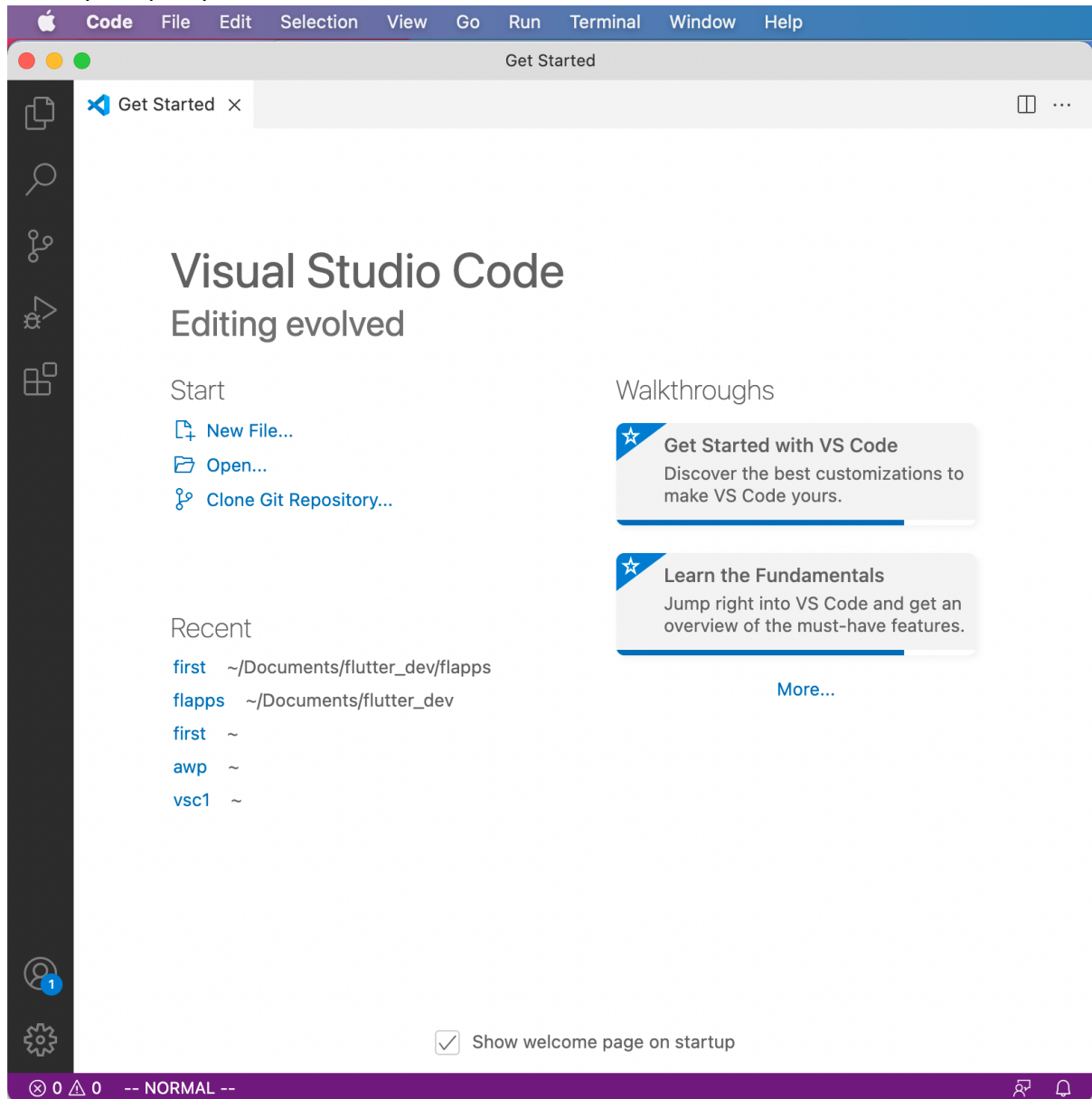


Flutter User Interface Using Scaffolds

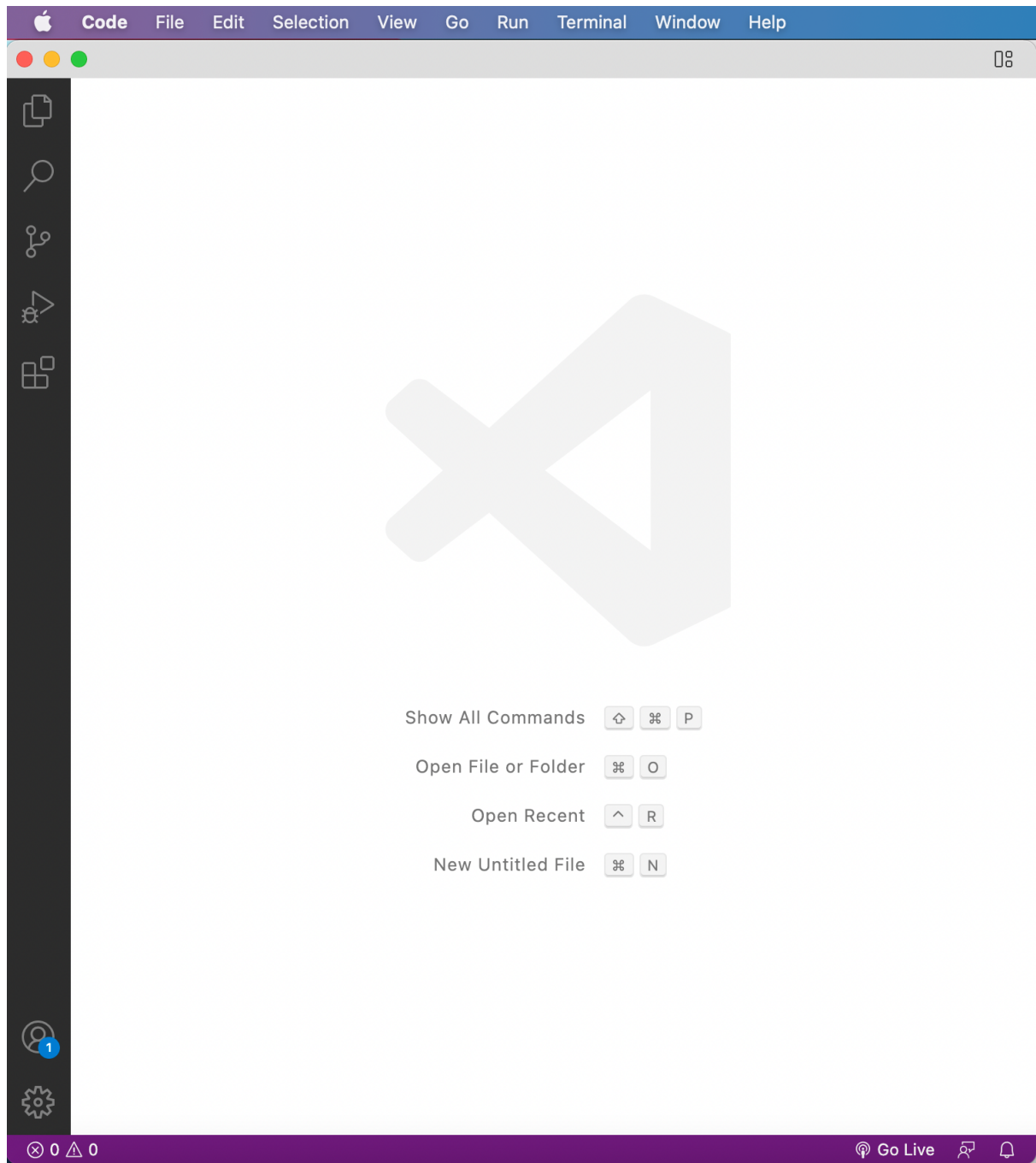
By Sazilah Salam

Module 1: Creating a Flutter App in Visual Studio Code

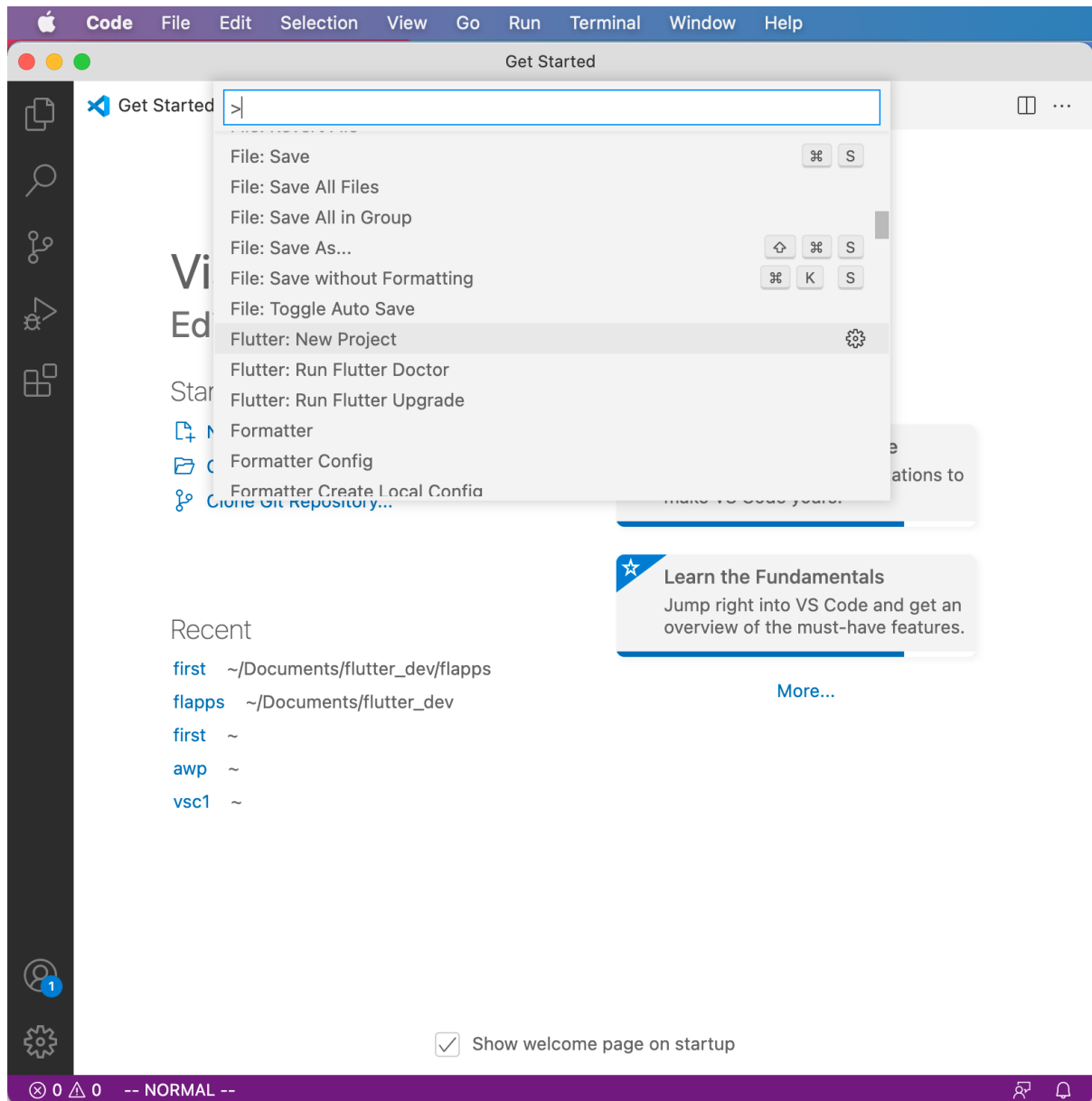
When you open your Visual Studio Code, the editor will look like this.



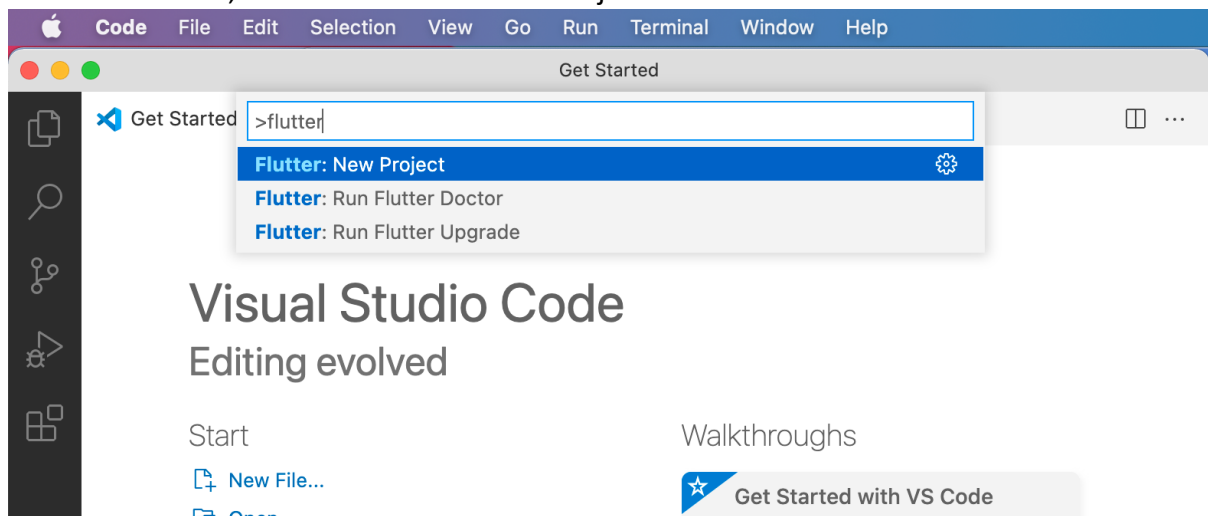
Or like this..



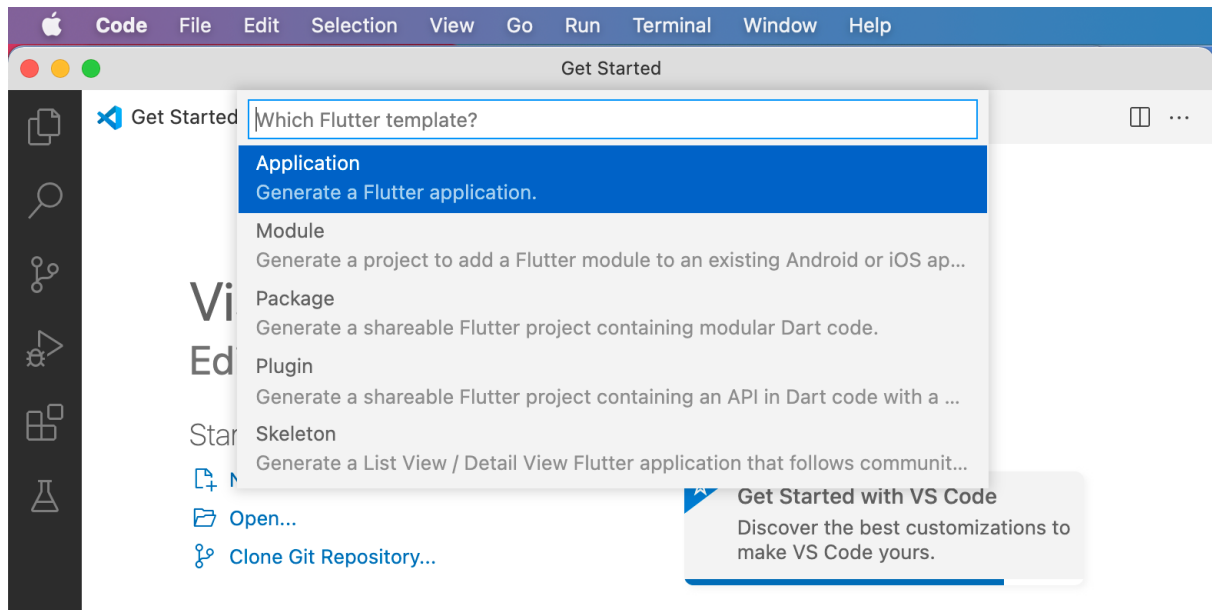
Go to **View > Command Palette > Flutter: New Project**.



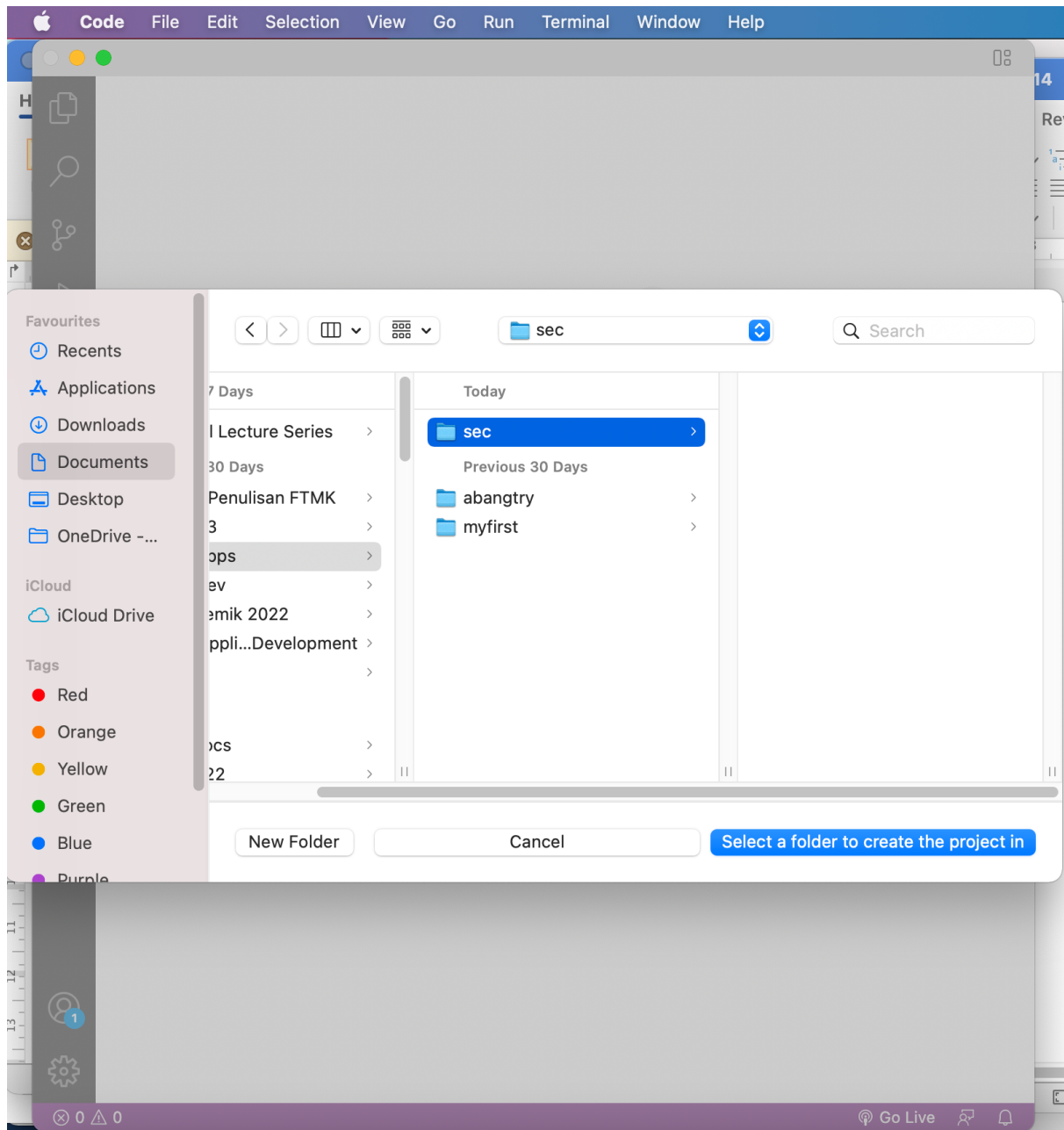
Search for flutter, and select Flutter: New Project from the search results.



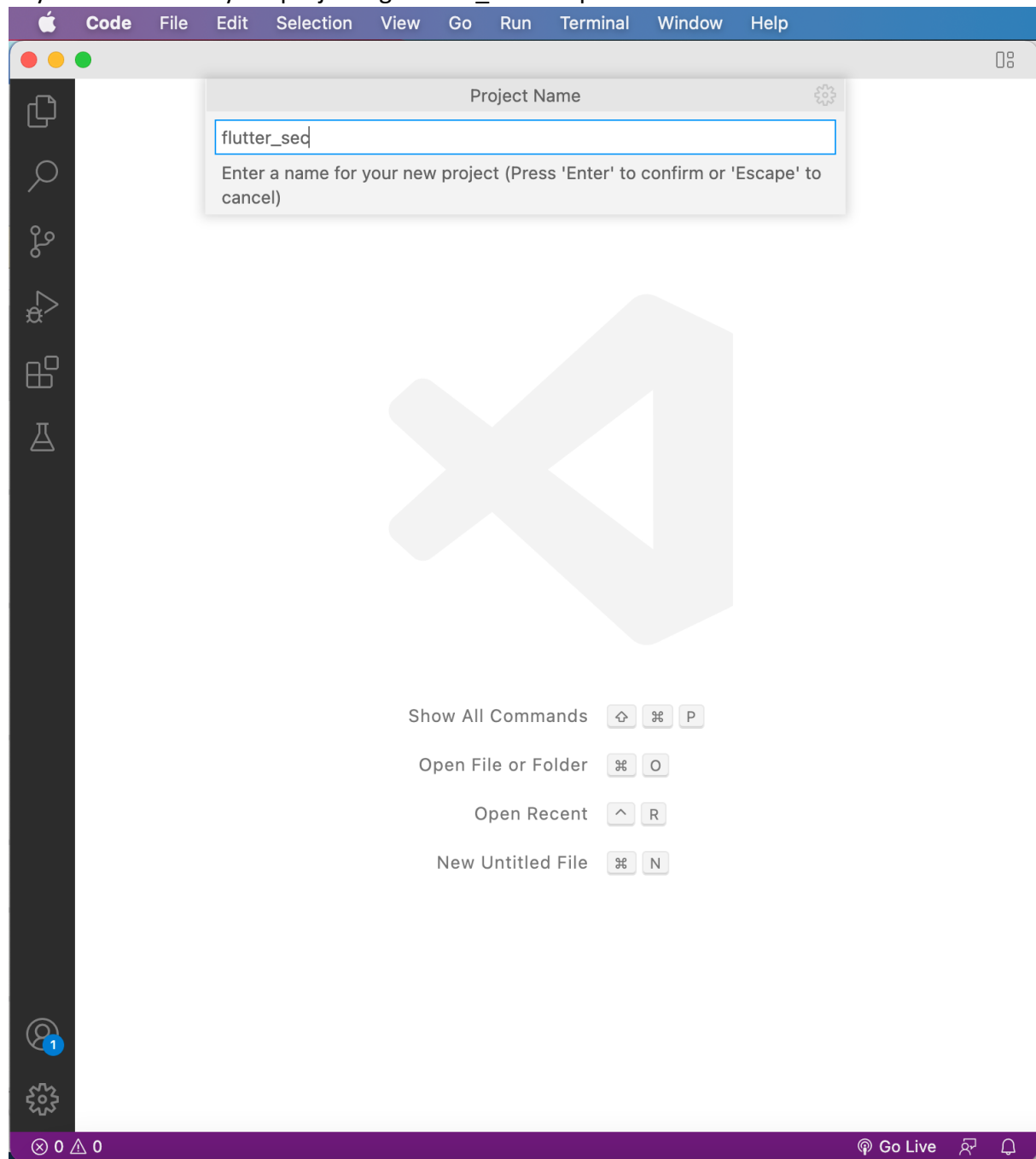
Select Application to generate a flutter application.



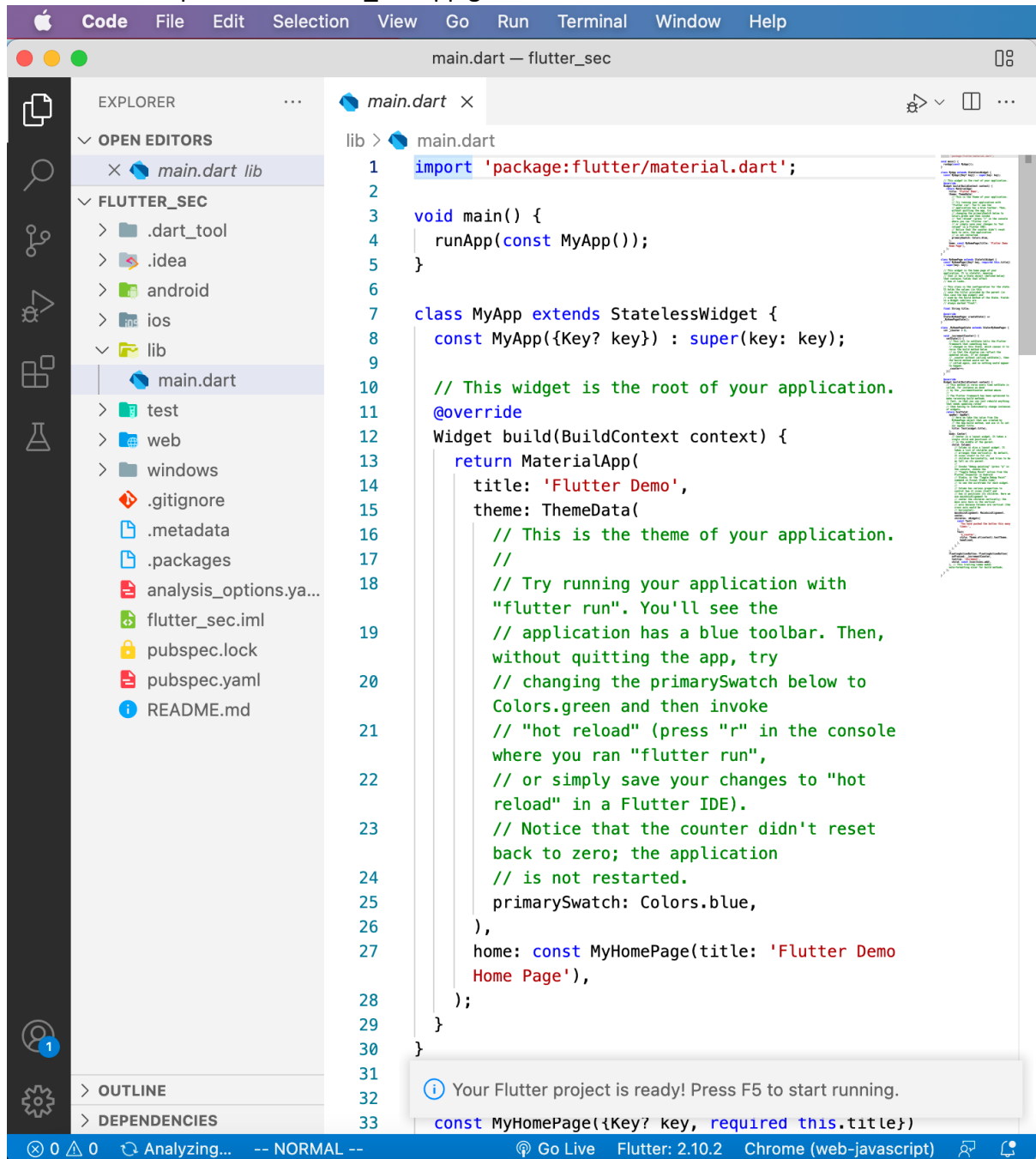
Then, you will be asked to select or create a folder. Create a new folder, for example flutter_apps. Click the Create button and then press Enter to select the folder that you have just created.



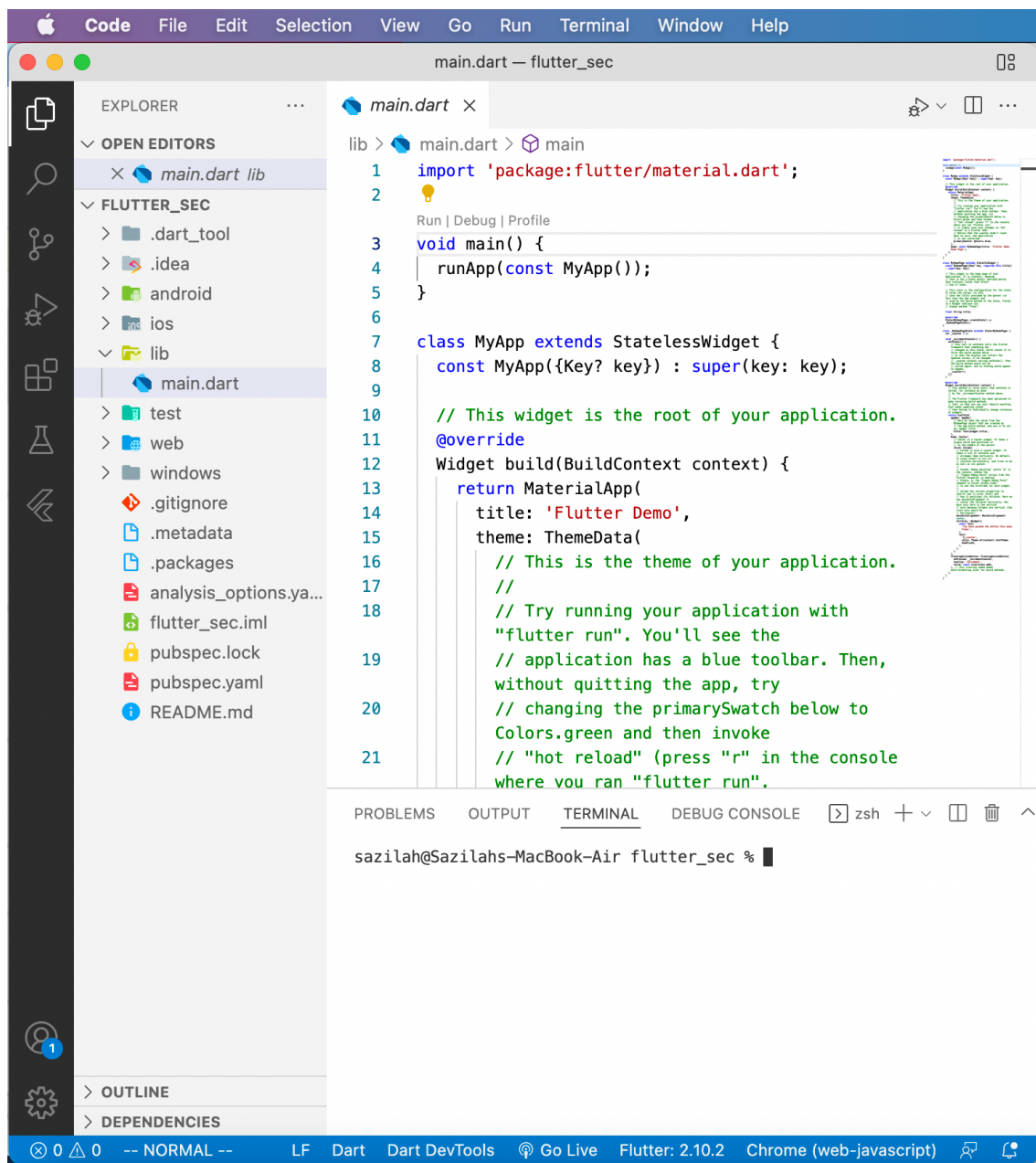
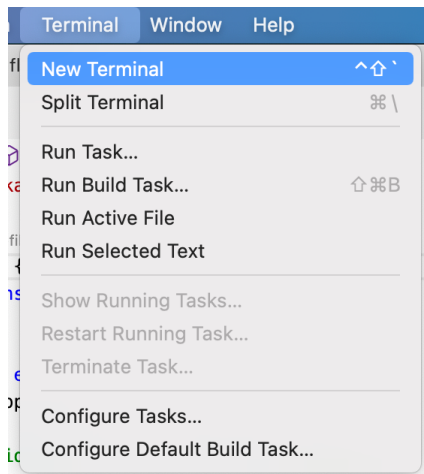
Key-in the name of your project eg. flutter_sec and press Enter.



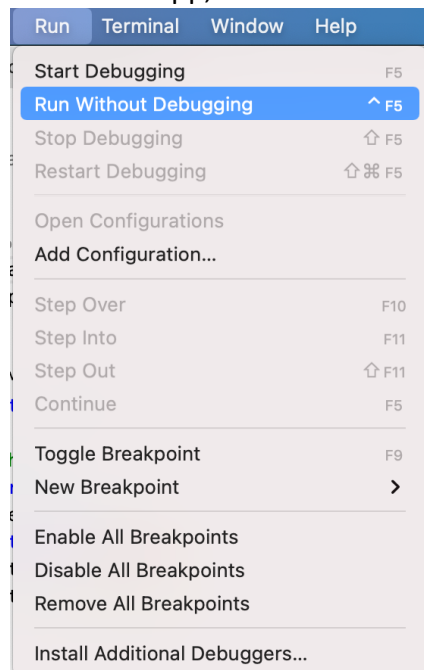
This is an example of the flutter_sec app generated.



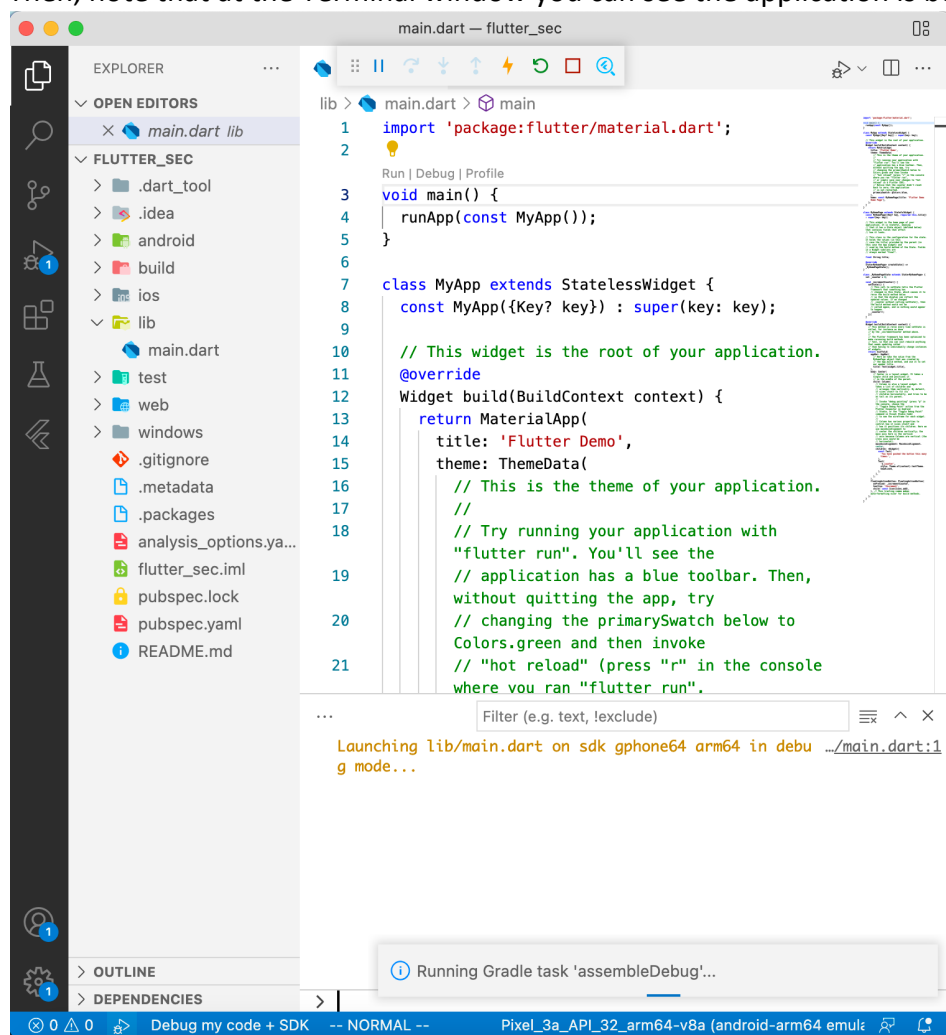
Next, open a new Terminal window inside the editor. Click Terminal>New Terminal.



To test the app, click on Run>Run Without Debugging.



Then, note that at the Terminal window you can see the application is being launched.



Once completed, the app will be shown on the emulator as follows.

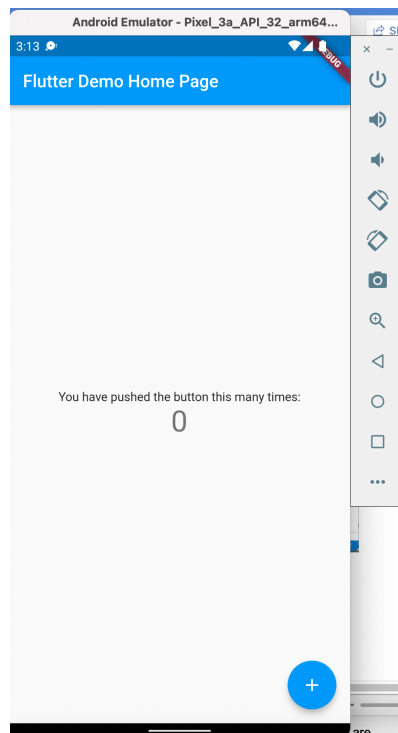
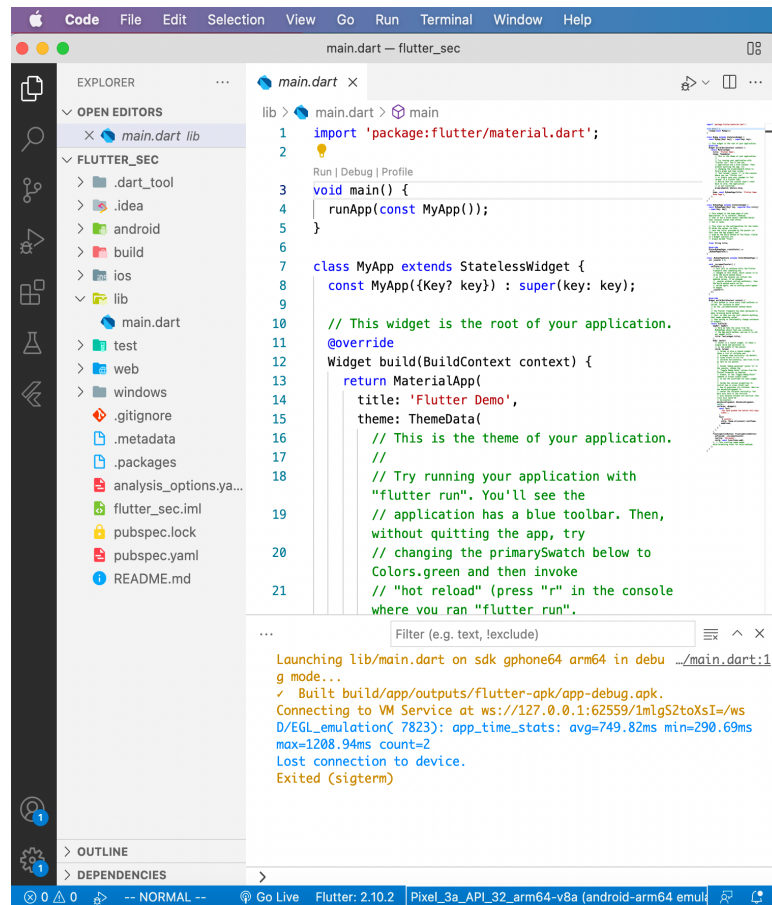
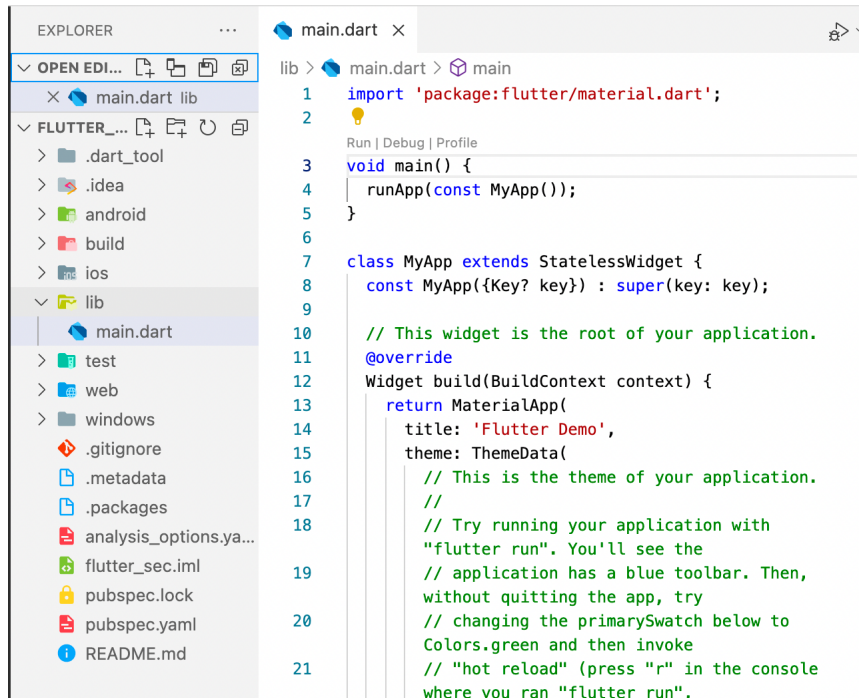


Figure 1: The app interface on an Android emulator.

Module 2: Building User Interface Using Scaffolds

Scaffolds, also known as layouts, are at the heart of creating user interfaces with Flutter. The layout we'll create will be a **MaterialApp** class with a Scaffold widget with an **AppBar** and a body property with a **FloatingActionButton**.

At the explorer window, click on lib>main.dart .



Copy the following code and paste it to replace your original main.dart.

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

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

class MyApp extends StatelessWidget { @override
Widget build(BuildContext context) {

return MaterialApp( debugShowCheckedModeBanner: false, home: Scaffold(

appBar: AppBar(
title: Text('Mobile Application Development'),

),
body: Center(

child: Text(
'My Flutter Layout Using Scaffolds', style: TextStyle(fontSize: 24),

),
```

```
),  
floatingActionButton: FloatingActionButton(  
  
    child: Icon(Icons.ac_unit),  
    onPressed: () {  
        print('onPressed..');  
    },  
  
),  
  
),  
  
);  
  
}  
  
}
```

Save the main.dart file.

Then, click **Run>Run Without Debugging**. If the emulator is already running, Flutter's hot reload mechanism will automatically update the application's user interface, which should look like Figure 2.

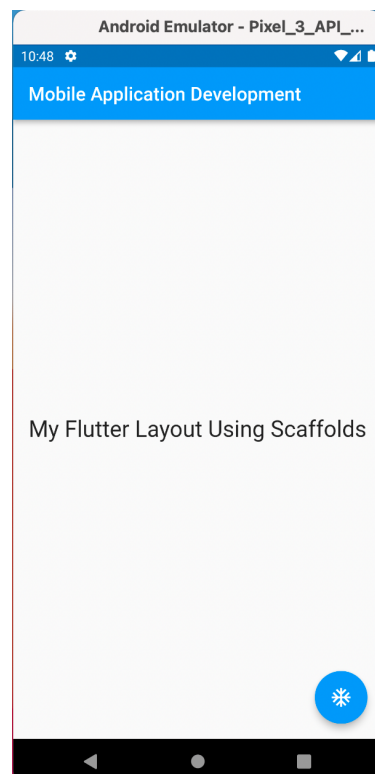


Figure 2: The app's user interface on an Android emulator.

Figure 3 shows the UI elements of the app that we have created.

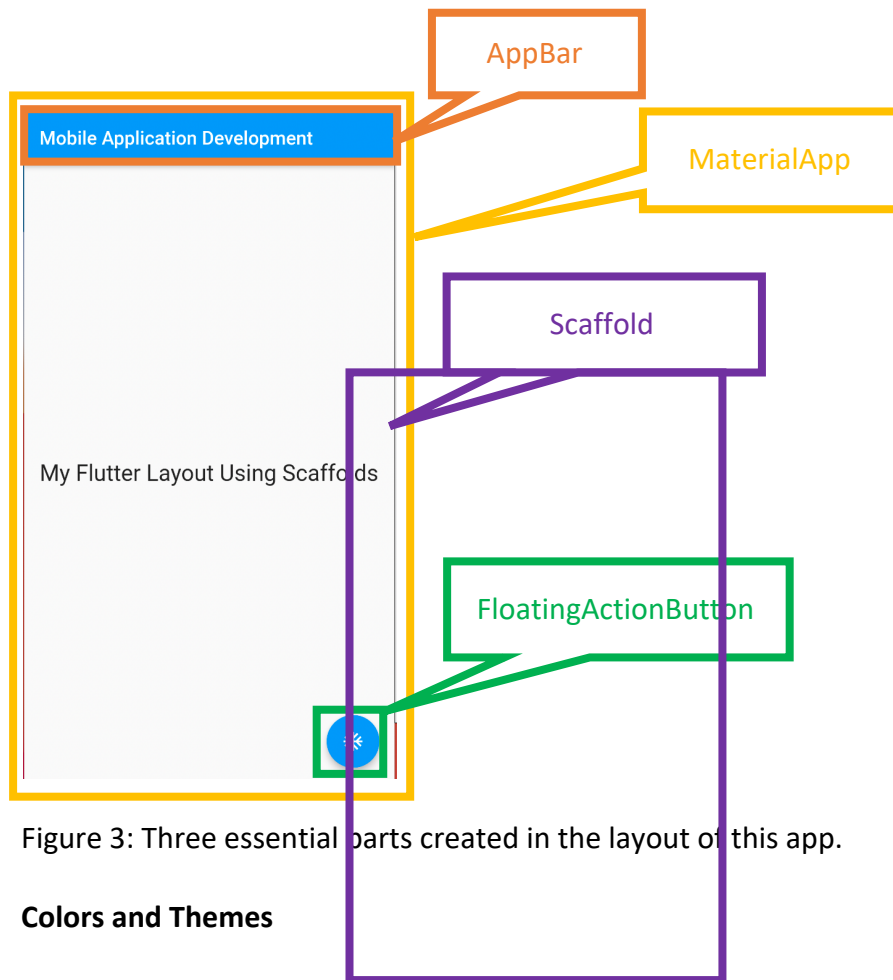


Figure 3: Three essential parts created in the layout of this app.

Colors and Themes

When we were creating the previous layout, you may have noticed that Flutter provided us with a "blue and white" application look and feel (theme) by default, which looks quite nice. This theme adheres to the Material Design guidelines.

However, we can easily customize the application's appearance while adhering to the Material Design specifications.

The first thing we can do is adjust the brightness of the app. This is very simple to accomplish by making a few changes to the main.dart code, which are highlighted in the listing below. Insert the highlighted code.

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: Scaffold(
```

```
appBar: AppBar(  
  title: Text('Mobile Application Development'),  
  
),  
body: Center(  
  
  child: Text('My Flutter Layout Using Scaffolds', style:  
    TextStyle(fontSize: 24),  
  
  ),  
  
),  
  
floatingActionButton: FloatingActionButton(  
  
  child: Icon(Icons.ac_unit),  
  onPressed: () {  
    print('onPressed..');  
  },  
  
),  
  
),  
);  
  
},  
  
}
```

theme: ThemeData(
 brightness: Brightness.dark,
),



The colours of the app's layout, as well as the text font on the app's header, can be customized. To accomplish this, we will make the following changes to the main.dart code.

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

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

class MyApp extends StatelessWidget { @override
Widget build(BuildContext context) {

return MaterialApp( debugShowCheckedModeBanner: false, home: Scaffold(

appBar: AppBar(
title: Text('Mobile Application Development'),

),
body: Center(
child: Text(
'My Flutter Layout Using Scaffolds', style: TextStyle(fontSize: 24),
),
),
floatingActionButton: FloatingActionButton(
child: Icon(Icons.ac_unit),
onPressed: () {
print('onPressed..');
}
```

```
    },  
  ),  
  ),  
  theme: ThemeData(  
    primaryColor: Colors.indigo,  
    accentColor: Colors.amber,  
    textTheme: TextTheme(  
      bodyText2: TextStyle(fontSize: 26, fontStyle: FontStyle.italic),  
    ),  
    brightness: Brightness.dark,  
  ),  
);  
}  
}
```

