(child: Text('OVERALL')),
    const Tab(child: Text('BUSINESS')),
    const Tab(child: Text('ENTERTAINMENT')),
    const Tab(child: Text('FASHION')),
    const Tab(child: Text('ALL CATEGORIES')),
];
```

Next, we have initiated the Tab Controller once the page loads.

```
@override
void initState() {
    _tabController =
        TabController(length: topTabs.length, initialInde\
x: 0, vsync: this)
        ..addListener(() {
          setState(() {});
        });

    super.initState();
}
```

After that, we ensure that the back home button should take us to the home page.

```
Future<bool> _onWillPop() async {
    if (_tabController?.index == 0) {
    await SystemNavigator.pop();
    }

    Future.delayed(const Duration(microseconds: 100), () {
    _tabController?.index = 0;
    });

    return _tabController?.index == 0;
}
```

In the bottom property we have defined few other properties.

Firstly, the Tab should be scrollable. Secondly, whenever users click, it should have a indicator shadow color below.

Finally, we have made it sure that the value of the controller indicates to the Tab Controller.

```
bottom: TabBar(
            controller: _tabController,
            indicatorColor: Colors.black,
            tabs: topTabs,
            isScrollable: true,
            ),
```

At the body property we have used a TabBarView Widget whose children property returns a List of pages.

```
body: TabBarView(
            controller: _tabController,
            children: const [
            /// all categories displayed on tabs
            ///
            LatestPosts(),
            PostsByCategoryIDOne(),
            PostsByCategoryIDTwo(),
            PostsByCategoryIDThree(),
            PostsByCategoryIDFour(),
            TestPage(),
            ],
        ),
```

In the next section we will discuss how we can get all the pages using the Drawer Widget.

If you want to see the related code snippets please read this branch of GitHub repository.

- For full code snippet please visit the respective GitHub Repository - [40]
- Read updated articles on Flutter, Dart, and Algorithm - [41]

[40]https://github.com/sanjibsinha/flutter_wordpress_challenge/tree/sixth-step
[41]https://sanjibsinha.com

14. Flutter app and Artificial Intelligence

Can we use artificial intelligence in the Flutter app? Yes, we can. We can make Artificial Intelligence Apps using Flutter.

Not only that, we can use Flutter as a part of a Machine Learning tool also.

How can we do that?

We can do that with the TensorFlow Lite package.

What is TensorFlow?

Firstly, we know that a scalar means a single number. Secondly, we know that a vector means one dimensional array. Thirdly, a matrix represents a two dimensional array.

In Flutter we use List and Map.

Right?

And finally, comes the tensor that represents the N-Dimensional array.

When we say, N dimension, it means N represents any number. It could be 3 and more than 3.

Now, it’s easy to say, 3 dimensions. But in reality, from 3 dimensions the complexity grows.

As a result, let’s try to make it simple.

In TensorFlow main objects are tensors. And as we progress we will find that the TensorFlow library manipulates tensors in many ways.

As a result, we get value. To do that, we need to import it first. And then we can check the version in Google Colab.

- To know more on TensorFlow please read the full discussion. _ [42]

Therefore, we can use TensorFlow Machine Learning concepts to develop artificially intelligent flutter apps.

As a result, we can train a model for the apps, besides, we can also use these trained-models in the apps.

Finally we can build Artificially Intelligent, Deep Learning and Machine Learning Apps for the Android Smart Phones and iOS using Flutter SDK with TensorFlow Lite.

Besides these we can also develop Flutter apps using Firebase Machine Learning Kit which we know as Firebase ML Kit.

We have discussed Machine Learning Algorithms separately on another website dedicated to TensorFlow, Machine Learning, and Artificial Intelligence.

How Artificial Intelligence helps us to move forward

We can make significant advancement and innovations using Artificial Intelligence.

As AI has become a rising star in the mobile app development services, we can use it to make more advanced Flutter apps.

[42]https://sinhasanjib.com

It’s true that the combination of Flutter and Artificial Intelligence may bring out some unique methods which is the present need for the business growth.

Are there any examples of Artificial Intelligence?

Certainly, there are.

Personalized Shopping Recommendations is one of them.

AI-driven Chat-bot has become popular. And there are many others.

Because AI has become a buzzword in the market of mobile app development, we must adopt it also.

We will discuss more on the Firebase ML kit later.

Why?

Because, there are many things to learn.

Above all, many things to come while we have been building our E Commerce App.

So stay tuned.

15. What Next

If you find these information useful, I am happy. For any Flutter related query, you may send an email to

1. sanjib12sinha@gmail.com.

And at the same time don't forget to visit the website where I write regularly on Flutter only.

You'll get the updated Flutter tips and tricks. So how about take the trips? :)

For more Flutter related Articles and Resources[43]

For more TensorFlow, Machine Learning and AI related Articles and Resources[44]

The code repositories for this book[45]

[43]https://sanjibsinha.com
[44]https://sinhasanjib.com
[45]https://github.com/sanjibsinha/