Unit-1

Flutter-Introduction

In general, developing a mobile application is a complex and challenging task. There are many frameworks available to develop a mobile application. Android provides a native framework based onJava language and iOS provides a native framework based on Objective-C / Shift language.

However, to develop an application supporting both the OS, we need to code in two different languages using two different frameworks. To help overcome this complexity, there exists mobile frameworks support in both OS. These frameworks range from simple HTML based hybrid mobile application framework (which uses HTML for User Interface and JavaScript for application logic) to complex language specific framework (which do the heavy lifting of converting code to native code). Irrespective of their simplicity or complexity, these frameworks always have many disadvantages, oneof the main drawback being their slow performance.

In this scenario, Flutter – a simple and high performance framework based on Dart language, provides high performance by rendering the UI directly in the operating system's canvas rather than through native framework.

Flutter also offers many ready to use widgets (UI) to create a modern application. These widgets are optimized for mobile environment and designing the application using widgets is as simple as designing HTML.

To be specific, Flutter application is itself a widget. Flutter widgets also supports animations and gestures. The application logic is based on reactive programming.

Widget may optionally have a state. By changing the state of the widget, Flutter will automatically (reactive programming)compare the widget's state(old and new) and render the widget with only the necessary changes instead of re-rendering the whole widget.

Features of Flutter

Flutter framework offers the following features to developers:

- ✓ Modern and reactive framework.
- ✓ Uses Dart programming language and it is very easy to learn.
- ✓ Fast development.
- ✓ Beautiful and fluid user interfaces.
- ✓ Huge widget catalog
- ✓ Runs same UI for multiple platforms

- ✓ High performance application

Advantages of Flutter:

- Single codebase for all platforms: There is no need to create separate code bases when working on iOS and Android devices. Flutter allows developers to builda single codebase and use it for several platforms such as the web, desktop and mobile. This results in quicker app launch and is cost effective.
- Reduced Development Time: The requirements for Flutter application development are much lower. So the positive outcome is that there are no additional maintenance charges. Flutter makes it possible to create larger apps that use unique features.
- Increased Time-to-Market Speed: The Flutter development framework is more responsive than others. This tool's quick time-to-market is one of its key advantages.It is mostly difficult to predict if iOS or Android versions will be able to provide a product on time. This is where Flutter plays its role, as it allows cross-platform usage.
- Native-like Performance: This is one of the Flutter advantages that stands out the most. Flutter works with Skia, a graphics engine which enables quick and well optimized development. It also is indistinguishable from native apps as it doesn't relyon interpreters or intermediary code representations.
- Powerful community: According to statista, Flutter has become one of the most popular frameworks and a first choice by developers globally. Over 40 percent of software developers have chosen Flutter over the course of the last three years. Thefollowing chart shows the growing interest in Flutter in comparison with other cross- platform app tools.
- Own Rendering Engine: The problem with some cross-platform solutions is thatthey are very similar on iOS and Android. This is why Flutter is the best option, because it consists of packages that contain a set of unique widgets for both operating systems.
- Hot Reload Feature: The ability to hot reload is one of the main benefits of using Flutter. This is for effective cross-platform development so it can complement the nature of Flutter. This feature's function speeds up application development.
- Flutter's Safety. Null-safety is supported on the syntax level. Flutter's UI code issingle-threaded, and computational threads are executed in isolated sandboxes which means no shared or unsafe resources at all.
- Excellent documentation and community support: Flutter benefits from extensive documentation and a vibrant community. Developers can find resources, tutorials, and support easily, making it easier to learn and troubleshoot any issues they may encounter.

OR

Flutter Disadvantages

- Large and Weighty Apps: A loophole that can’t be dismissed isthe size of the large size of the applications under development.Software developers working with this toolkit may find it difficult to work with large files. This can cause them to choose a lighter alternative.
- Dart’s low popularity: It’s a fact that Dart is a reliable programming language since it is fast. It is also true that developers are starting to make it an option. Yet, Dart is still notable to compete with other top programming languages such asjava, Kotlin, etc.
- Problems with iOS: The Flutter framework works well on both Android and iOS. However, Flutter was developed by Google, which gives Android apps a key advantage. It isn’t considered amajor issue, but it is something to consider in case some problems may be present in the future.
- Limited number of third-party libraries: Flutter being relativelycannot be compared to native programming languages. Developers still need to spend more time building as many libraries as possible.

Flutter SDK:

open- source software development kit (SDK) for cross-platform mobile application development. Using a single platform-agnostic codebase, Flutter helps developers build high-performance, scalable applications with attractiveand functional user interfaces for Android or IOS

Running a development team for each mobile platform sucks up resources from other work. Flutter is the most popular way for one development team to build on all platforms.

Life revolves around our mobile devices and applications. Worldwide, there are currently more than six billion smartphone subscriptions. In addition, between the Google Play Store and the Apple App Store, there are nearly five million mobile applications available for download. Although the mobile application market is increasingly competitive, this is where many startups and developers focus their efforts.

So it is only natural that mobile app developers are always looking for simpler, faster, and cheaper ways to get their products to market. Effective development tools play a significant role in that process. From programming languages to app frameworks to software development kits, the tools developers use determine how quickly they can have a product in front of consumers.

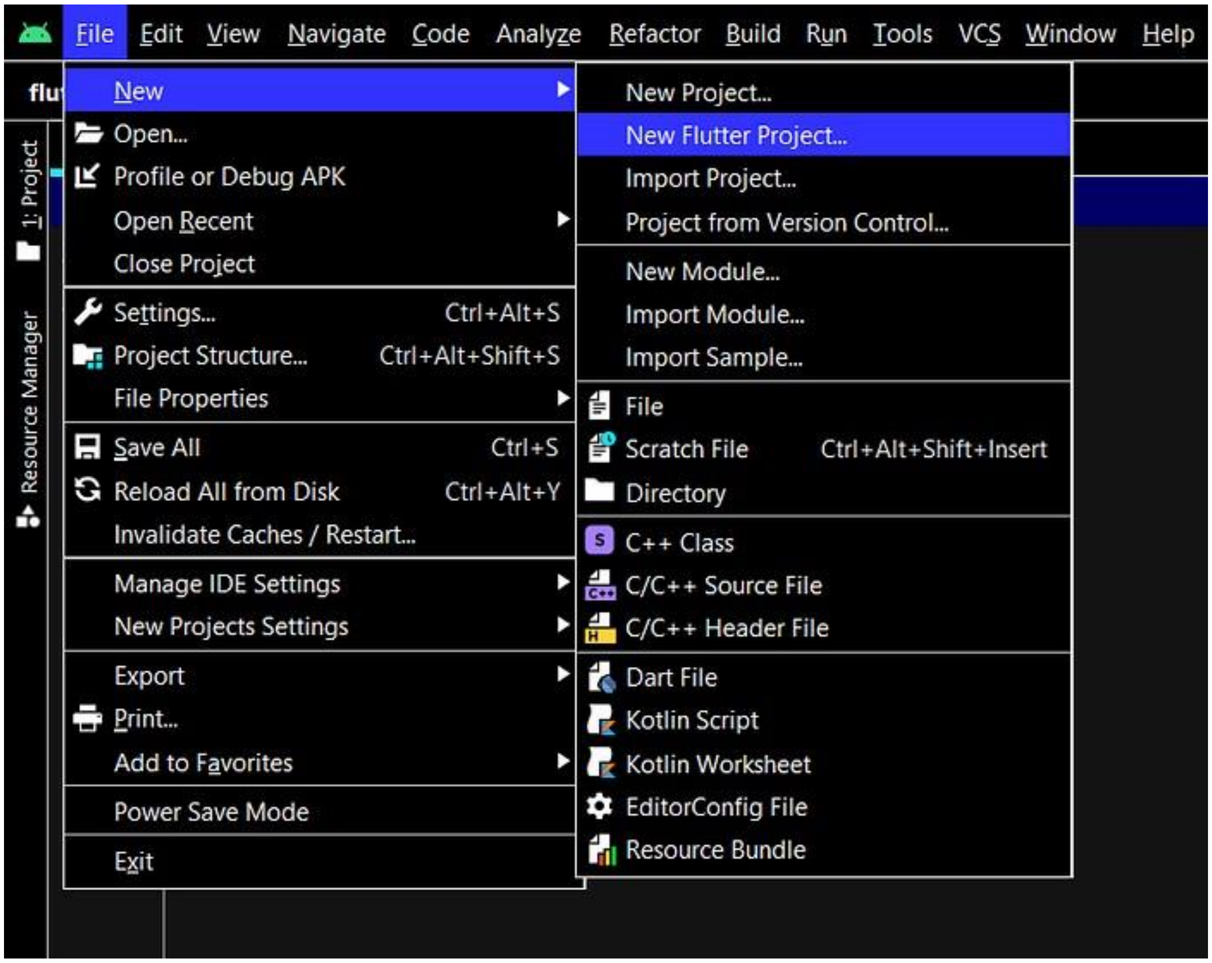
Let's take a look at Flutter's explosive rise in popularity and the features that make it so advantageous for developers.

Android Studio and Creating Simple Hello world Application with flutter

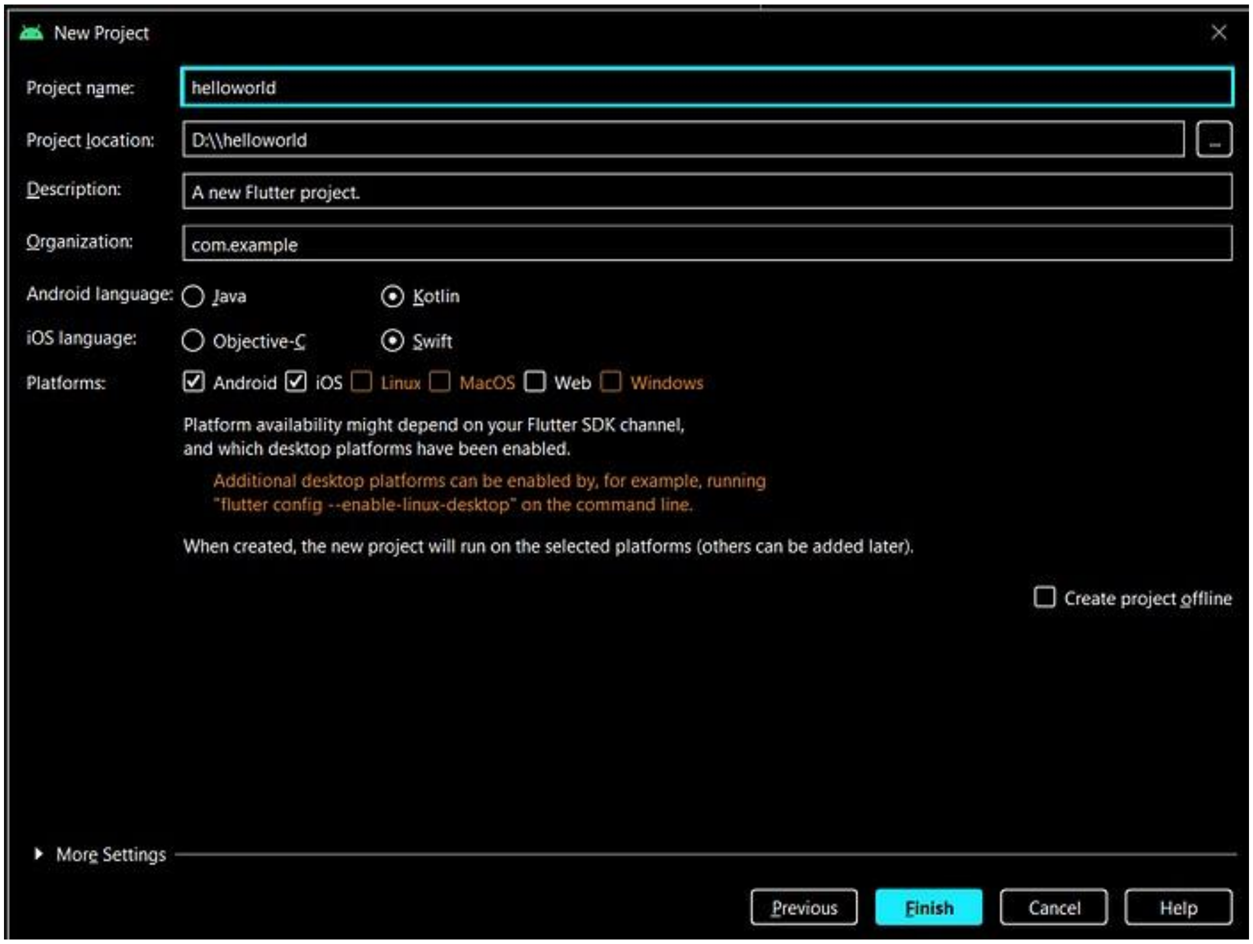
Let us Create a Simple Flutter Application to Understand ofbasics of creating a Flutter Application in Android Studio.

Step 1 : Open Android Studio.

Step 2 : Create Flutter Project , (File -> New -> New FlutterProject)



Step 3 : Select Flutter Application , and Click Next . and Then Enter Project name , Project Location , Project Description andFill-up all Details in Dialogue box and Click Next.



Project name: helloworld

Project location: D:\\helloworld

Description: A new Flutter project.

Organization: com.example

Android language: Java [✓ Kotlin]

iOS language: Objective-C [✓ Swift]

Platforms: [✓ Android] [✓ iOS] Linux MacOS Web Windows

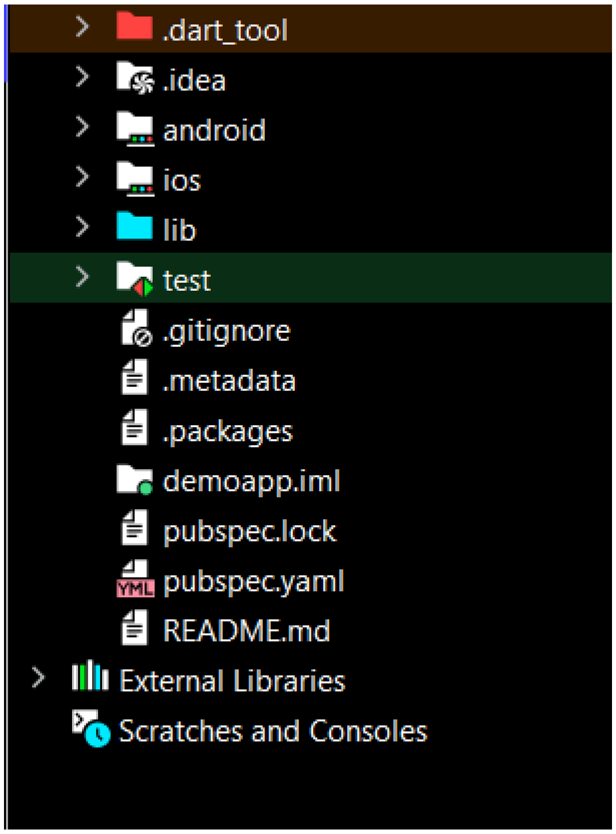
Platform availability might depend on your Flutter SDK channel,
and which desktop platforms have been enabled.

Additional desktop platforms can be enabled by, for example, running
"flutter config --enable-linux-desktop" on the command line.

When created, the new project will run on the selected platforms (others can be added later).

Create project offline

Step 4 : Flutter Application is Successfully Created and then wesee The Structure of Application in Left-top side following .



```
> .dart_tool
> .idea
> android
> ios
> lib
> test
  .gitignore
  .metadata
  .packages
  demoapp.iml
  pubspec.lock
  pubspec.yaml
  README.md
> External Libraries
  Scratches and Consoles
```

Various Components of structure of the Application areExplained Here 👇

Android : Source Code to Create Android Application

ios : Source Code for Create ios Application

lib : dart files Containing Folder

lib/main.dart : Entry Point of the Flutter Application test : Test Folder Containing the Dart Code for Testing a Flutter Application

test/widget_test.dart : Sample or Dummy Code

.gitignore : Git Version Control file

.metadata : Flutter Tools

.packages : file to Track Flutter Packages

.iml : Project file used by Android Studio

pubspec.yaml : Flutter Package Manager

pubspec.lock : Auto Generated by Flutter Package Manager

README.md : Project Description File

Step 5 : Replace Sample Code With Below code in lib/main.dartfile :-

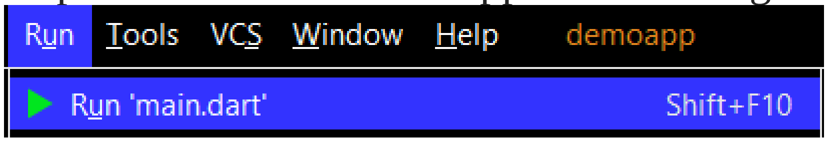
```
import 'package:flutter/material.dart';

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

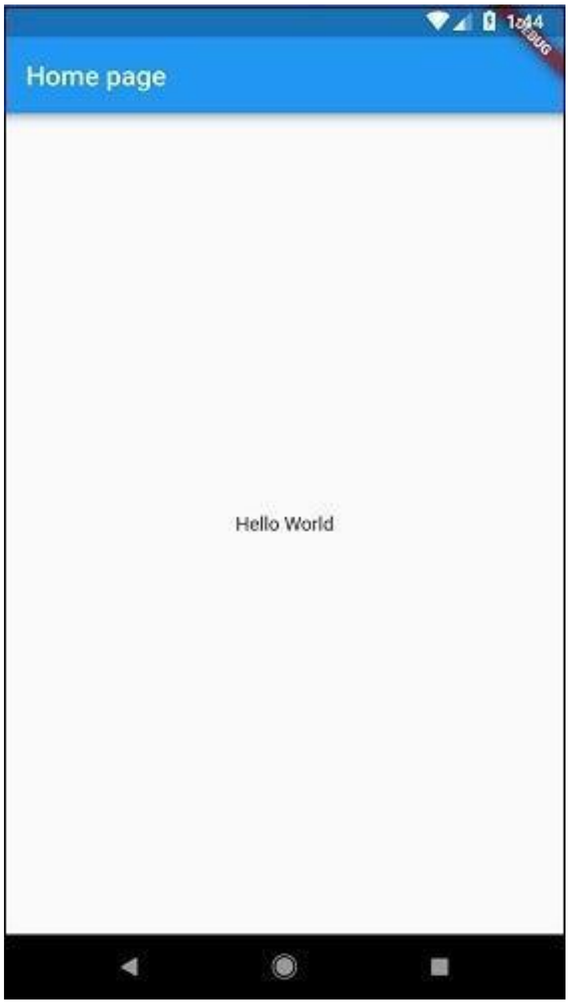
```
class MyApp extends StatelessWidget {
   // This widget is the root of your application.
   @override
   Widget build(BuildContext context) {
      return MaterialApp(
         title: 'Hello World Demo Application',
         theme: ThemeData(
            primarySwatch: Colors.blue,
         ),
         home: MyHomePage(title: 'Home page'),
      );
   }
}
class MyHomePage extends StatelessWidget {
   MyHomePage({Key key, this.title}) : super(key: key);
   final String title;

   @override
   Widget build(BuildContext context) {
      return Scaffold(
         appBar: AppBar(
            title: Text(this.title),
         ),
         body: Center(
            child:
            Text(
               'Hello World',
            )
         ),
      );
   }
}
```

Step 6 : Now Run Flutter Application Using below Figure :



Step 7 : Finally , The Output of Our Flutter Application :



When we build any dart application it takes time to execute for the first time. So to resolve this problem we have two features in a flutter, namely Hot Reload and Hot restart which help to decrease the execution time of our app once we execute it. These two features help to decrease the execution time. They are far better and faster than the default restart. It is important to note that it can only be used if you have executed your program once.

Hot Reload: A hot reload is a great functionality present in a flutter. It is the easiest and the fastest function which helps you to apply changes, fix bugs, creating UIs, and add features. It takes approximately one second to perform its functionality. In hot reload it does not destroy the preserved state. But you cannot use a hot reload once the app gets killed.

Perform Hot Reload:

- Run your flutter editor from the app or using the command prompt. We can use hot reload in flutter debug mode.
- Once your flutter project has been created do some changes in your code and perform a hot reload.
- In windows, you can perform a hot reload using 'ctrl+\' or using the hot reload button. In mac devices, you perform a hot reload using 'cmd+s'. If you are working in the command prompt using flutter run enter 'r' to run.

Hot Restart:

A hot restart has a slightly different functionality as compared to a hot reload. It is faster as compared to the full restart function. It destroys the preserved states of our app, and the code gets fully compiled again and starts from the default state. It takes more time as compared to hot reload but takes less time than the full restart function.

Perform Hot Restart:

- Run your flutter editor from the app or using the command prompt.
- Once your flutter project has been created do some changes in yourcode and perform a hot restart.
- You can perform a hot restart using the hot reload button or pressing ctrl+shift+\.

Key Differences are as follows:

Hot Reload

- It performs very fast as compared to hot restart or default restart offlutter.
- If we are using the state in our app then hot reload will not change thestate of the app.
- We perform hot reload by using key ctrl+\.

Hot Restart

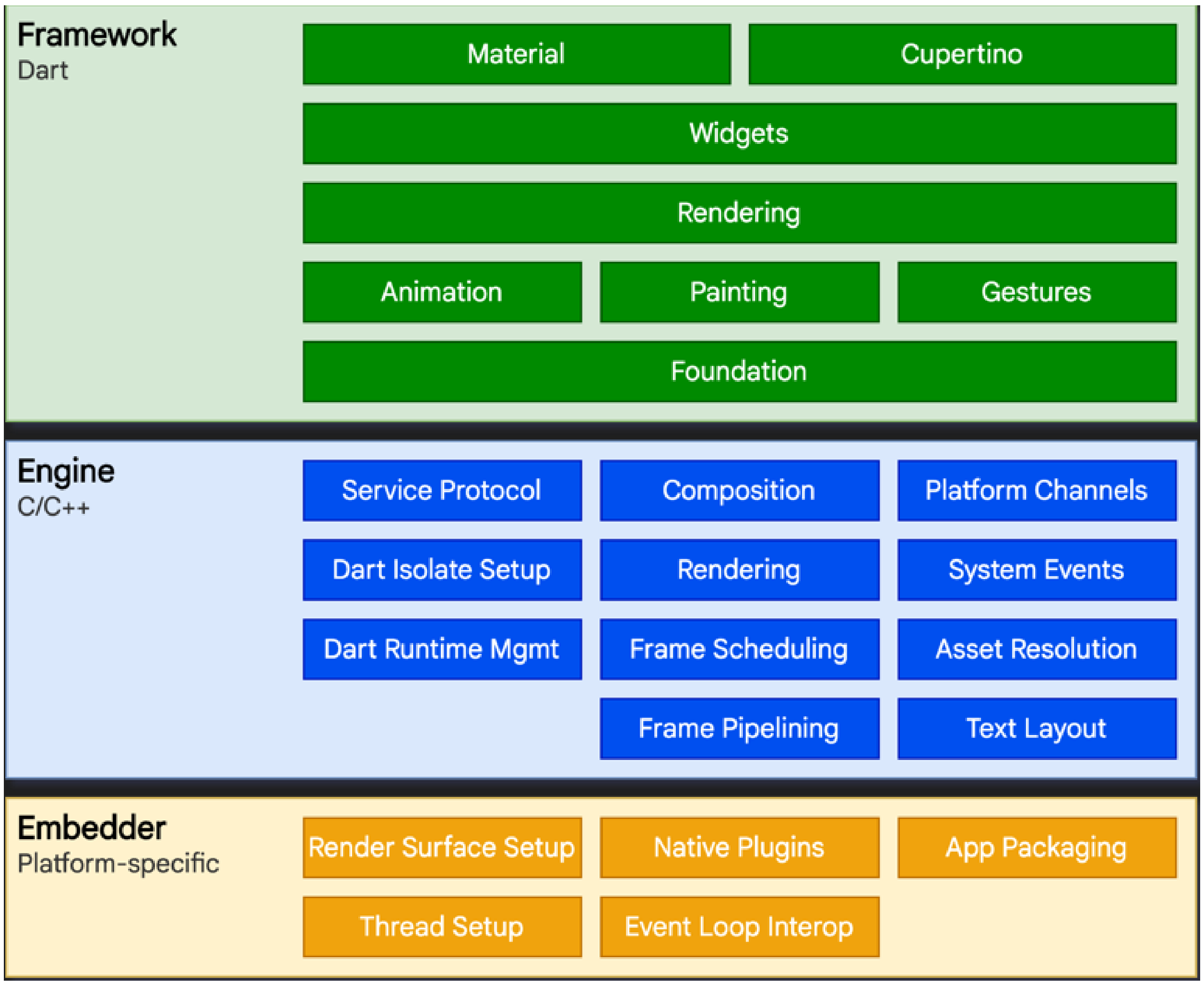
- It is slower than hot reload but faster than the default restart.
- It doesn’t preserve the state of our it starts from the initial state of ourapp.
- We perform hot restart using ctrl+shift+\

Video

Hot Reload	Hot Restart
Hot Reload allows us to see the reflected change after bug fixes, building User interfaces and even adding certain features to the app withoutrunning your application afresh over and over again.	Hot restart destroys the preservedState value and set them to their default.
When Hot Reload is invoked, the host machine checks the edited code since the last compilation and recompiles that again.	Hot Restart takes much higher time than Hot reload.
Hot Reload does not work when Enumerated types are changed to regular Classes and when classes are changed to enumerated types.	Hot reload is also know as ‘stateful hot reload’
Hot Reload does not work when generic types are modified	Hot Reload is useful because it saves time by just implementing the functionality based on the closest build class in less than 10secs.

Architectural layers

Flutter is designed as an extensible, layered system. It exists as a series of independent libraries that each depend on the underlying layer. No layer has privileged access to the layer below, and every part of the framework level is designed to be optional and replaceable.



To the underlying operating system, Flutter applications are packaged in the same way as any other native application. A platform-specific embedder provides an entrypoint; coordinates with the underlying operating system for access to services like rendering surfaces, accessibility, and input; and manages the message event loop. The embedder is written in a language that is appropriate for the platform: currently Java and C++ for Android, Objective-C/Objective-C++ for iOS and macOS, and C++ for Windows and Linux. Usingthe embedder, Flutter code can be integrated into an existing application as amodule, or the code may be the entire content of the application. Flutter includes a number of embedders for common target platforms, but other embedders also exist.

At the core of Flutter is the Flutter engine, which is mostly written in C++ and supports the primitives necessary to support all Flutter applications. The engine is responsible for rasterizing composited scenes whenever a new frame needs to be painted. It provides the low-level implementation of Flutter'score API, including graphics (through Impeller on iOS and coming to Android, and Skia on other platforms) text layout, file and network I/O, accessibility support, plugin architecture, and a Dart runtime and compile toolchain.

The engine is exposed to the Flutter framework through dart:ui, which wrapsthe underlying C++ code in Dart classes. This library exposes the lowest-levelprimitives, such as classes for driving input, graphics, and text rendering subsystems.

Typically, developers interact with Flutter through the Flutter framework, which provides a modern, reactive framework written in the Dart language. It includes a rich set of platform, layout, and foundational libraries, composed ofa series of layers. Working from the bottom to the top, we have:

- Basic foundational classes, and building block services such as animation, painting, and gestures that offer commonly usedabstractions over the underlying foundation.
- The rendering layer provides an abstraction for dealing with layout.With this layer, you can build a tree of renderable objects. You can manipulate these objects dynamically, with the tree automatically updating the layout to reflect your changes.
- The widgets layer is a composition abstraction. Each render object in the rendering layer has a corresponding class in the widgets layer. In addition, the widgets layer allows you to define combinations of classesthat you can reuse. This is the layer at which the reactive programmingmodel is introduced.
- The Material and Cupertino libraries offer comprehensive sets of controls that use the widget layer's composition primitives to implementthe Material or iOS design languages.

The Flutter framework is relatively small; many higher-level features thatdevelopers might use are implemented as packages, including platform plugins like camera and webview, as well as platform-agnostic features like characters, http, and animations that build upon the core Dart and Flutter libraries. Some of these packages come from the broader ecosystem, covering services like in-app payments, Apple authentication, and animations.

The rest of this overview broadly navigates down the layers, starting with the reactive paradigm of UI development. Then, we describe how widgets are composed together and converted into objects that can be rendered as part ofan application. We describe how Flutter interoperates with other code at a platform level, before giving a brief summary of how Flutter's web support differs from other targets.