

ICT4153

Mobile Application Development

Practical 04

Create first Flutter Project - 02

Objective

To develop a basic Flutter application that displays "Welcome to Flutter" on the screen.

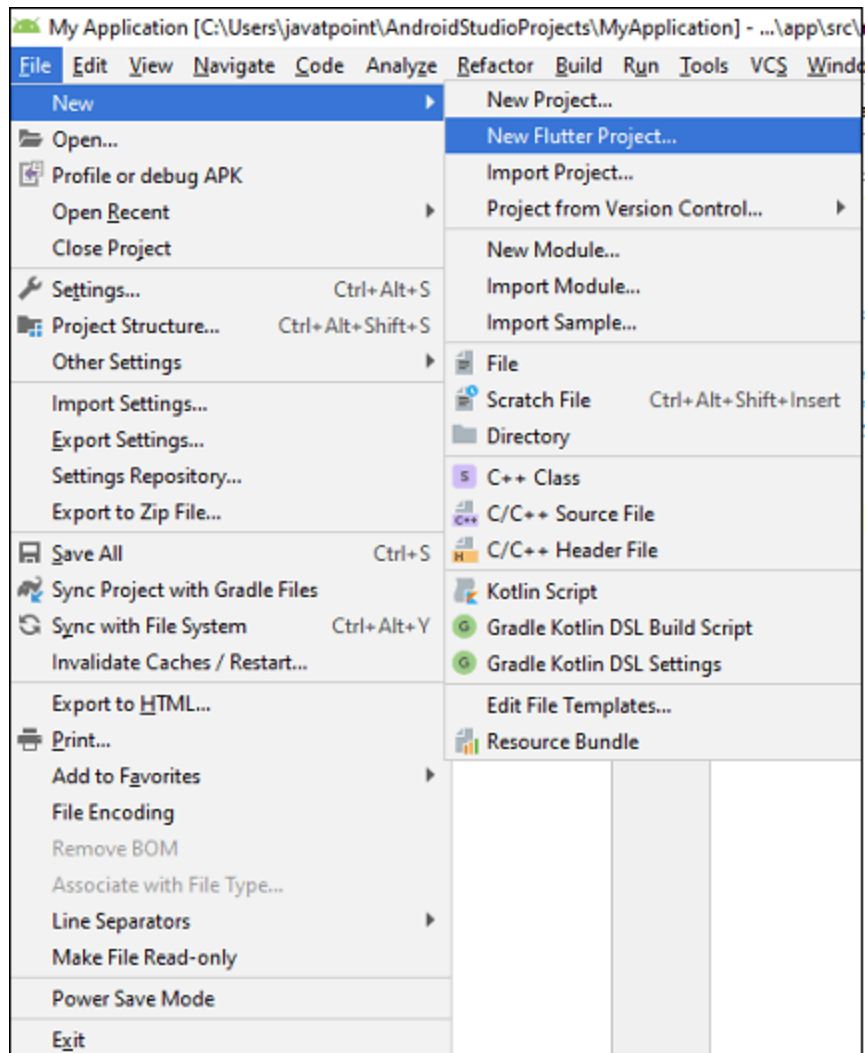
Prerequisites

- 1. Install Flutter SDK: Follow the official installation guide: Flutter Install.**
- 2. Install Android Studio or Visual Studio Code:**
 - **Add the Flutter and Dart plugins to your editor.**
- 3. Set up an Android/iOS Emulator or connect a physical device with USB debugging enabled.**

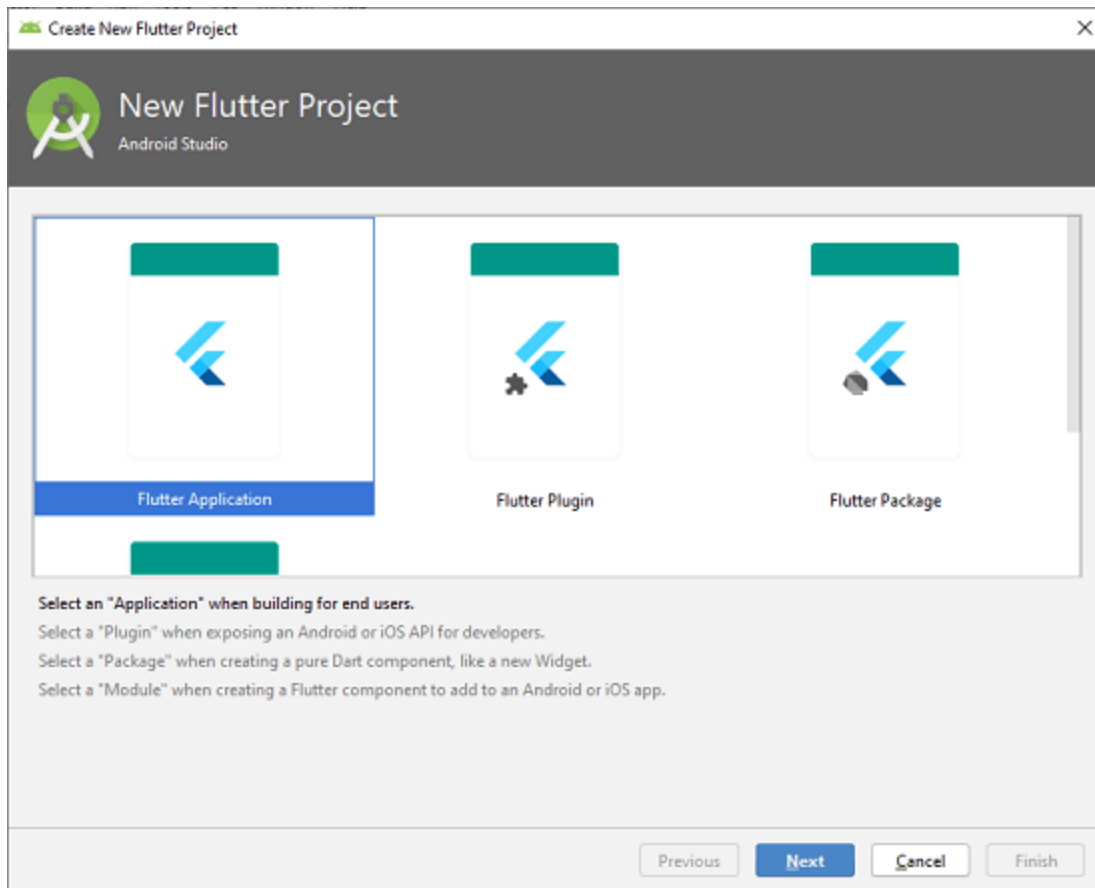
In this section, we are going to learn how to create a simple application in Android Studio to understand the basics of the Flutter application. To create Flutter application, do the following steps:

Step 1: Open the Android Studio.

Step 2: Create the Flutter project. To create a project, go to File-> New->New Flutter Project. The following screen helps to understand it more clearly.



Step 3: In the next wizard, you need to choose the Flutter Application. For this, select Flutter Application-> click Next, as shown in the below screen.



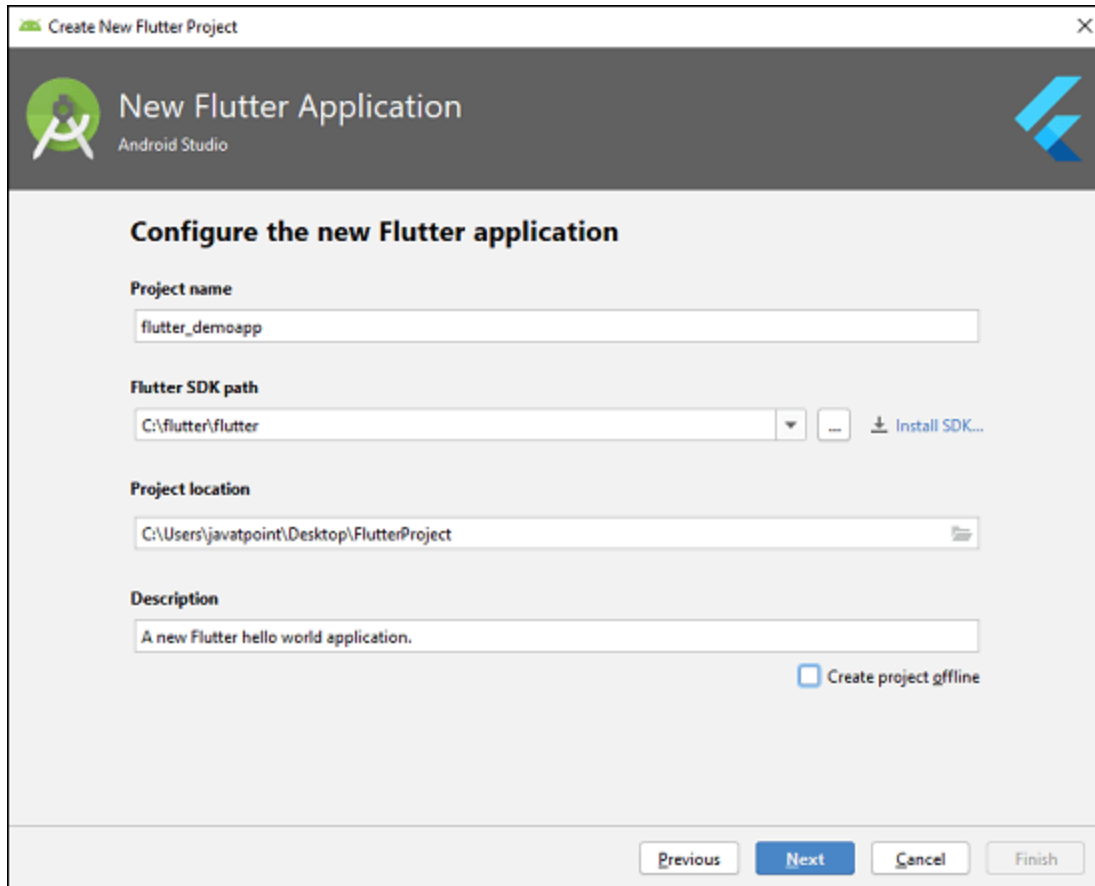
Step 4: Next, configure the application details as shown in the below screen and click on the Next button.

Project Name: Write your Application Name.

Flutter SDK Path: <path_to_flutter_sdk>

Project Location: <path_to_project_folder>

Descriptions: <A new Flutter hello world application>.



Create New Flutter Project

New Flutter Application
Android Studio

Configure the new Flutter application

Project name
flutter_demoapp

Flutter SDK path
C:\flutter\flutter ... [Install SDK...](#)

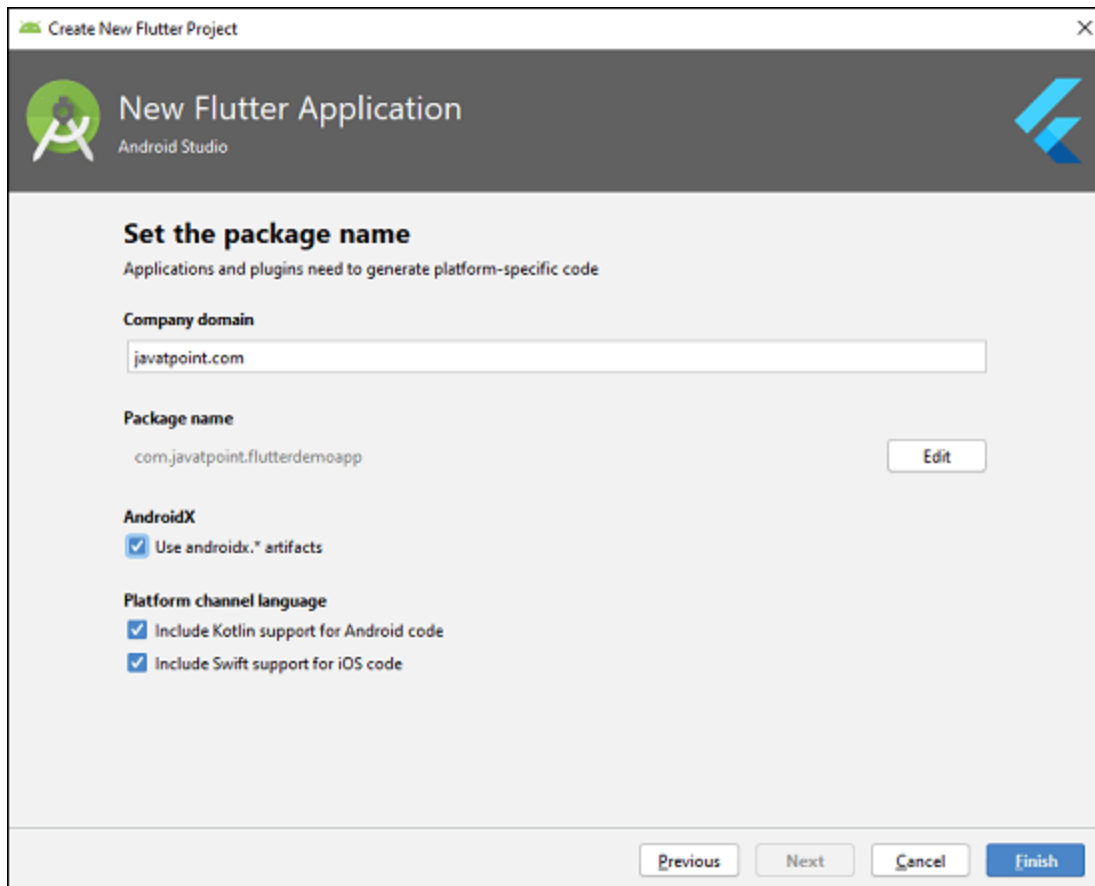
Project location
C:\Users\javatpoint\Desktop\FlutterProject

Description
A new Flutter hello world application.

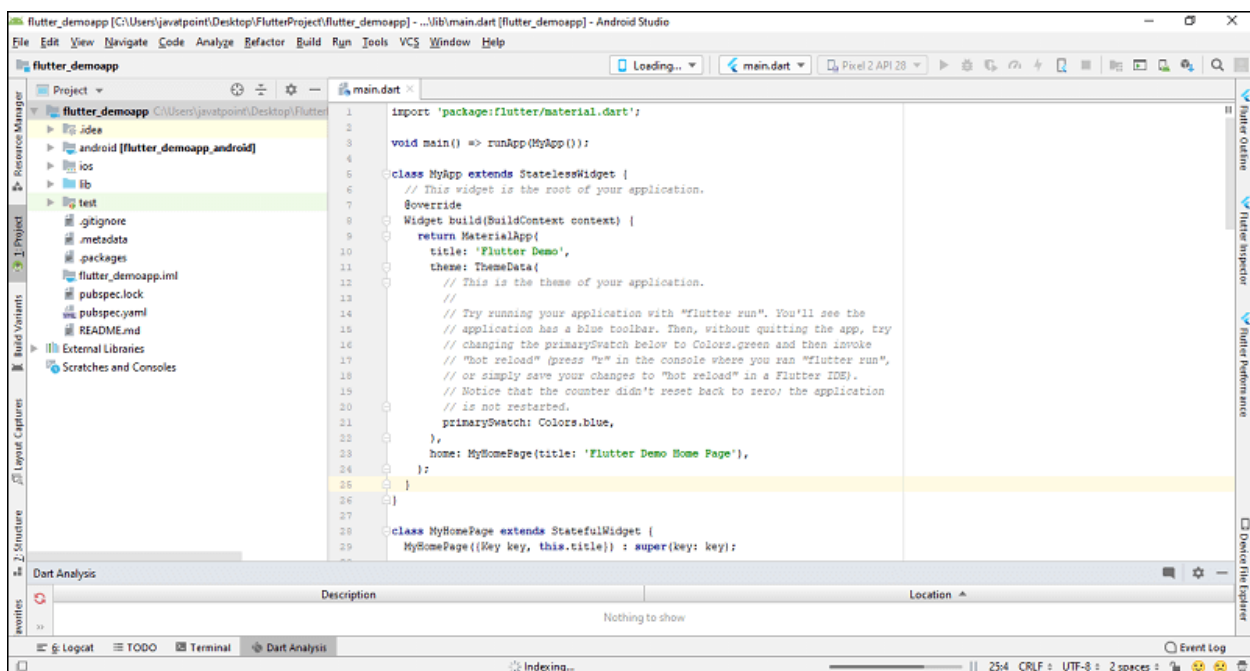
☐ Create project offline

Previous Next Cancel Finish

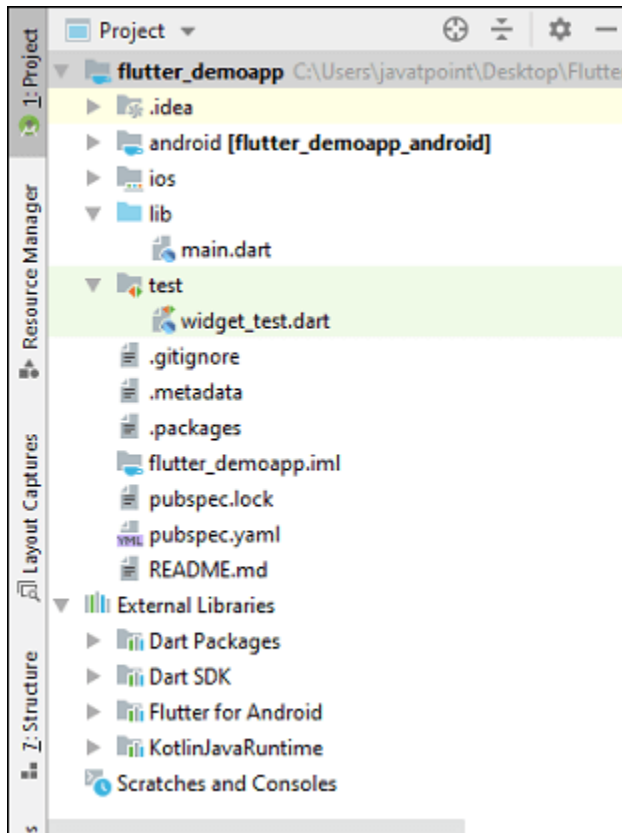
Step 5: In the next wizard, you need to set the company domain name and click the Finish button.



After clicking the Finish button, it will take some time to create a project. When the project is created, you will get a fully working Flutter application with minimal functionality.



Step 6: Now, let us check the structure of the Flutter project application and its purpose. In the below image, you can see the various folders and components of the Flutter application structure, which are going to discuss here.



.idea: This folder is at the very top of the project structure, which holds the configuration for Android Studio. It doesn't matter because we are not going to work with Android Studio so that the content of this folder can be ignored.

.android: This folder holds a complete Android project and used when you build the Flutter application for Android. When the Flutter code is compiled into the native code, it will get injected into this Android project, so that the result is a native Android application. **For**

Example: When you are using the Android emulator, this Android project is used to build the Android app, which further deployed to the Android Virtual Device.

.ios: This folder holds a complete Mac project and used when you build the Flutter application for iOS. It is similar to the android folder that is used when developing an app for Android. When the Flutter code is compiled into the native code, it will get injected into this iOS project, so that the result is a native iOS application. Building a Flutter application for iOS is only possible when you are working on macOS.

.lib: It is an essential folder, which stands for the library. It is a folder where we will do our 99 percent of project work. Inside the lib folder, we will find the Dart files which contain the code

of our Flutter application. By default, this folder contains the file **main.dart**, which is the entry file of the Flutter application.

.test: This folder contains a Dart code, which is written for the Flutter application to perform the automated test when building the app. It won't be too important for us here.

We can also have some default files in the Flutter application. In 99.99 percent of cases, we don't touch these files manually. These files are:

.gitignore: It is a text file containing a list of files, file extensions, and folders that tells Git which files should be ignored in a project. Git is a version-control file for tracking changes in source code during software development Git.

.metadata: It is an auto-generated file by the flutter tools, which is used to track the properties of the Flutter project. This file performs the internal tasks, so you do not need to edit the content manually at any time.

.packages: It is an auto-generated file by the Flutter SDK, which is used to contain a list of dependencies for your Flutter project.

flutter_demoapp.iml: It is always named according to the Flutter project's name that contains additional settings of the project. This file performs the internal tasks, which is managed by the Flutter SDK, so you do not need to edit the content manually at any time.

pubspec.yaml: It is the project's configuration file that will use a lot during working with the Flutter project. It allows you how your application works. This file contains:

- Project general settings such as name, description, and version of the project.
- Project dependencies.
- Project assets (e.g., images).

pubspec.lock: It is an auto-generated file based on the **.yaml** file. It holds more detail setup about all dependencies.

README.md: It is an auto-generated file that holds information about the project. We can edit this file if we want to share information with the developers.

Step 7: Open the **main.dart** file and replace the code with the following code snippets.

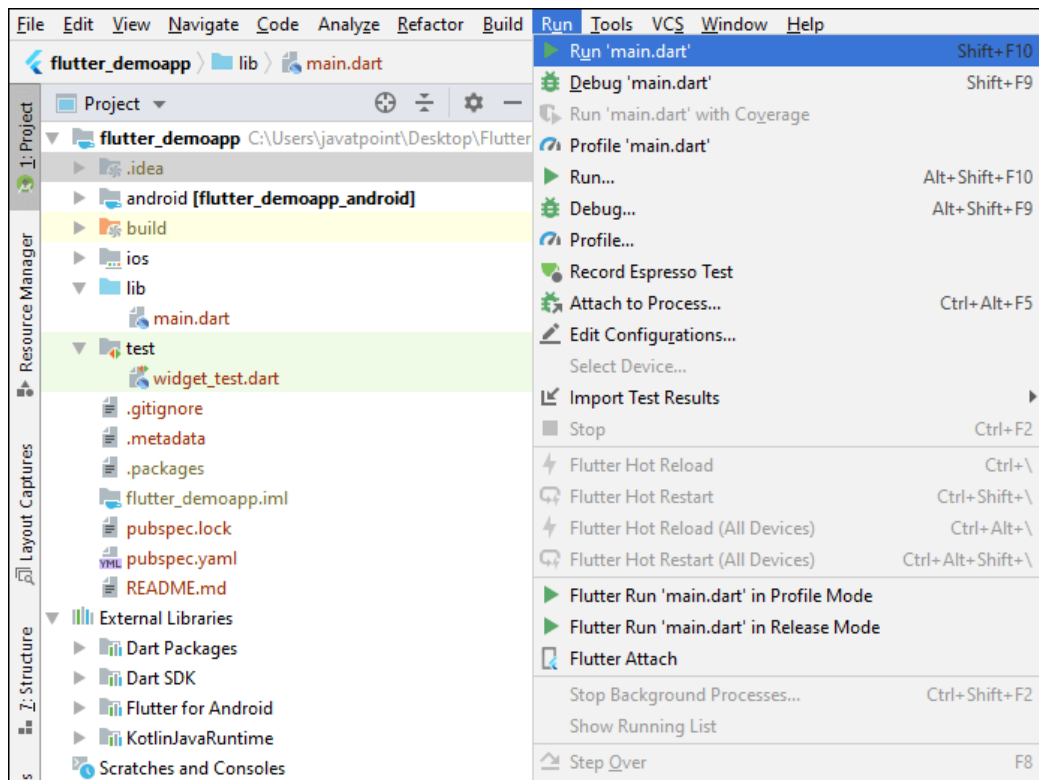
1. **import** 'package:flutter/material.dart';
- 2.
3. **void** main() => runApp(MyApp());
- 4.
5. **class** MyApp **extends** StatelessWidget {
6. // This widget is the root of your application.

```
7.  @override
8.  Widget build(BuildContext context) {
9.    return MaterialApp(
10.      title: 'Hello World Flutter Application',
11.      theme: ThemeData(
12.        // This is the theme of your application.
13.        primarySwatch: Colors.blue,
14.      ),
15.      home: MyHomePage(title: 'Home page'),
16.    );
17.  }
18. }
19. class MyHomePage extends StatelessWidget {
20.  MyHomePage({Key key, this.title}) : super(key: key);
21.  // This widget is the home page of your application.
22.  final String title;
23.
24.  @override
25.  Widget build(BuildContext context) {
26.    return Scaffold(
27.      appBar: AppBar(
28.        title: Text(this.title),
29.      ),
30.      body: Center(
31.        child: Text('Hello World'),
32.      ),
33.    );
34.  }
35. }
```

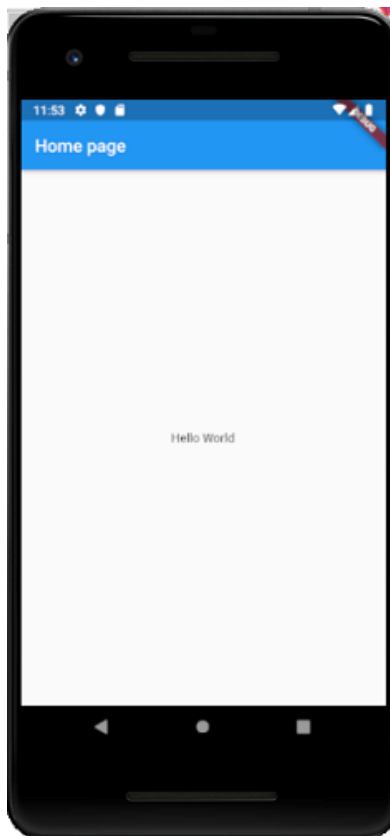

Step 8: Let us understand the above code snippet line by line.

- To start Flutter programming, you need first to import the Flutter package. Here, we have imported a **Material package**. This package allows you to create user interface according to the Material design guidelines specified by Android.
- The second line is an entry point of the Flutter applications similar to the main method in other programming languages. It calls the **runApp** function and pass it an object of **MyApp**. The primary purpose of this function is to attach the given widget to the screen.
- Line 5 to 18 is a widget used for creating UI in the Flutter framework. Here, the **StatelessWidget** does not maintain any state of the widget. **MyApp** extends **StatelessWidget** that overrides its **build**. The build method is used for creating a part of the UI of the application. In this block, the build method uses **MaterialApp**, a widget to create the root level UI of the application and contains three properties - title, theme, and home.
 - **Title:** It is the title of the Flutter application.
 - **Theme:** It is the theme of the widget. By default, it set the blue as the overall color of the application.
 - **Home:** It is the inner UI of the application, which sets another widget (**MyHomePage**) for the application.
- Line 19 to 35, the **MyHomePage** is similar to **MyApp**, except it will return the **Scaffold**. **Scaffold** widget is a top-level widget after the **MaterialApp** widget for creating the user interface. This widget contains two properties **appBar** and **body**. The **appBar** shows the header of the app, and **body** property shows the actual content of the application. Here, **AppBar** render the header of the application, **Center** widget is used to center the child widget, and **Text** is the final widget used to show the text content and displays in the center of the screen.

Step 9: Now, run the application. To do this, go to Run->Run main.dart, as shown in the below screen.



Step 10: Finally, you will get the output as below screen.



Exercise

- Change the text in the Text widget to something personal, e.g., "Hello, Flutter World!".
- Experiment with the **TextStyle** properties such as font size, color, and font weight.
- Add more widgets like **Icons** or a **Button** to explore further.
- Customize the text in the **Text widget** and change the background color of the app.
- Create a new Flutter project and experiment with different widgets like **Image** or **Button**.
- Explore the flutter doctor output and fix any warnings or errors.
- Submit a screenshot of the running app on the emulator or device.
- Submit the modified code for the app with changes to the text and background color.