



# Chapter 1

## Introduction to Flutter

*Instructor: Ketema K. (Asst/Professor)*

# Introduction

---

- In general, creating a **mobile application** is a very *complex and challenging* task. There are many frameworks **available**, which provide excellent features to *develop mobile applications*.
- For *developing mobile apps*, Android provides a *native framework* based on *Java and Kotlin language*, while iOS provides a **framework** based on **Objective-C/Swift** language.
- Thus, we need *two* different **languages** and **frameworks** to *develop applications* for both **OS**.

# Cont..

---

- Today, to overcome form this ***complexity***, there are several ***frameworks*** have introduced that ***support*** both ***OS*** along with ***desktop apps***.
- These types of the ***framework*** are known as ***cross-platform*** development tools.

# Cont.

---

- The *cross-platform* development framework has the ability to write ***one code*** and can deploy on the **various platform** (*Android, iOS, and Desktop*).
- It saves a lot of time and development ***efforts*** of ***developers***. There are **several tools available** for ***cross-platform development***, including web-based tools, such as Ionic from Drifty Co. in 2013, Phone gap from Adobe, *Xamarin* from *Microsoft*, and *React Native* from *Facebook*.

# Cont..

---

- Each of these ***frameworks*** has varying degrees of success in the *mobile industry*.
- In recent, a ***new framework*** has introduced in *the cross platform development family* named **Flutter** developed from **Google**.

# What is Flutter ?

---

- Flutter is a **UI** toolkit for building *fast, beautiful, natively compiled* applications for **mobile, web, and desktop** with *one programming language* and *single codebase*.
- **Flutter** is an open source framework developed by **Google** to create high quality, high performance mobile applications across mobile **operating systems** - *Android and iOS*.
- It provides a **simple, powerful, efficient** and **easy** to understand **SDK** to write **mobile** applications in **Google's own language, Dart**.

## Cont..

---

- With Flutter, we can write the *app's code once* and *deploy* it on multiple platforms(**Android, IOS, Desktop**).
- Due to its *flexibility* and *performance capabilities*, Flutter is the *top pick for modern app development* across various platforms.

# Cont..

---

- **Flutter apps** use **Dart programming** language for creating an app.
- The **dart programming** shares several same features as other programming languages, such as Kotlin and Swift, and can be trans-compiled into *JavaScript code*.
- Flutter is mainly optimized for 2D mobile apps that can run on both *Android and iOS platforms*.
- We can also use it to build *full-featured apps*, including *camera, storage, geolocation, network, third-party SDKs*, and more.



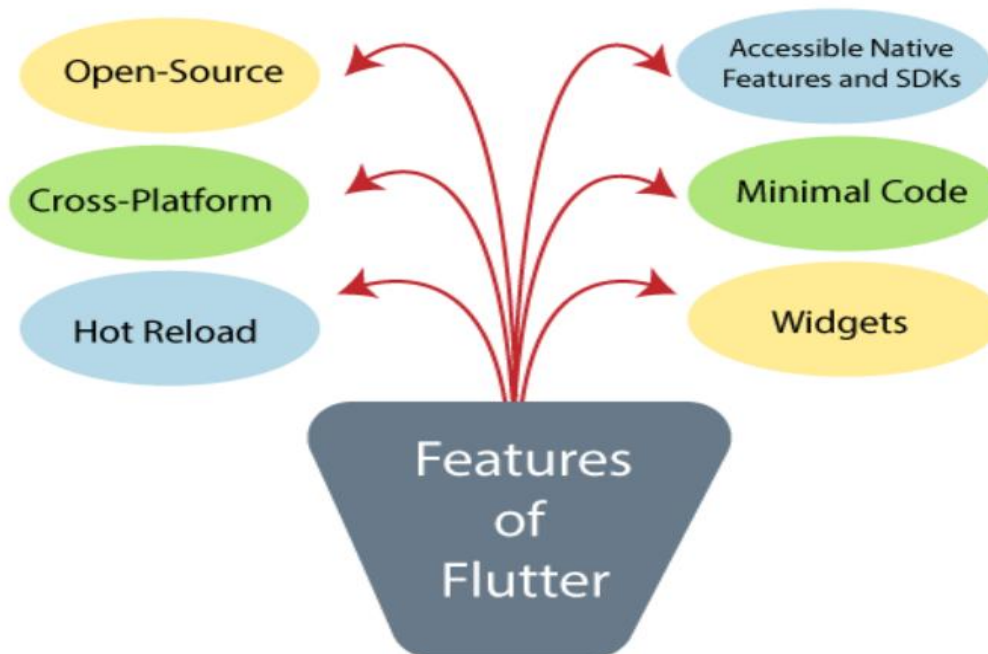
# What Makes Flutter Unique?

---

- Flutter is different from other **frameworks** because it neither uses **WebView** nor the **OEM**(*original equipment manufacturer*) widgets that **shipped** with the device.
- Instead, it uses its own **high-performance** rendering engine to draw **widgets**.
- It also implements most of its **systems** such as **animation**, **gesture**, and **widgets** in **Dart** programming language that allows developers to **read, change, replace, or remove** things easily.
- It gives **excellent** control to the **developers** over the system.

# Features

- Flutter gives easy and simple methods to start building beautiful mobile and desktop apps with a rich set of material design and widgets. Here, we are going to discuss its main features for developing the *mobile framework*.



# Cont..

---

- **Open-Source:** Flutter is a **free and open-source** framework for developing mobile applications.
- **Cross-platform:** This feature allows Flutter to write the code once, maintain, and can run on different platforms. It ***saves the time, effort, and money*** of the developers.
- **Hot Reload:** Whenever the developer makes changes in the ***code***, then these changes can be seen **instantaneously** with Hot Reload. It means the changes immediately visible in the ***app itself***. It is a very handy feature, which allows the developer to ***fix the bugs*** instantly.

# Cont..

- **Accessible Native Features and SDKs:** This feature allows the app development process easy and *wonderful* through Flutter's native code, third-party integration, and platform APIs. Thus, we can easily access the *SDKs on both platforms*.
- **Minimal code:** Flutter app is developed by **Dart programming language**, which uses *Just-in-Time* (JIT) and *Ahead-of-Time* (AOT) compilation to improve the overall **start-up** time, functioning and accelerates the performance. JIT enhances the development system and refreshes the UI without putting extra effort into **building** a new one.

# Cont..

---

- **Widgets:** The Flutter framework offers widgets, which are capable of developing customizable specific designs. Most importantly, Flutter has *two sets of widgets*: ***Material Design*** and **Cupertino widgets** that help to provide a glitch-free experience on all platforms.

# Advantage

---

- It makes the *app development* process extremely fast because of the *hot-reload* feature. This feature allows us to *change or update* the code are reflected as soon as the *alterations* are made.
- It provides the smoother and seamless scrolling experiences of using the app without much hangs or cuts, which makes running applications faster in comparison to other mobile app development frameworks.

# Advantage

---

- Flutter reduces the ***time and efforts*** of testing. As we know, flutter apps are cross-platform so that testers do not always need to run the same set of tests on different platforms for the ***same app***.
- It has an excellent user interface because it uses a **design-centric widget, high-development tools, advanced APIs**, and many more features.

# Advantage

---

- It is similar to a **reactive framework** where the developers do not need to update the UI content manually.
- It is suitable for MVP (Minimum Viable Product) apps because of its speedy development *process and cross-platform* nature.



# Disadvantages

---

- The ***Flutter*** is a comparatively new language that needs continuous integration support through the ***maintenance*** of scripts.
- It provides very limited access to SDK libraries. It means a developer does not have a lot of ***functionalities*** to create a mobile application. Such types of ***functionalities*** need to be developed by the ***Flutter developer*** themselves.

# Disadvantages

---

- The Flutter apps do not support the browser. It only supports Android and iOS platforms.
- It uses Dart programming for coding, so a developer needs to learn new technologies. However, it is easy to learn for developers.

# History

---

- Flutter is a **free** and **open-source** *UI software development kit* introduced by **Google**. It is used to build applications for Android, iOS, Windows, and the web.
- The **first version** of Flutter was **announced** in the year **2015** at the Dart Developer Summit. It was initially known as codename "**Sky**" and can **run** on the ***Android OS***.
- After the announcement of **Flutter**, the first ***Flutter Alpha version*** (v-0.06) was released in **May 2017**.

# History

---

- Later, during the keynote of Google Developer days in Shanghai, **Google launched** the second preview of Flutter in **September 2018** that was the **last big** release before Flutter 1.0 version.
- On December 4, 2018, the first stable version of the *Flutter framework* was released at the **Flutter Live event**, denoting Flutter 1.0. The current stable release of the *framework* is Flutter v1.9.1+hotfix.6 on October 24, 2019.

# Flutter Installation

- In this section, we are going to learn how to set up an environment for the ***successful*** development of the ***Flutter application***.

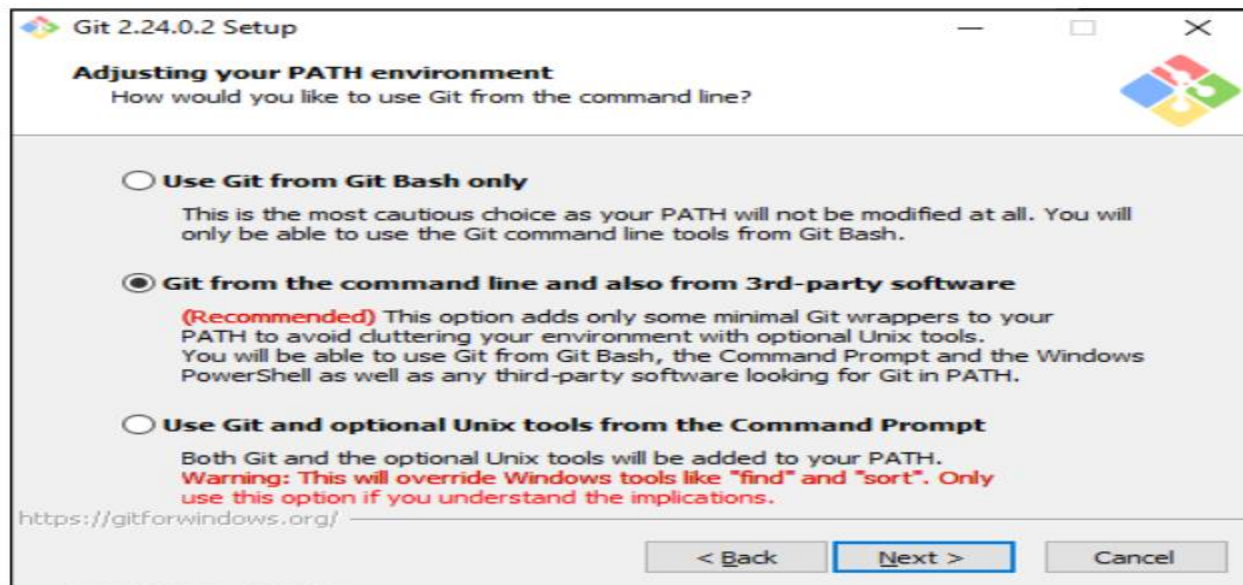
## Installation in Windows

- To install and run Flutter on the **Windows system**, you need first to meet these ***requirements*** for your development ***environment***.

Operating System	Windows 7 or Later (I am Windows 10. You can also use Mac or Linux OS.).
Disk Space	400 MB (It does not include disk space for IDE/tools).
Tools	1. Windows PowerShell 2. Git for Windows 2.x (Here, Use Git from Windows Command Prompt option).
SDK	Flutter SDK for Windows
IDE	Android Studio (Official)

# Install Git

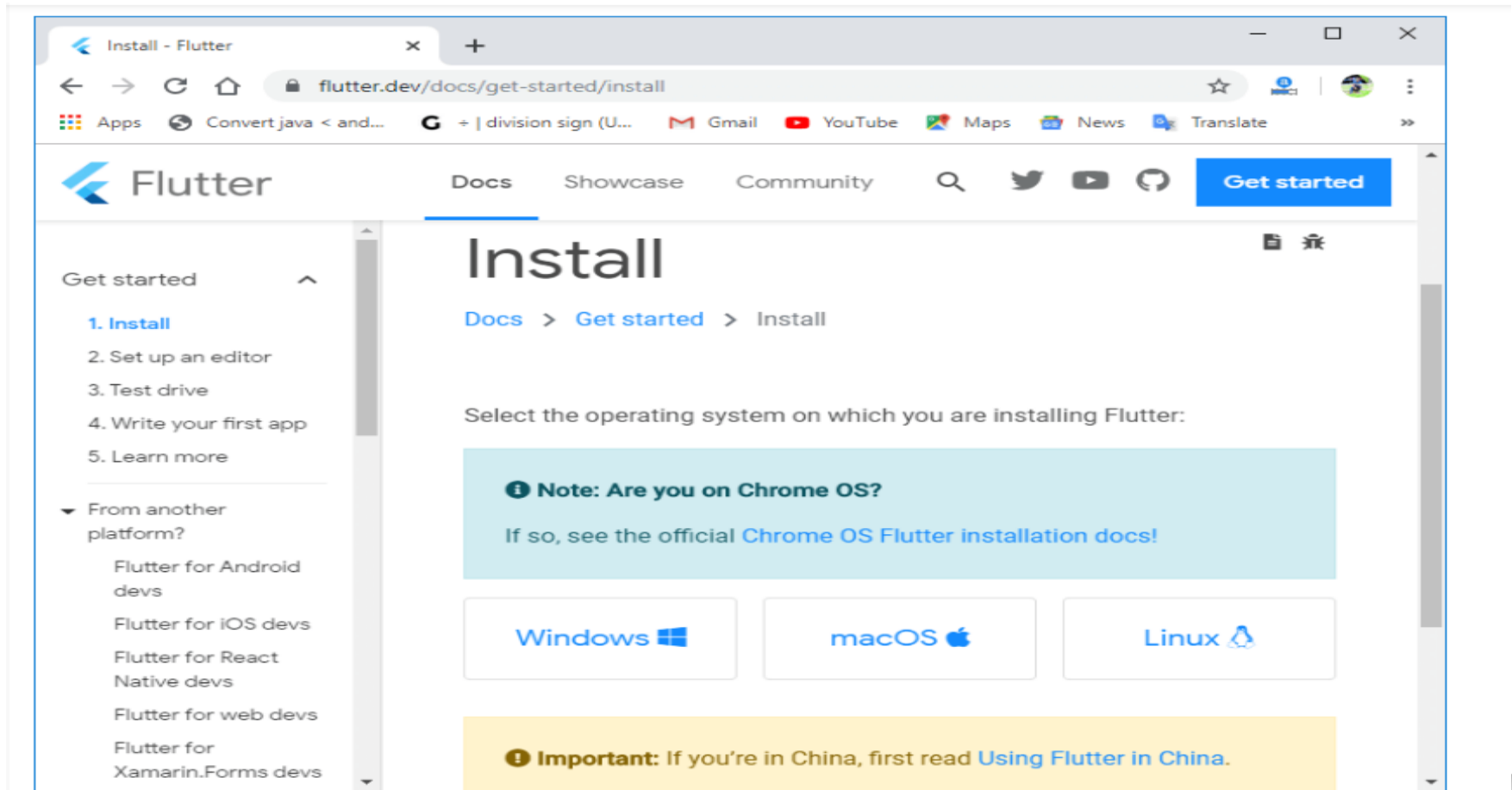
- **Step 1:** To download Git, [click here](#).
- **Step 2:** Run the **.exe** file to complete the installation. During installation, make sure that you have selected the recommended option.



To read more information about installing Git, [click here](#).

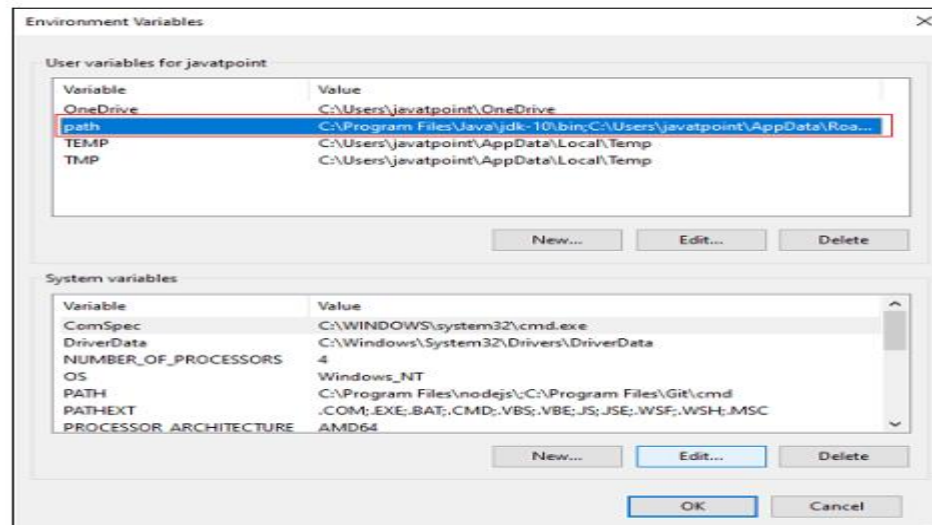
# Install the Flutter SDK

- **Step 1:** Download the *installation* bundle of the *Flutter Software Development Kit for windows*. To download Flutter SDK, Go to its official [website](https://flutter.dev/docs/get-started/install), click on **Get started** button, you will get the following screen.



# Cont..

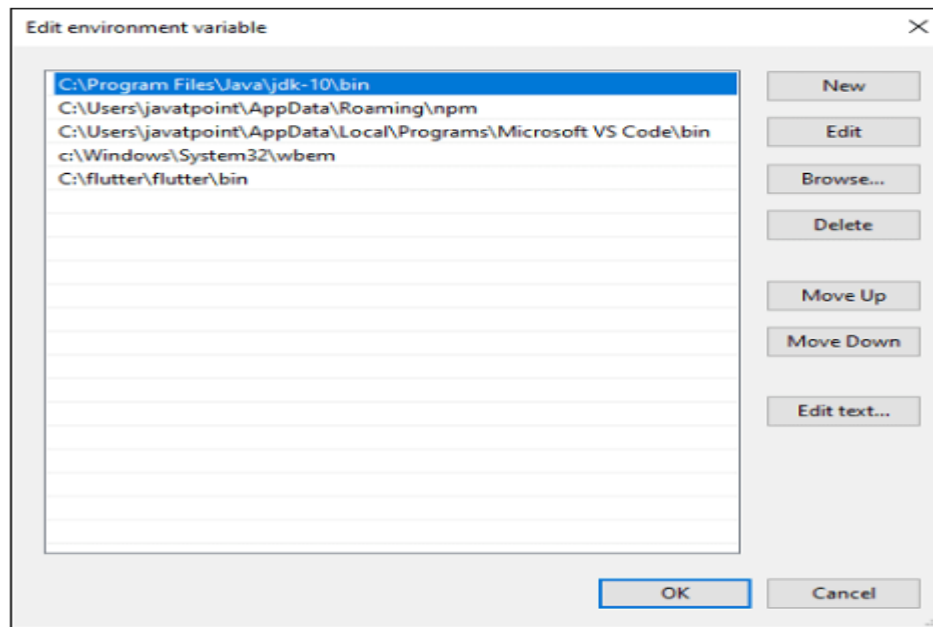
- **Step 2:** Next, to download the latest Flutter SDK, click on the **Windows icon**. Here, you will find the download link for **SDK**.
- **Step 3:** When your download is complete, extract the **zip** file and place it in the desired installation folder or location, for example, *D: /Flutter*.
- ❖ Note: The Flutter SDK should not be placed where the administrator's permission is required.
- **Step 4:** To **run** the **Flutter command** in regular **windows console**, you need to update the system path to include the **flutter bin directory**. The following steps are required to do this:
- Step 4.1: Go to MyComputer properties -> advanced tab -> environment variables. You will get the following screen.





# Cont..

- **Step 4.2:** Now, select path -> click on edit. The following screen appears.



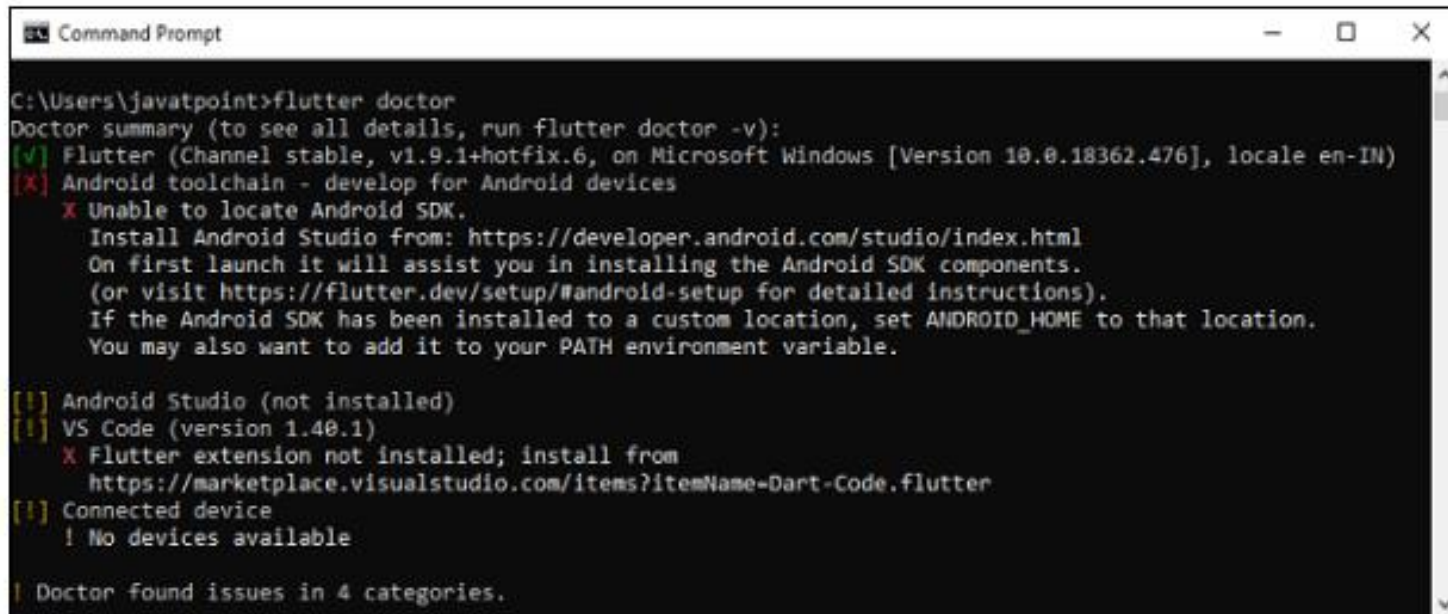
**Step 4.3:** In the above window, click on New->write path of *Flutter bin* folder in variable value -> ok -> ok -> ok.

**Step 5:** Now, run the \$ **flutter doctor** command. This *command checks for all the requirements of Flutter app development* and displays a *report* of the status of your Flutter installation.

# Cont..

```
$ flutter doctor
```

- Step 6: When *you run the above command*, it will analyze the system and show its report, as shown in the below image. Here, you will find the details of all missing tools, which required to run Flutter as well as the development tools that are available but not connected with the device.



```
Command Prompt
C:\Users\javatpoint>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, v1.9.1+hotfix.6, on Microsoft Windows [Version 10.0.18362.476], locale en-IN)
[X] Android toolchain - develop for Android devices
    X Unable to locate Android SDK.
      Install Android Studio from: https://developer.android.com/studio/index.html
      On first launch it will assist you in installing the Android SDK components.
      (or visit https://flutter.dev/setup/#android-setup for detailed instructions).
      If the Android SDK has been installed to a custom location, set ANDROID_HOME to that location.
      You may also want to add it to your PATH environment variable.

[!] Android Studio (not installed)
[!] VS Code (version 1.40.1)
    X Flutter extension not installed; install from
      https://marketplace.visualstudio.com/items?itemName=Dart-Code.flutter
[!] Connected device
    ! No devices available

! Doctor found issues in 4 categories.
```

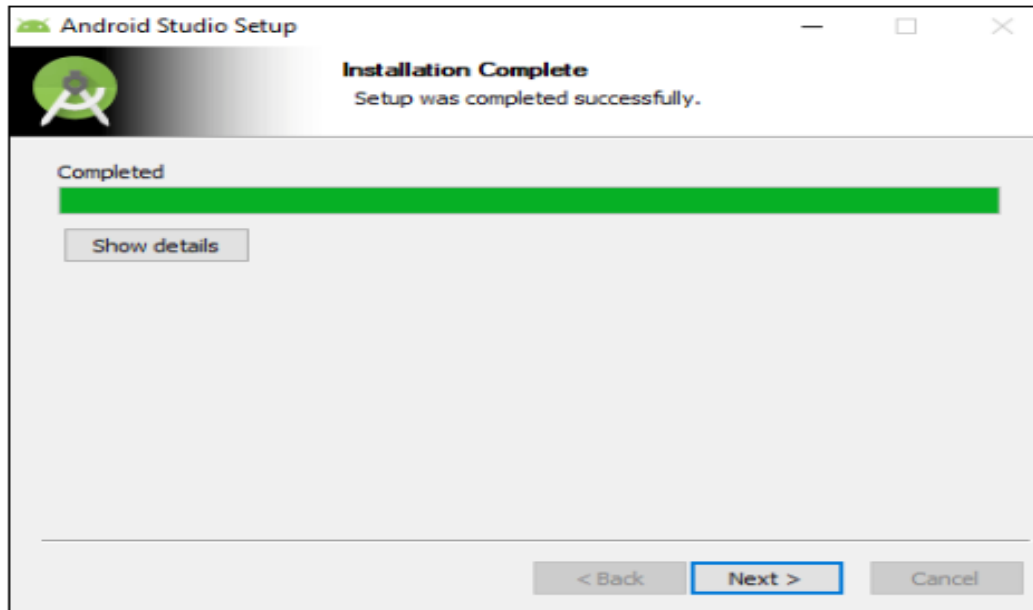
# Cont..

- **Step 7: Install the Android SDK.** If the flutter doctor command does not find the Android SDK tool in your system, then you need first to install the Android Studio IDE. To install Android Studio IDE, do the following steps.
- **Step 7.1:** Download the latest Android Studio executable or zip file from the [official site](#).
- **Step 7.2:** When the download is complete, open the **.exe** file and run it. You will get the following dialog box.



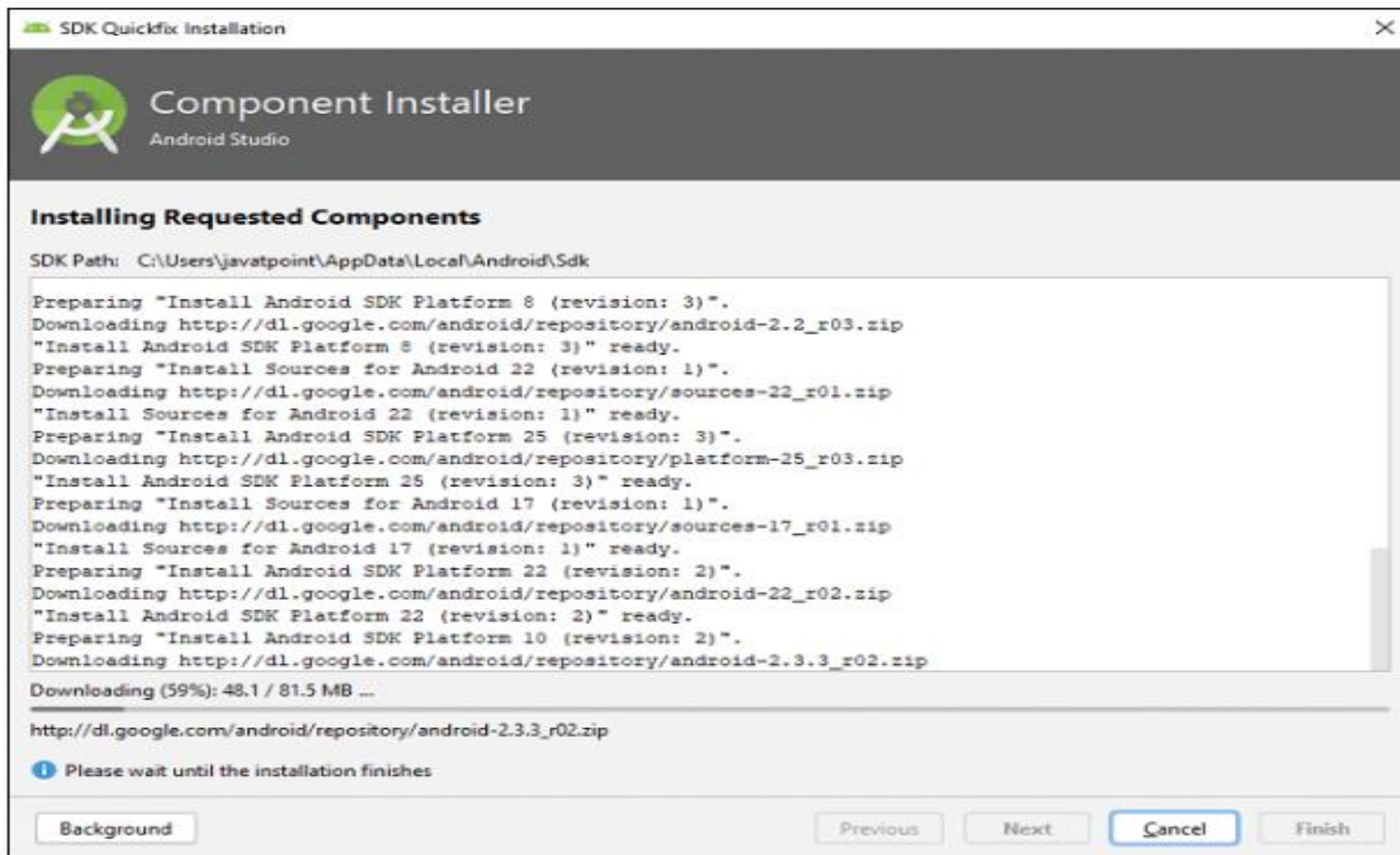
# Cont..

- **Step 7.3:** Follow the steps of the installation wizard. Once the installation wizard completes, you will get the following screen.



- **Step 7.4:** In the above screen, **click Next-> Finish**. Once the Finish button is clicked, you need to choose the 'Don't import Settings option' and click OK. It will start the Android Studio.
- **Note:>** Meanwhile, the installation wizard also includes downloading Android SDK components that are required by Flutter for development.

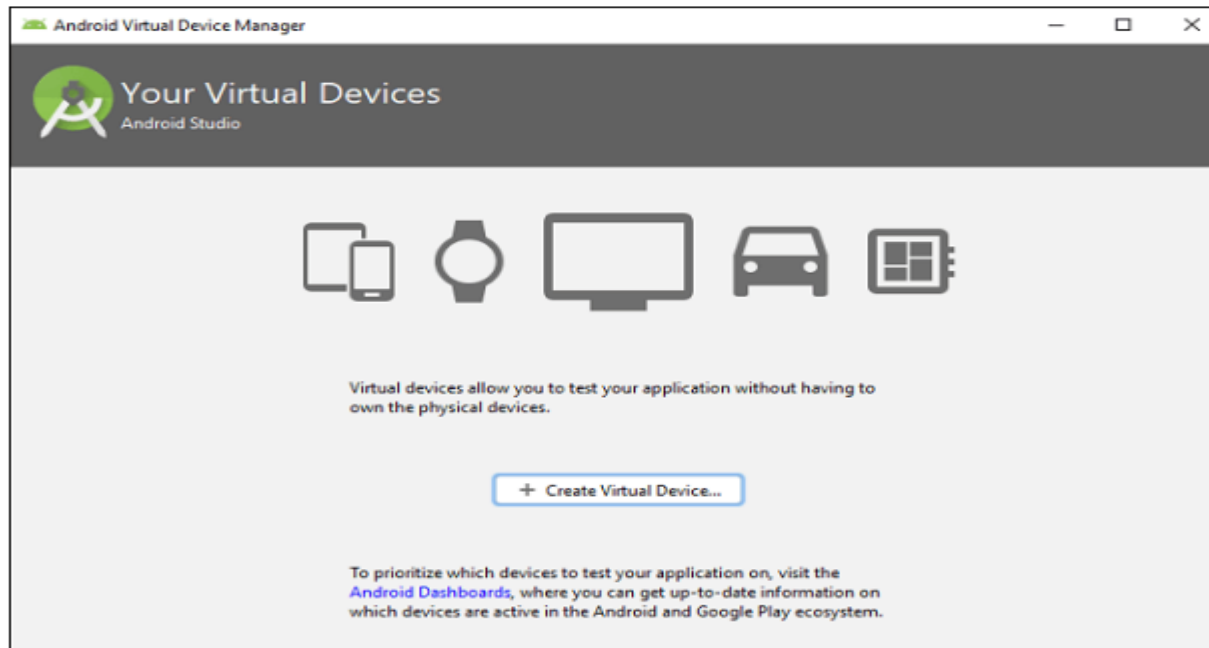
# Cont..



❖ **Step 8:** Next, you need to set up an **Android emulator**. It is responsible for running and testing the **Flutter application**.

# Cont..

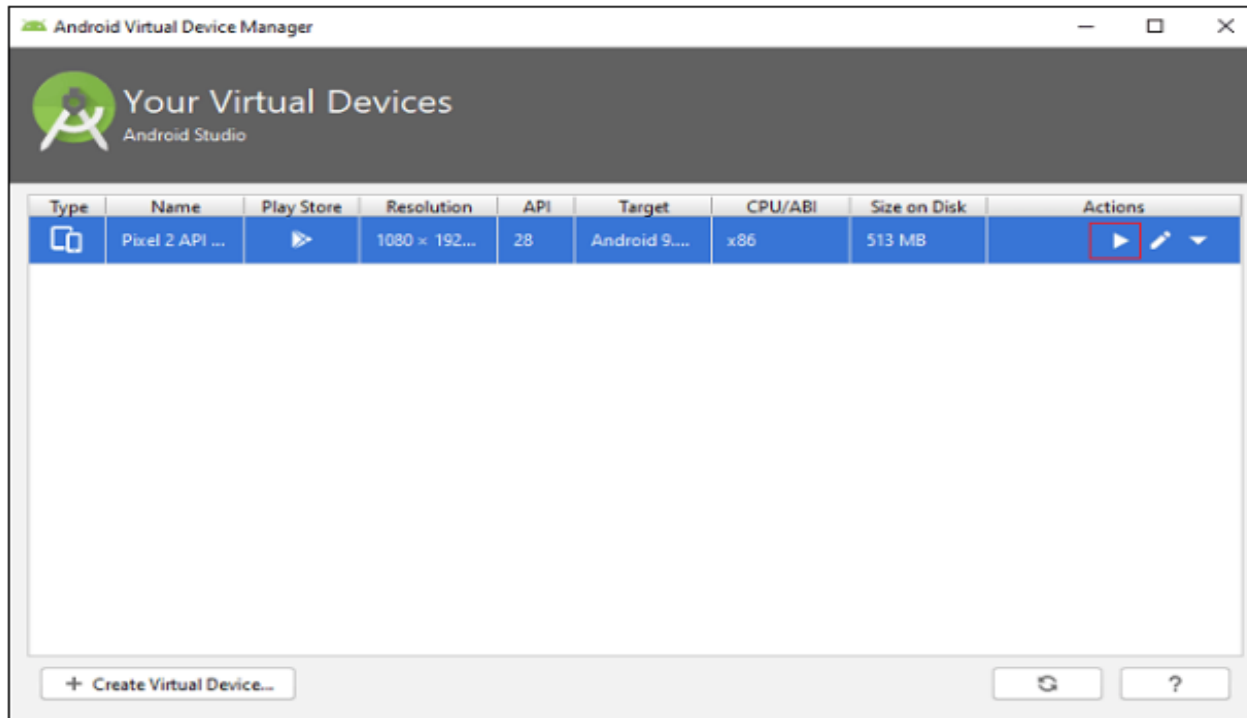
- **Step 8.1:** To set an *Android emulator*, go to Android Studio > Tools > Android > AVD Manager and select Create **Virtual Device**. Or, go to Help->Find Action->Type Emulator in the **search box**. *You will get* the following screen.



**Step 8.2:** Choose your **device** definition and click on Next.

# Cont..

- **Step 8.3:** Select the system image for the latest Android version and click on Next.
- **Step 8.4:** Now, verify the all AVD configuration. If it is correct, click on Finish. The following screen appears.



# Cont..

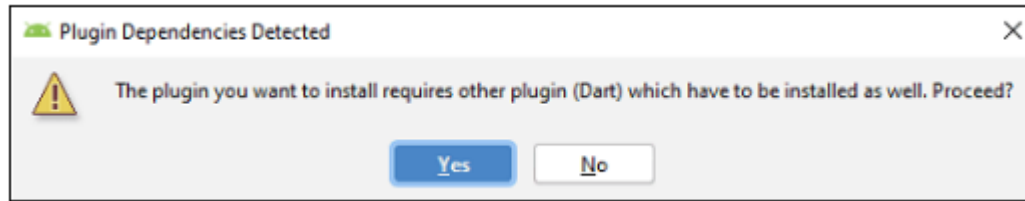
- **Step 8.5:** Last, click on the icon pointed into the **red color rectangle**. The Android emulator displayed as below screen.





# Cont..

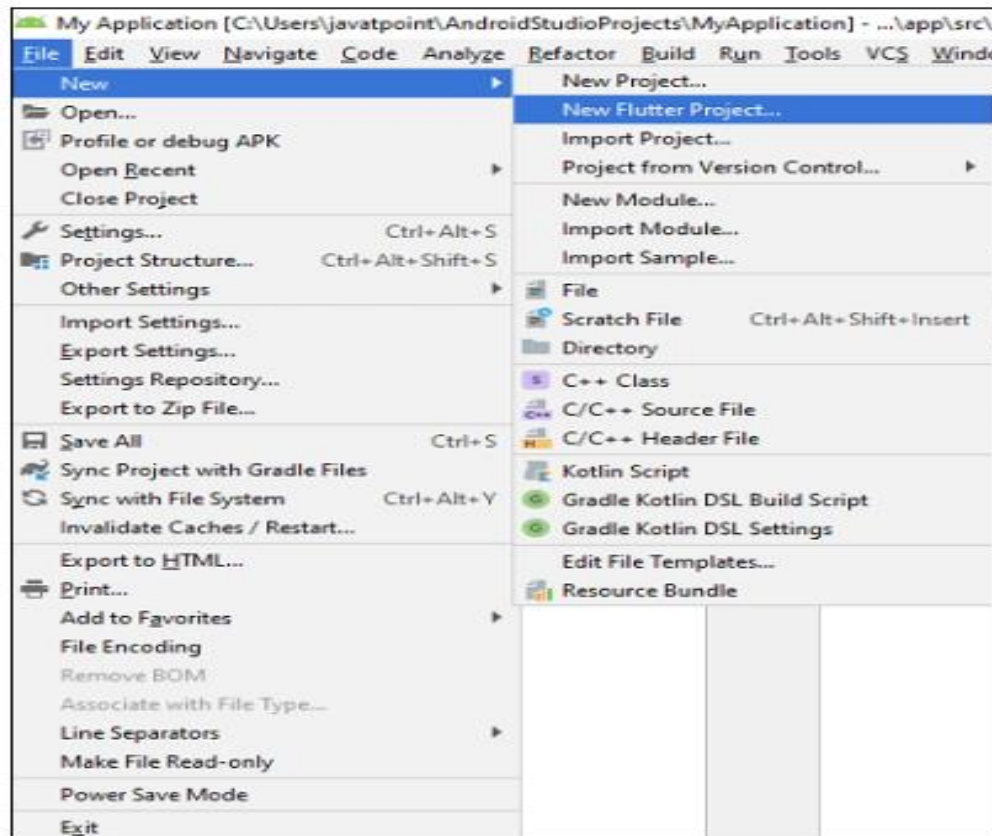
- **Step 9.1:** Open the Android Studio and then go to *File->Settings->Plugins*.
- **Step 9.2:** Now, search the **Flutter plugin**. If found, select Flutter plugin and click install. When you click on install, it will ask you to install Dart plugin as below screen. Click yes to proceed.



- **Step 9.3:** Restart the **Android Studio**

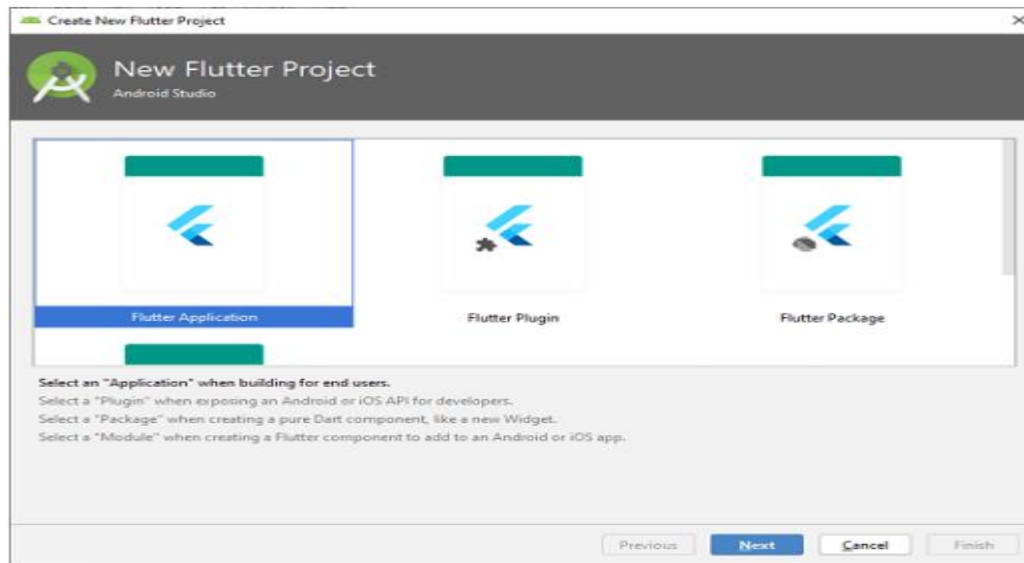
# Flutter First Application

- 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.



# Cont..

- **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.

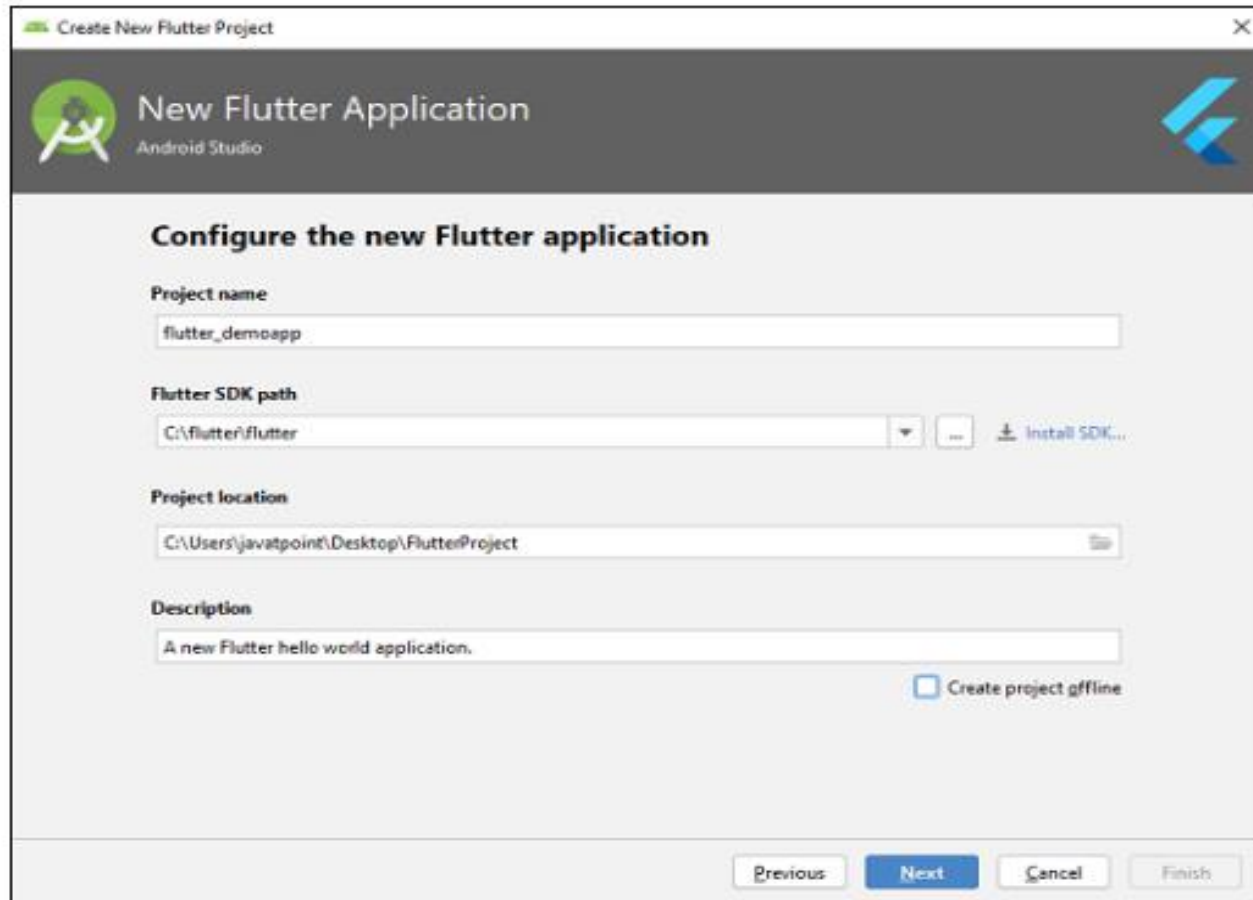
**Project Name:** Write your Application Name.

**Flutter SDK Path:** <path\_to\_flutter\_sdk>

**Project Location:** <path\_to\_project\_folder>

**Descriptions:** <A new Flutter hello world application>.

# Cont..



The screenshot shows the 'Create New Flutter Project' dialog box in Android Studio. The dialog has a title bar with a close button. The main header area contains the Android Studio logo, the text 'New Flutter Application', and the Flutter logo. The main content area is titled 'Configure the new Flutter application' and contains four sections: 'Project name' with a text field containing 'flutter\_demoapp'; 'Flutter SDK path' with a text field containing 'C:\flutter\flutter', a dropdown arrow, a browse button, and a link to 'Install SDK...'; 'Project location' with a text field containing 'C:\Users\javatpoint\Desktop\FlutterProject' and a browse button; and 'Description' with a text field containing 'A new Flutter hello world application.' and a checkbox labeled 'Create project gffline'. At the bottom, there are four buttons: 'Previous', 'Next', 'Cancel', and 'Finish'.

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...

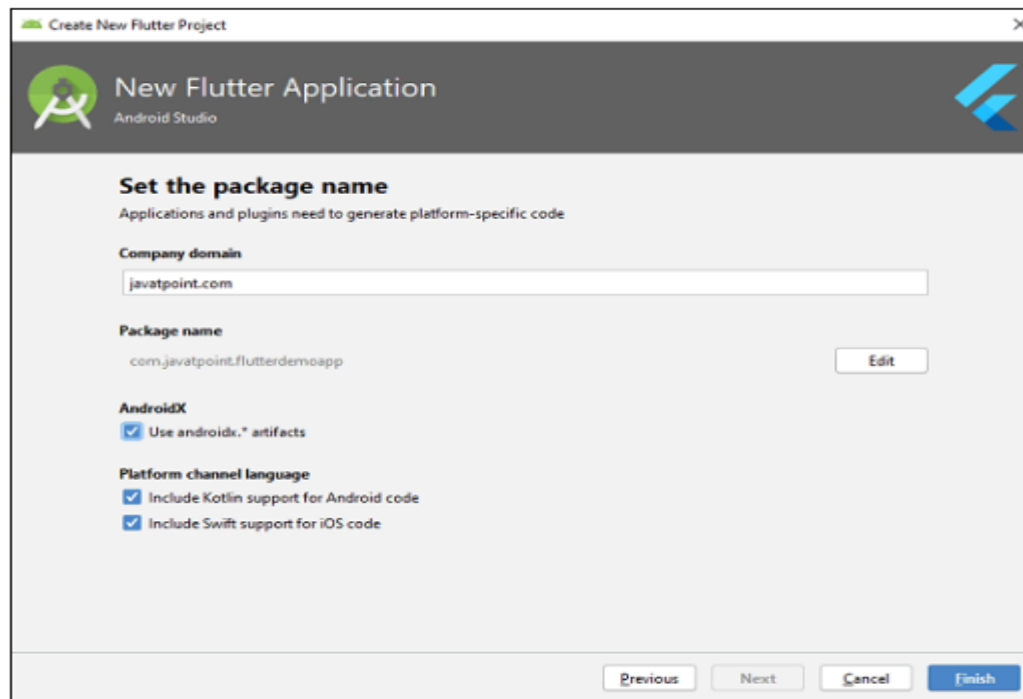
Project location  
C:\Users\javatpoint\Desktop\FlutterProject

Description  
A new Flutter hello world application.  
☐ Create project gffline

Previous Next Cancel Finish

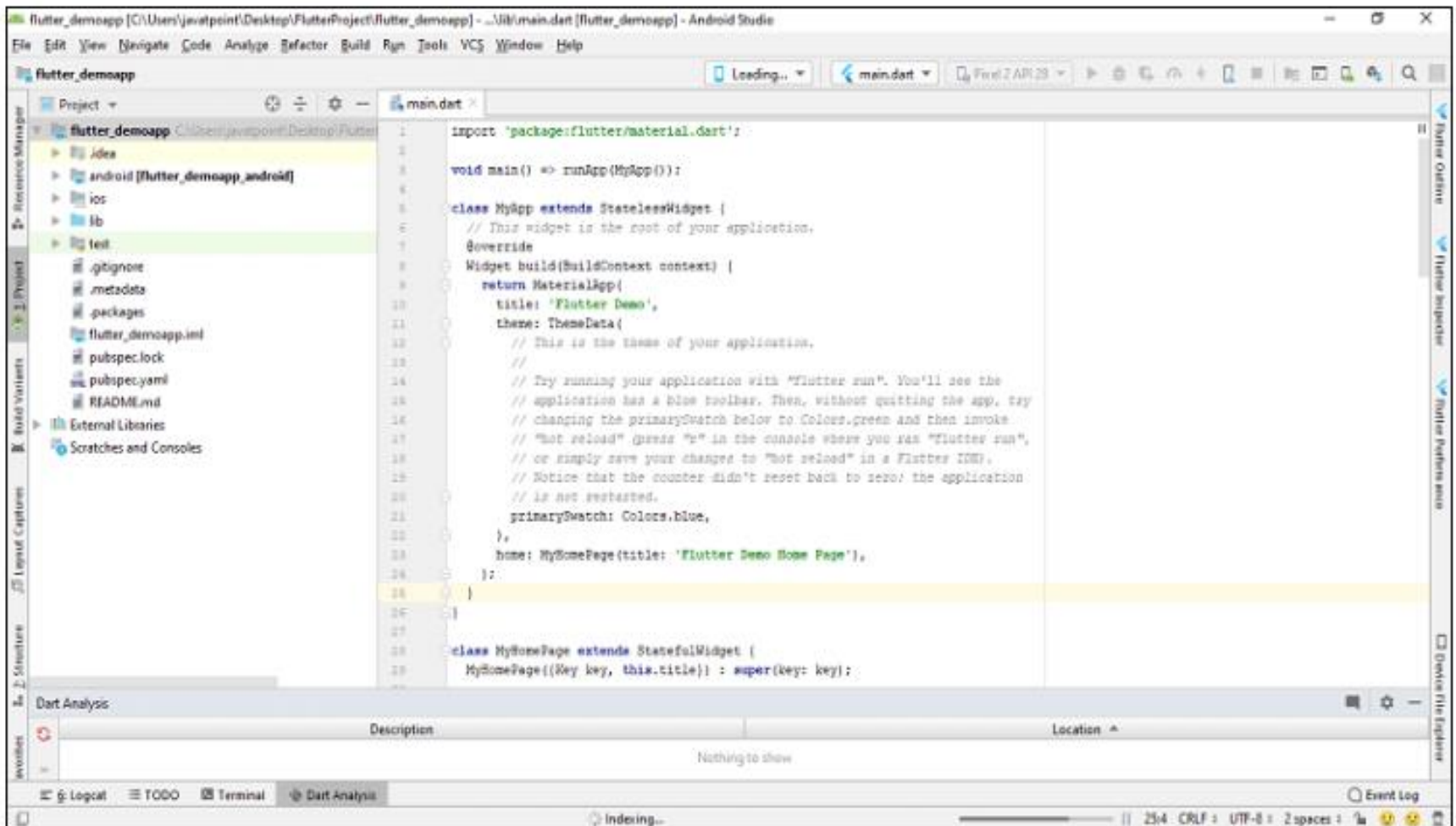
**Step 5:** In the next wizard, you need to set the company domain name and click the Finish button.

# Cont..



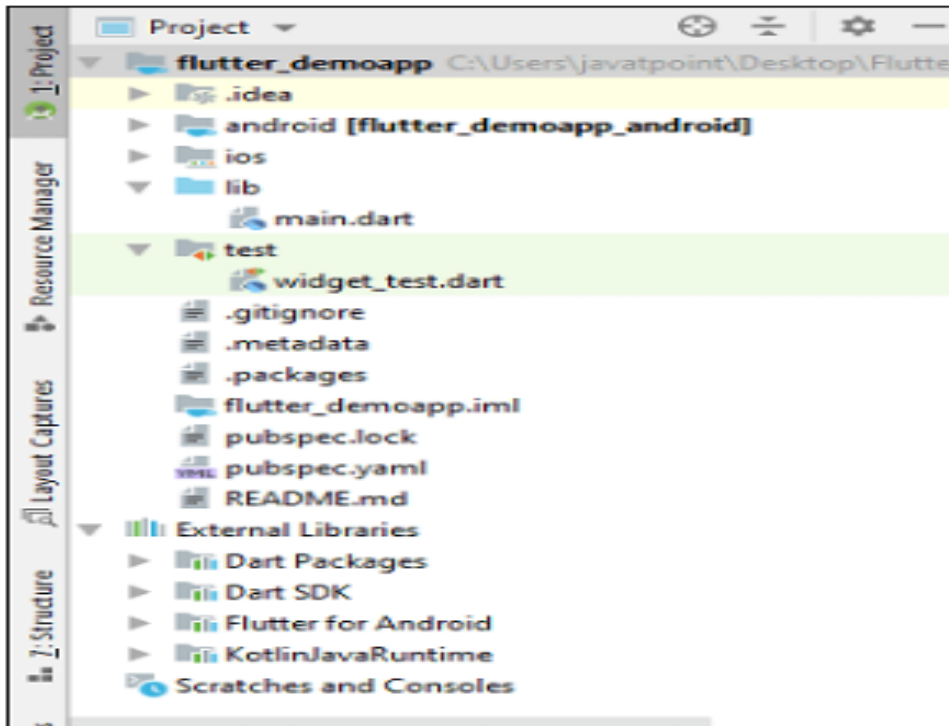
After clicking the Finish button, it will take some time to create a project. When the project is created, you will get a fully working Flutter application with minimal functionality.

# Cont..



# Cont..

- **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.



## Various components of the structure of the application are explained 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**.



# Cont..

---

- **.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**.

# Cont..

---

- **.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.

# Cont..

---

- 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:
  - **.git ignore:** 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.

# Cont..

---

- **.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.

# Cont..

---

- **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.

# Summary of Various components of the structure of the application

---

- ❖ **android** - Auto generated source code to create android application
- ❖ **ios** - Auto generated source code to create ios application
- ❖ **lib** - Main folder containing Dart code written using flutter framework
- ❖ **lib/main.dart** - Entry point of the Flutter application
- ❖ **test** - Folder containing Dart code to test the flutter application
  - ❖ **test/widget\_test.dart** - Sample code
  - ❖ **.gitignore** - Git version control file
  - ❖ **.metadata** - auto generated by the flutter tools
  - ❖ **.packages** - auto generated to track the flutter packages
  - ❖ **.iml** - project file used by Android studio
  - ❖ **pubspec.yaml** - Used by Pub, Flutter package manager
  - ❖ **pubspec.lock** - Auto generated by the Flutter package manager, Pub
  - ❖ **README.md** - Project description file written in Markdown format

# cont..

```
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. }
```

# Cont..

- **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.



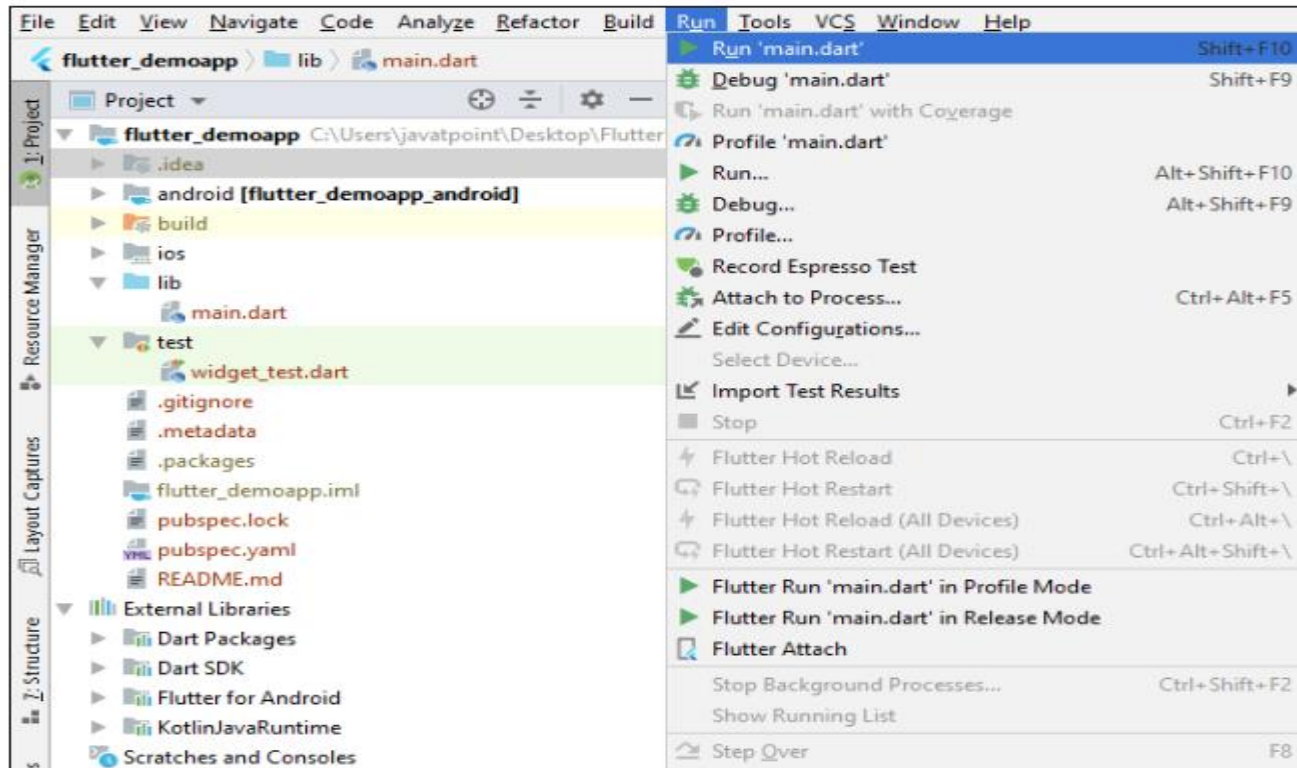
# Cont..

---

- **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.

# Cont..

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



# Cont..

---

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



# Flutter-architecture

---

- In this section, we are going to discuss the architecture of the Flutter framework. The Flutter architecture mainly comprises of ***four components***.
  1. Flutter Engine
  2. Foundation Library
  3. Widgets
  4. Design Specific Widgets

## Flutter Engine

- It is a portable runtime for high-quality mobile apps and primarily based on the C++ language. It implements Flutter core libraries that include animation and graphics, file and network I/O, plugin architecture, accessibility support, and a dart runtime for developing, compiling, and running Flutter applications. It takes Google's open-source graphics library, Skia, to render low-level graphics.

## Foundation Library

- It contains all the required packages for the basic building blocks of writing a Flutter application. These libraries are written in Dart language.

# Cont..

---

## Widgets

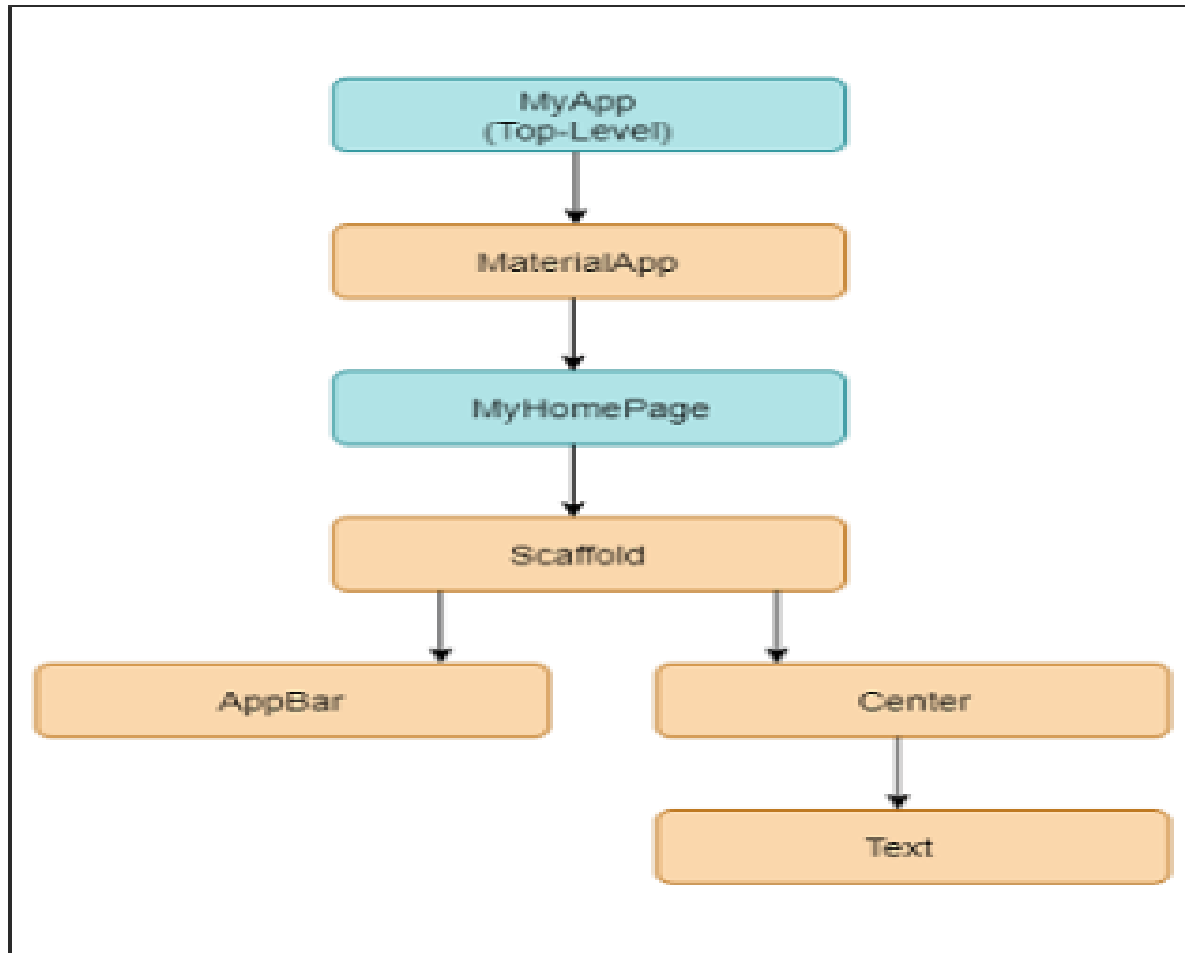
- In Flutter, everything is a **widget**, which is the core concept of this framework.
- Widget in the Flutter is basically a user interface component that affects and controls the view and interface of the app.
- It represents an immutable description of part of the user interface and includes graphics, text, shapes, and animations that are created using widgets.
- The widgets are similar to the React components.

# Cont..

---

- In Flutter, the application is itself a widget that contains many **sub widgets**.
- It means the **app** is the **top-level widget**, and its **UI** is build using one or more children widgets, which again includes sub child widgets.
- This feature helps you to create a complex user interface very easily.
- We can understand it from the *hello world* example created in the previous section. Here, we are going to explain the example with the following diagram.

# Cont..



- ❖ In the above example, we can see that all the components are widgets that contain child widgets. Thus, the Flutter application is itself a widget.

# Cont...

---

## Design Specific Widgets

- The *Flutter framework* has two sets of widgets that conform to specific design languages.
- These are **Material Design** for **Android** application and **Cupertino** Style for **IOS** application.

## Gestures

- It is a widget that provides interaction (how to listen for and respond to) in Flutter using GestureDetector. **GestureDector** is an invisible widget, which includes *tapping, dragging, and scaling interaction* of its child widget. We can also use other *interactive features* into the existing widgets by composing with the *GestureDetector* widget.



# State Management

---

- **Flutter widget** maintains its state by using a special widget, **Stateful Widget**. It is always auto re-rendered whenever its internal state is changed. The re-rendering is optimized by calculating the distance between old and new widget UI and render only necessary things that are changes.

■

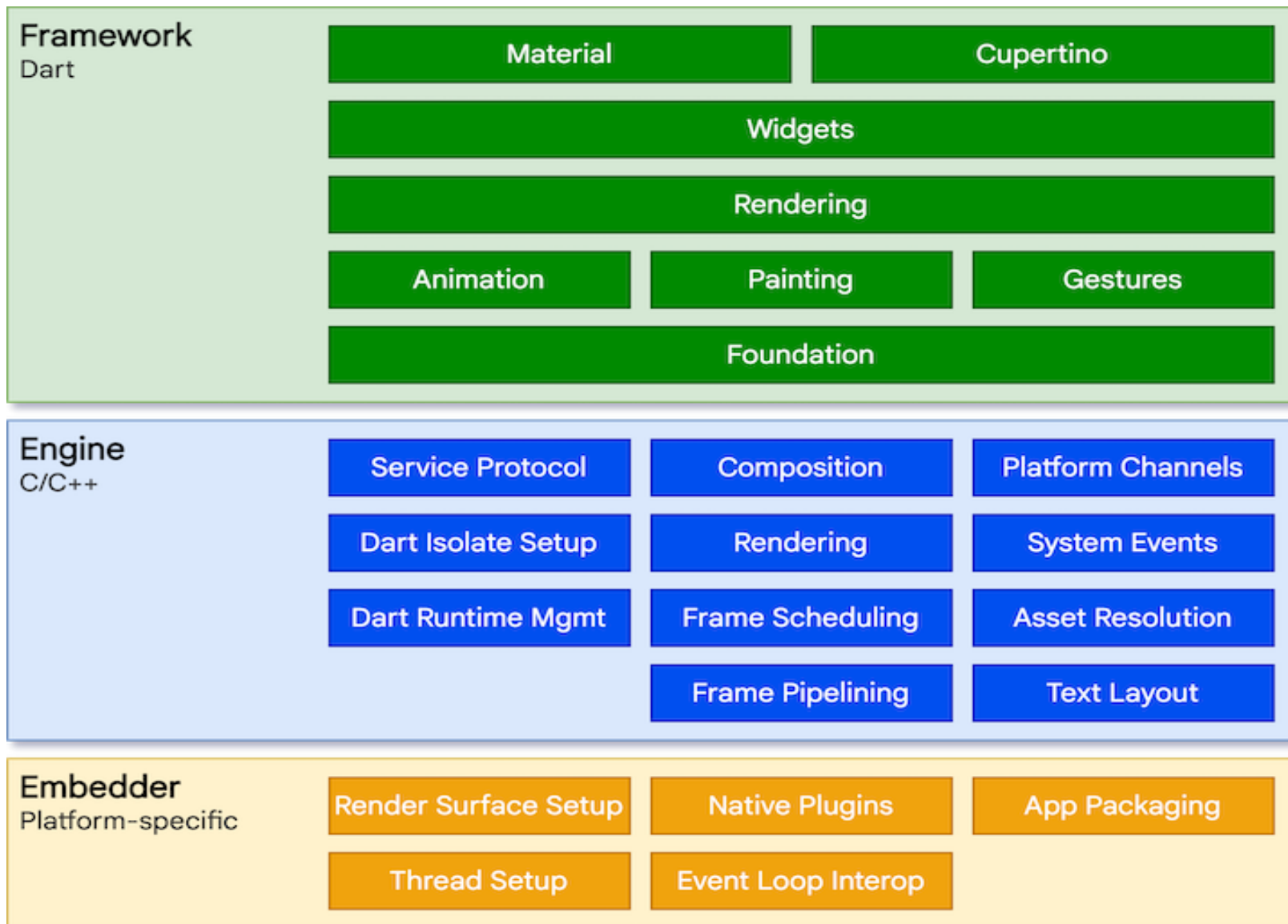
# Layers



Flutter is packaged with three layers:

- Embedder (lowest layer)
- Engine
- Framework (highest layer)

# Flutter System Overview





*THANK YOU*

