

# **ICT4153**

## **Mobile Application Development**

### **Practical 03**

## **Create first Flutter Project – 01**

### **Objective**

**To set up Android Studio for Flutter development and run a sample Flutter application.**

### **Creating an empty template project**

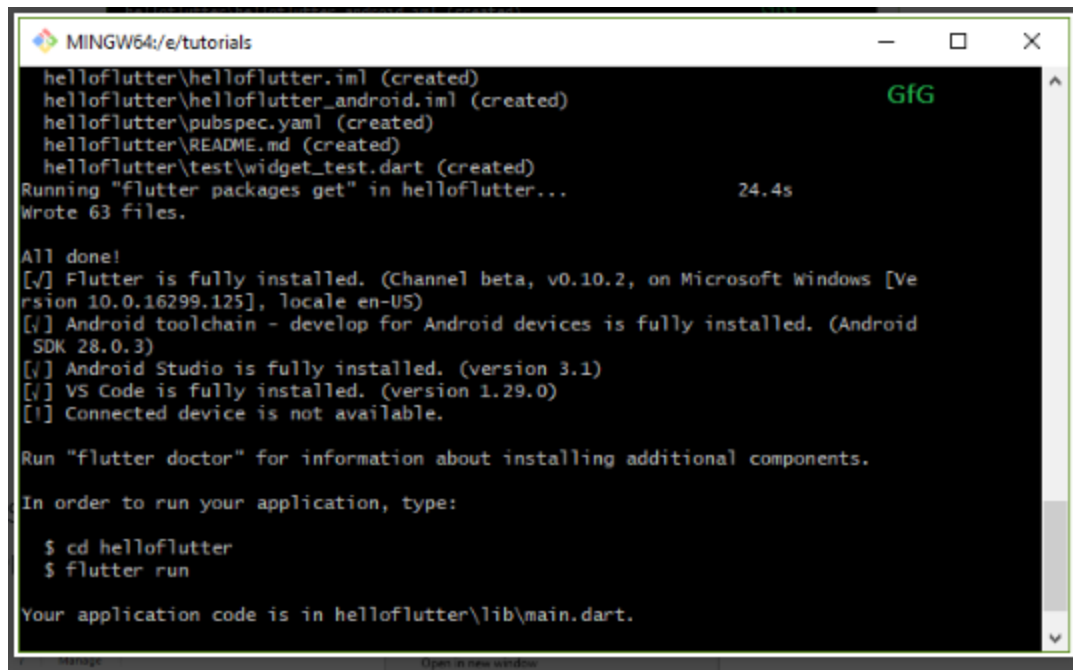
1. Navigate to the place where you want your project to be created. Open a command prompt (you can also use the context '**Git Bash here**' by right clicking, to open Git Bash in this location) and type in the command for creating a new project:

**flutter create project\_name**

2. For example: for a project named '**helloflutter**' executing

**flutter create helloflutter**

3. will create a new Flutter project with name *helloflutter*



```
MINGW64/e/tutorials
helloflutter\helloflutter.iml (created)
helloflutter\helloflutter_android.iml (created)
helloflutter\pubspec.yaml (created)
helloflutter\README.md (created)
helloflutter\test\widget_test.dart (created)
Running "flutter packages get" in helloflutter... 24.4s
Wrote 63 files.

All done!
[✓] Flutter is fully installed. (Channel beta, v0.10.2, on Microsoft Windows [Version 10.0.16299.125], locale en-US)
[✓] Android toolchain - develop for Android devices is fully installed. (Android SDK 28.0.3)
[✓] Android Studio is fully installed. (version 3.1)
[✓] VS Code is fully installed. (version 1.29.0)
[!] Connected device is not available.

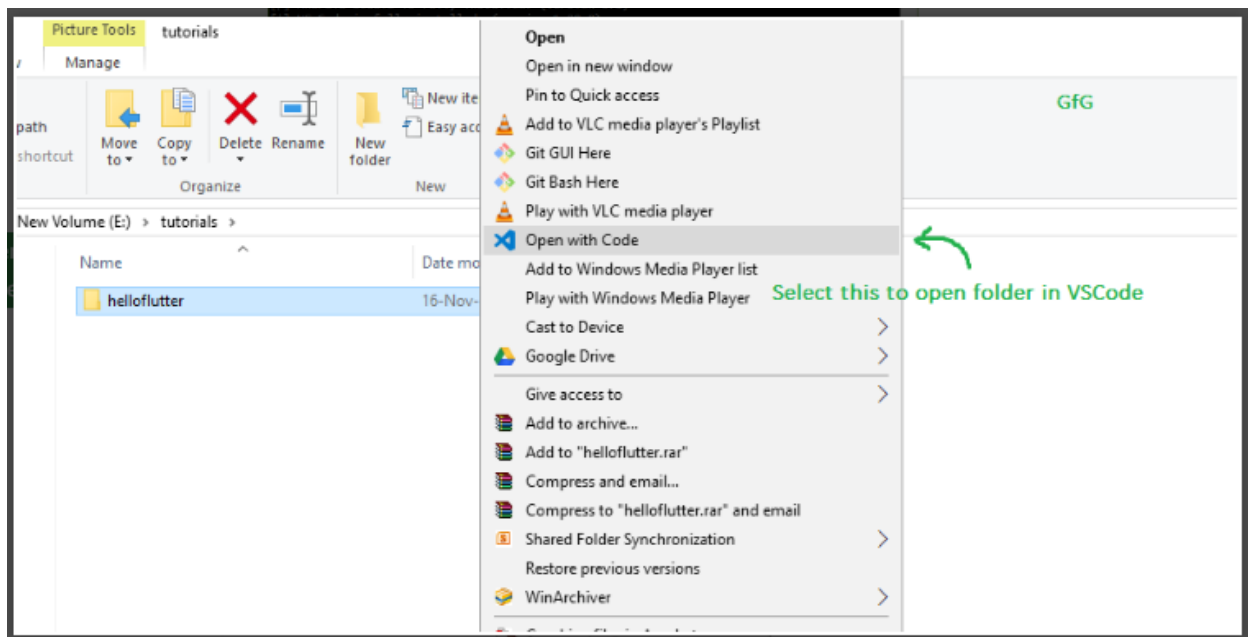
Run "flutter doctor" for information about installing additional components.

In order to run your application, type:

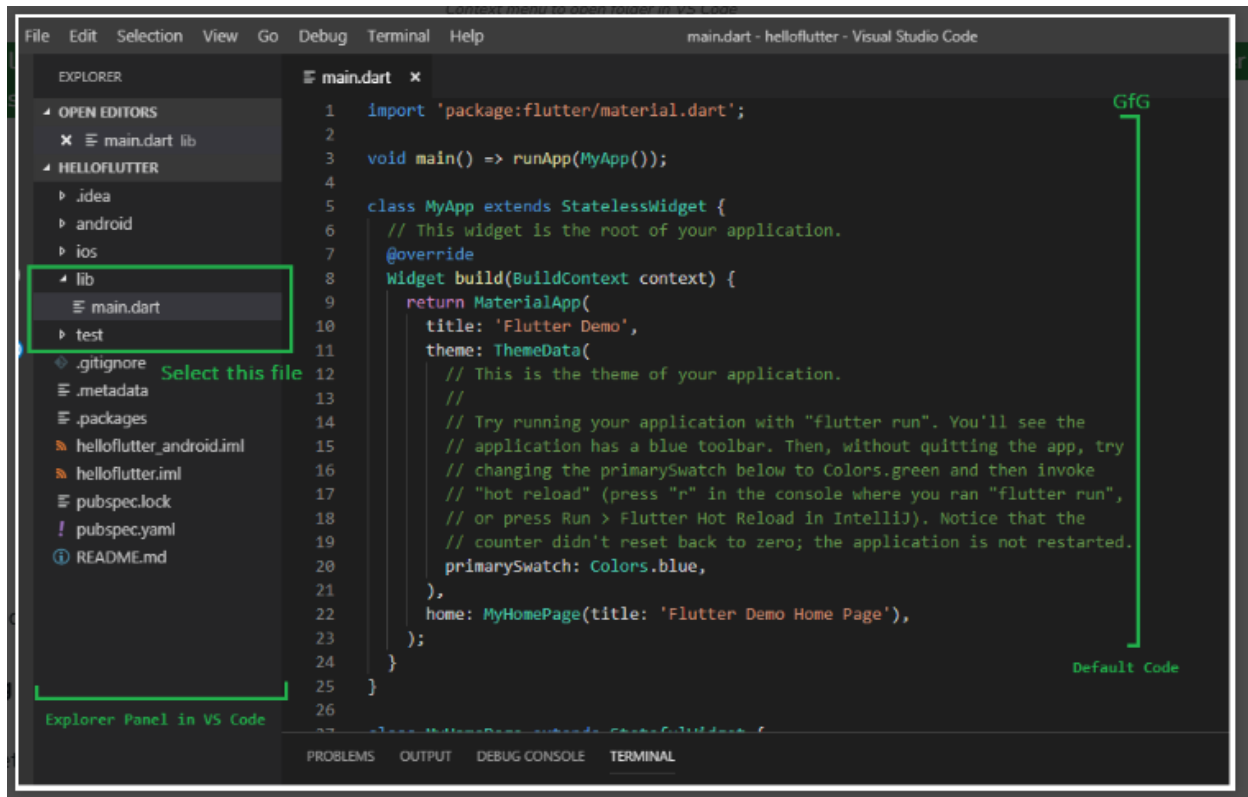
$ cd helloflutter
$ flutter run

Your application code is in helloflutter\lib\main.dart.
```

4. Open this folder in VS Code. You can right-click and use the context menu to open directly into VS Code, or start VS Code first and then open this folder as a project.



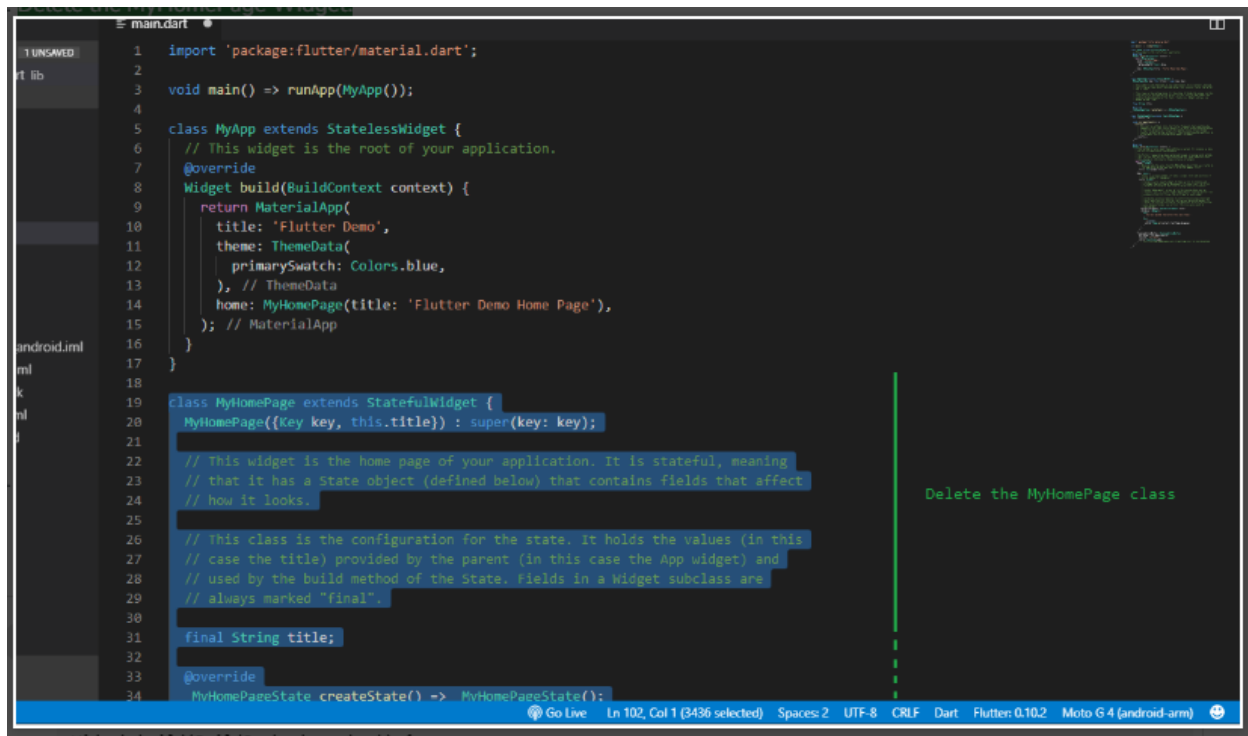
5. The large panel on the left that displays all the files and folders is known as the **Explorer Panel**. Navigate to 'lib' folder and select the 'main.dart' file. This file is the entry point from where the app starts its execution.



6. The code that opens up is that of the template application. Try running this simple app right away!

### **Saying Hello Flutter!**

7. Delete the MyHomePage Widget.



```
main.dart
1 import 'package:flutter/material.dart';
2
3 void main() => runApp(MyApp());
4
5 class MyApp extends StatelessWidget {
6   // This widget is the root of your application.
7   @override
8   Widget build(BuildContext context) {
9     return MaterialApp(
10       title: 'Flutter Demo',
11       theme: ThemeData(
12         primarySwatch: Colors.blue,
13       ), // ThemeData
14       home: MyHomePage(title: 'Flutter Demo Home Page'),
15     ); // MaterialApp
16   }
17 }
18
19 class MyHomePage extends StatefulWidget {
20   MyHomePage({Key key, this.title}) : super(key: key);
21
22   // This widget is the home page of your application. It is stateful, meaning
23   // that it has a State object (defined below) that contains fields that affect
24   // how it looks.
25
26   // This class is the configuration for the state. It holds the values (in this
27   // case the title) provided by the parent (in this case the App widget) and
28   // used by the build method of the State. Fields in a Widget subclass are
29   // always marked "final".
30
31   final String title;
32
33   @override
34   MyHomePageState createState() => MyHomePageState();
35 }
```

Delete the MyHomePage class

8. Create a new Stateless Widget and name it HelloFlutter. [Stateless Widgets](#) are used to define Widgets which don't have to deal with changes to its internal state. They are mostly used to build components which once drawn, are not required to update.

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

  @override
  Widget build(BuildContext context) {
    return Container(

    );
  }
}
```

```
13         ), // ThemeData
14         home: MyHomePage(title: 'Flutter Demo Home Page'),
15     ); // MaterialApp
16 }
17 }
18
19 class HelloFlutter extends StatelessWidget {
20     @override
21     Widget build(BuildContext context) {
22         return Container(
23
24         );
25     }
26 }
```

GfG

new Widget named HelloFlutter

9. Replace the Container widget with a Scaffold widget: A [Scaffold](#) implements basic material design visual layout structure. This Widget provides APIs for showing drawers, appbars and the body of the app. The body property of the Scaffold will be used here to display the contents of the app.

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

    @override
    Widget build(BuildContext context) {
        return const Scaffold(

        );
    }
}
```

10. Declare a Container Widget in the body of the Scaffold. A [Container](#) Widget is a useful widget which combines common painting, positioning, and sizing widgets. You can wrap any widget with a Container and control the above mentioned properties.

```
class HelloFlutter extends StatelessWidget {  
  const HelloFlutter({Key? key}) : super(key: key);  
  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      body: Container(  
  
      ),  
    );  
  }  
}
```

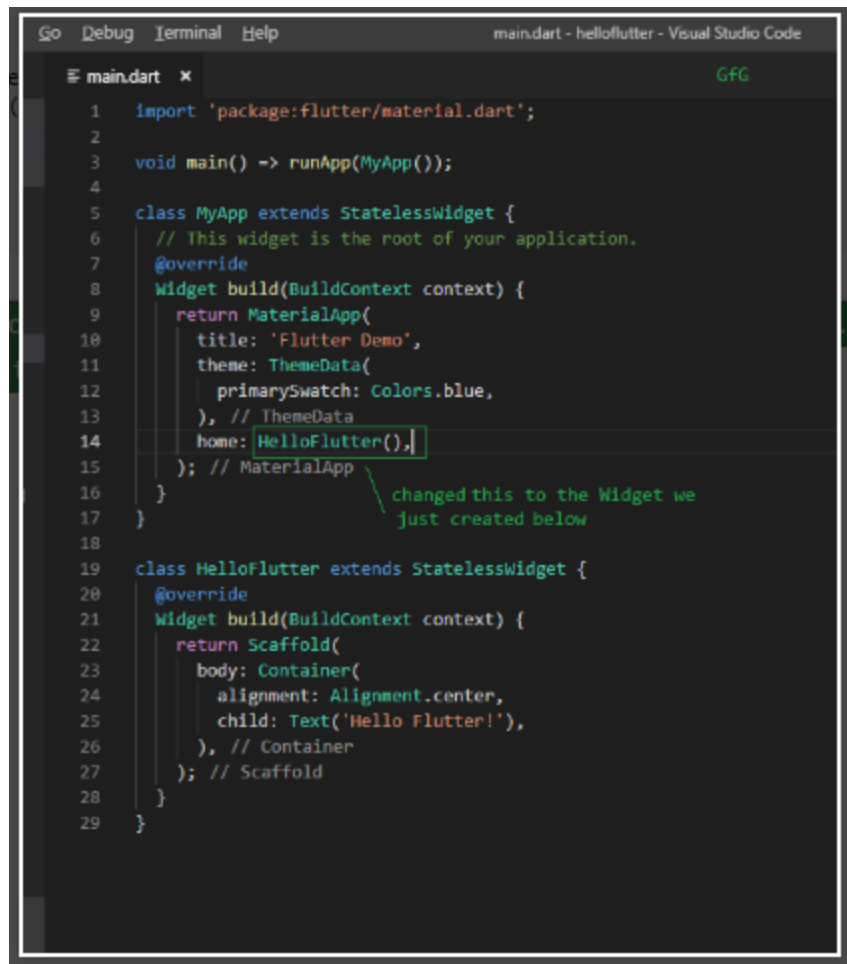
11. The Container widget has an alignment property which will help to position the Widget to the center of the screen. Set the alignment with the Alignment class:

```
alignment: Alignment.center
```

12. In the child property of the Container Widget, declare a Text Widget: The [Text](#) Widget deals with displaying and handling text. After creating the Text Widget, put in 'Hello Flutter' between the parentheses in single quotes. Whatever is put in between the single quotes is displayed by the Text Widget.

```
class HelloFlutter extends StatelessWidget {  
  const HelloFlutter({Key? key}) : super(key: key);  
  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      body: Container(  
        alignment: Alignment.center,  
        child: const Text('Hello Flutter!'),  
      ),  
    );  
  }  
}
```

13. Finally, in the home property of the main My App class above, change it from MyHomePage(...) to HelloFlutter(). This allows the main MyApp class to refer to the Hello Flutter just created.



```
1  import 'package:flutter/material.dart';
2
3  void main() => runApp(MyApp());
4
5  class MyApp extends StatelessWidget {
6    // This widget is the root of your application.
7    @override
8    Widget build(BuildContext context) {
9      return MaterialApp(
10        title: 'Flutter Demo',
11        theme: ThemeData(
12          primarySwatch: Colors.blue,
13        ), // ThemeData
14        home: HelloFlutter(),
15      ); // MaterialApp
16    }
17  }
18
19  class HelloFlutter extends StatelessWidget {
20    @override
21    Widget build(BuildContext context) {
22      return Scaffold(
23        body: Container(
24          alignment: Alignment.center,
25          child: Text('Hello Flutter!'),
26        ), // Container
27      ); // Scaffold
28    }
29  }
```

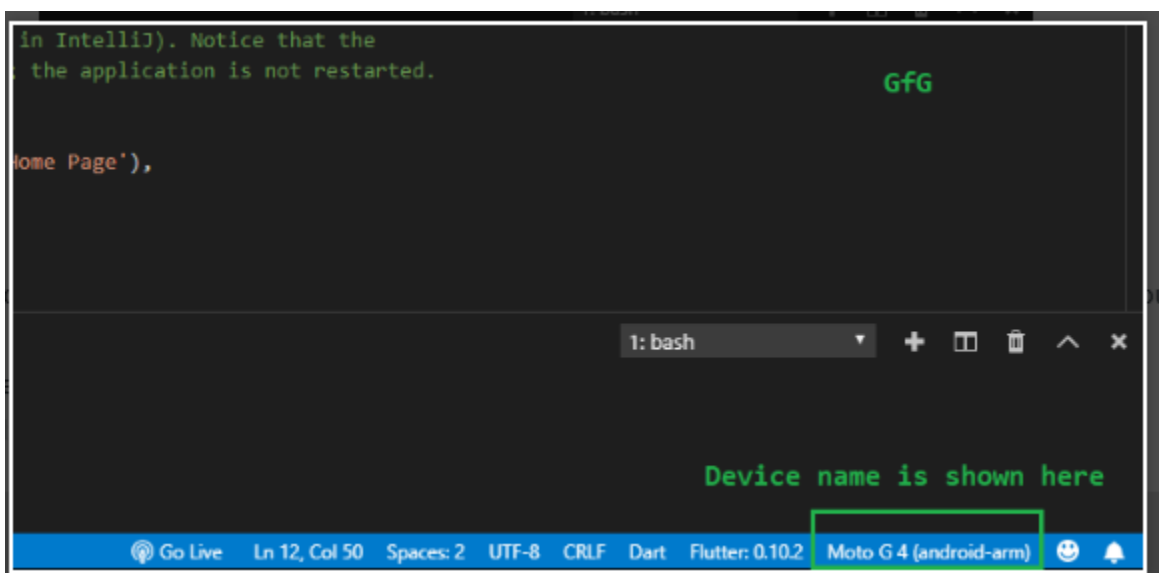
14. Now run the app using the *'flutter run'* command.



15. A text 'Hello Flutter!' will appear written in the middle of the screen.

### **Running the HelloFlutter App**

16. Connect a physical device to the PC and enable [Developer Mode](#). If the device is successfully recognised by the PC, the device name would appear in the lower-right corner of VS Code.

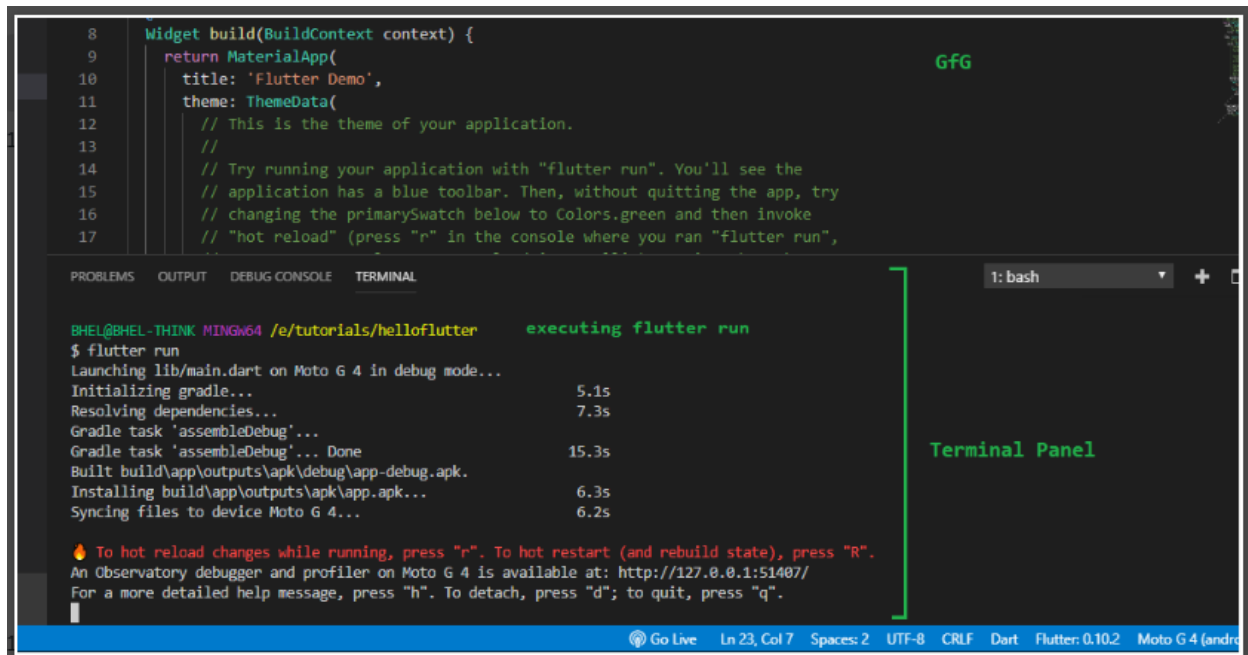




17. If you would like to setup an emulator instead, see: [Set up the Android emulator](#). The emulated device would also show up here similarly.
18. Open the integrated terminal by pressing the key combination [CTRL + `] (Control key + backtick).
19. Run the command:

```
flutter run
```

20. Wait for a few minutes. As this is first run, some downloads and installation take place in the background related to gradle. Subsequent compilations would be a lot faster.



The screenshot shows an IDE with a Dart file and a terminal panel. The Dart code is a simple Flutter widget that displays 'GfG'. The terminal panel shows the output of the 'flutter run' command, including the initialization of gradle, resolving dependencies, and building the app for a Moto G 4 device. The output indicates that the app was successfully installed and is running on the device.

```
8   Widget build(BuildContext context) {  
9     return MaterialApp(  
10       title: 'Flutter Demo',  
11       theme: ThemeData(  
12         // This is the theme of your application.  
13         //  
14         // Try running your application with "flutter run". You'll see the  
15         // application has a blue toolbar. Then, without quitting the app, try  
16         // changing the primarySwatch below to Colors.green and then invoke  
17         // "hot reload" (press "r" in the console where you ran "flutter run",  
18         // or press the hot key combination "r" in the editor).  
19         primarySwatch: Colors.green,  
20       ),  
21     );  
22   }  
23 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

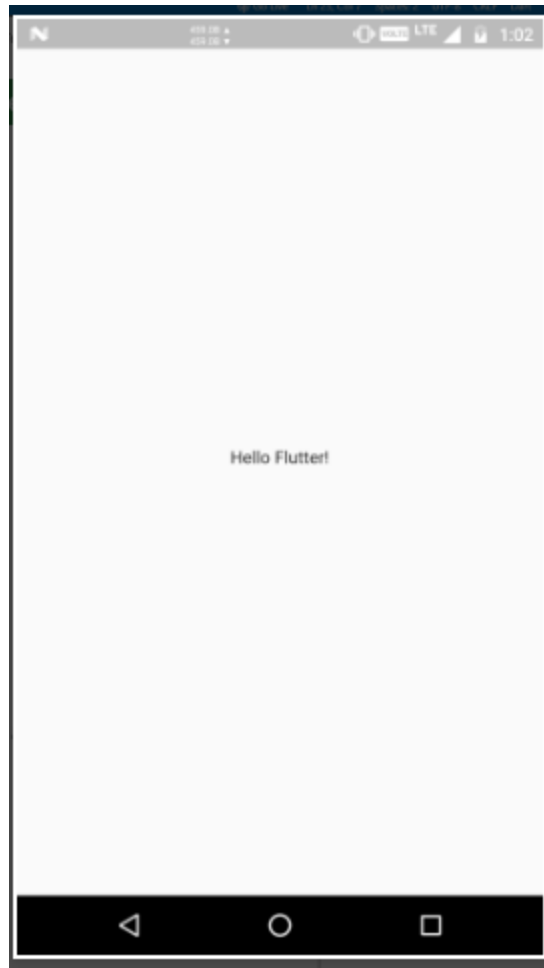
1: bash

executing flutter run

```
$ flutter run  
Launching lib/main.dart on Moto G 4 in debug mode...  
Initializing gradle... 5.1s  
Resolving dependencies... 7.3s  
Gradle task 'assembleDebug'... Done 15.3s  
Built build\app\outputs\apk\debug\app-debug.apk.  
Installing build\app\outputs\apk\app.apk... 6.3s  
Syncing files to device Moto G 4... 6.2s  
  
To hot reload changes while running, press "r". To hot restart (and rebuild state), press "R".  
An Observatory debugger and profiler on Moto G 4 is available at: http://127.0.0.1:51407/  
For a more detailed help message, press "h". To detach, press "d"; to quit, press "q".
```

Go Live Ln 23, Col 7 Spaces: 2 UTF-8 CRLF Dart Flutter: 0.10.2 Moto G 4 (andro

21. After compilation, the app will get installed and run on the connected device or emulator automatically.



22. Stop the app by pressing 'd' in the terminal. This is what the compilation and running of any app will be like.