

A Seminar On
“Flutter By Google”

For the Partial Fulfilment of the Award of the Degree of
Bachelor of Computer Application of Veer Narmad South
Gujarat University, Surat

Bachelor of Computer Application [B.C.A]
Semester – VI

By
[Kakadiya Nikunj B.] Exam No[1393]

Guided By
[Mr. Darshan G. Devdhara]

DEVIBA INSTITUTE OF COMPUTER APPLICATION, SABARGAM

Year : 2019-2020



**AMBABA COMMERCE COLLEGE, MANIBA
INSTITUTE OF BUSINESS MANAGEMENT &
DEVIBA INSTITUTE OF COMPUTER APPLICATION,
SABARGAM (Self Finance)**

At. Sabargam, Po. Niyol, Tal. Choryasi, Surat-394 325.

Managed By Shree Dakshin Gujarat Shikshan Samaj, Kumbharia

Certificate

This is to certify that _____ Kakadiya Nikunj B. _____

Exam Seat Number: _____ 1393 _____ have/has satisfactorily presented their seminar work Entitled **"Flutter by google"** as a partial fulfillment of requirements for **6th Semester – B.C.A.**, during the academic Year 2019-2020.

Guide :

Dr. Chetan C. Patel

I/C Principal

**Ambaba Commerce College, MIBM &
Deviba Institute of Computer Application, Sabargam**

Date :

Place : Sabargam, Surat

Abstract

Flutter is an open-source UI software development kit created by Google. It is used to develop applications for Android, iOS, Windows, Mac, Linux, Google Fuchsia and the web from a single codebase.

The first version of Flutter was known as codename "Sky" and ran on the Android operating system. It was unveiled at the 2015 Dart developer summit, with the stated intent of being able to render consistently at 120 frames per second. During the keynote of Google Developer Days in Shanghai, Google announced Flutter Release Preview 2 which is the last big release before Flutter 1.0. On December 4, 2018, Flutter 1.0 was released at the Flutter Live event, denoting the first "stable" version of the Framework. On December 11, 2019, Flutter 1.12 was released at the Flutter Interactive event.

The Dart SDK in version 2.8 and the Flutter in version 1.17.0 were released, where support was added to the Metal API, improving performance on iOS devices (approximately 50%), new Material widgets, and new network tracking tools.

Introduction

Flutter is a multi platform, open source, and free framework for creating mobile applications, created by Google. It is very easy to learn and currently it is getting more and more popular. With this blogpost you will learn some basic stuff about Flutter, and after reading it, you will be able to create a simple application using this technology.

To write applications in Flutter, You need to use the Dart language. Dart is scripting language, which in my opinion, is somewhat similar to both Java and JavaScript. You can check this language following this link:

Flutter widgets are built using a modern framework that takes inspiration from React. The central idea is that you build your UI out of widgets. Widgets describe what their view should look like given their current configuration and state. When a widget's state changes, the widget rebuilds its description, which the framework diffs against the previous description in order to determine the minimal changes needed in the underlying render tree to transition from one state to the next.

When writing an app, you'll commonly author new widgets that are subclasses of either `StatelessWidget` or `StatefulWidget`, depending on whether your widget manages any state. A widget's main job is to implement a `build()` function, which describes the widget in terms of other, lower-level widgets. The framework builds those widgets in turn until the process bottoms out in widgets that represent the underlying `RenderObject`, which computes and describes the geometry of the widget.

1. Mobile Development Approaches

When it comes to developing mobile applications, one major issue is to decide what all operating systems should the application support. Majority of applications stick to just Android and IOS because of their unprecedented market share in Mobile OS.

Second major issue is to decide the approach while developing the application for both platforms. Listed below are few such approaches:

1. Single Platform — Native Application
2. Multiple Platform — Hybrid or Native Applications

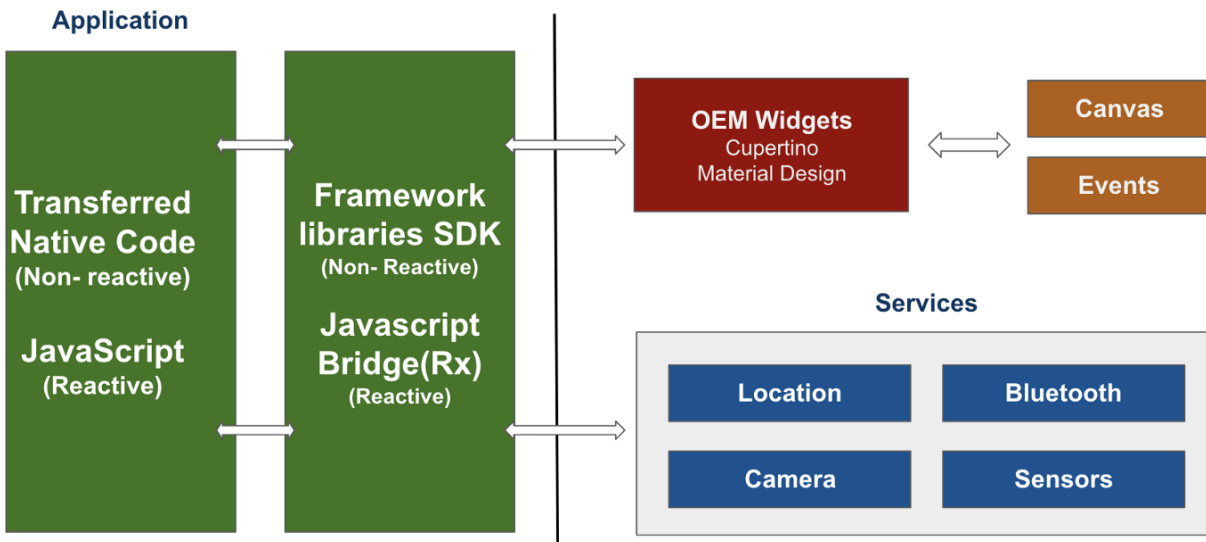
1.1:- Single Platform - Native Application

Hybrid Application: These are web applications (or web pages) in the native browser, such as UIWebView in iOS and WebView in Android (not Safari or Chrome). Hybrid apps are developed using HTML, CSS and Javascript, and then wrapped in a native application using platforms like Cordova, Ionic, PhoneGap etc.

Cross-Platform Applications: These are types of software application that work on multiple operating systems or devices, which are often referred to as platforms. A platform means an operating system such as Windows, Mac OS, Android or iOS.

1.2:- Multiple Platforms – Hybrid or Native App

Cross-Platform Approach



To solve few of the bottlenecks faced in fore-mentioned approaches, Flutter comes to our rescue, checking most boxes and giving us good reasons to choose it as a framework for cross-platform mobile application development.

To start with, the first real advantage of using flutter is that it's fast. The whole intent behind launching it was to provide fluid user interface at 120fps. Its eliminates the need of bridge to render views as it comes with its own Flutter engine(c++) to render views.

Pros: Speed, performance, flexibility, native look and feel, rapid development.

Cons: Its relatively new in comparison to other options in the list. The apk/ipa size is relatively large as it is shipped with the native code, the render engine and platform channel which is a problem for all cross-platform application approach.

2:- What is Flutter ?

Flutter is an open-source UI software development kit created by Google. It is used to develop applications for Android, iOS, Windows, Mac, Linux, Google Fuchsia and the web from a single codebase. The first version of Flutter was known as codename "Sky" and ran on the Android operating system.]

Flutter is Google's portable UI toolkit for crafting beautiful, natively compiled applications for mobile, web, and desktop from a single codebase. Flutter works with existing code, is used by developers and organizations around the world, and is free and open source.

Stable release: v1.17.4 / June 19, 2020; 3 days ago

Preview release: 1.19.0-4.0.pre / June 5, 2020; 17 days ago

Developer(s): Google and community

Initial release date: May 2017

Original author(s): Google

Platforms: Android, iOS, Google Fuchsia, Web platform, Microsoft Windows, macOS, Linux

3:- What makes Flutter unique?

For users, Flutter makes beautiful app UIs come to life.

For developers, Flutter lowers the bar to entry for building apps. It speeds up development of apps and reduces the cost and complexity of app production across platforms.

For designers, Flutter helps deliver the original design vision, without loss of fidelity or compromises. It also acts as a productive prototyping tool.

- Compiles to Native Code (ARM Binary code)
- No reliance on OEM widgets - No bridge needed

Flutter is different than most other options for building mobile apps because Flutter uses neither WebView nor the OEM widgets that shipped with the device. Instead, Flutter uses its own high-performance rendering engine to draw widgets.

In addition, Flutter is different because it only has a thin layer of C/C++ code. Flutter implements most of its system (compositing, gestures, animation, framework, widgets, etc) in Dart (a modern, concise, object-oriented language) that developers can easily approach read, change, replace, or remove. This gives developers tremendous control over the system, as well as significantly lowers the bar to approachability for the majority of the system.

- **Who makes Flutter?**

Flutter is an open source project, with contributions from Google and the community.

4:- Who is Flutter for ?

Flutter is for developers that want a faster way to build beautiful apps, or a way to reach more users with a single investment.

Flutter is also for engineering managers that lead development teams. Flutter allows eng managers to create a single mobile, web, and desktop app dev team, unifying their development investments to ship more features faster, ship the same feature set to multiple platforms at the same time, and lower maintenance costs.

Flutter is also for designers that want their original design visions delivered consistently, with high fidelity, to all users. In fact, CodePen now supports Flutter.

Fundamentally, Flutter is for users that want beautiful apps, with delightful motion and animation, and UIs with character and an identity all their own.

- **Who uses Flutter?**

Developers inside and outside of Google use Flutter to build beautiful natively-compiled apps for iOS and Android. To learn about some of these apps, visit the showcase.

5. What does Flutter provide ?

Flutter 1.0 was launched on Dec 4th, 2018. Thousands of apps have shipped with Flutter to hundreds of millions of devices. See some sample apps in the showcase.

For more information on the launch and subsequent releases, see Flutter 1.0: Google's Portable UI Toolkit.

Heavily optimized, mobile-first 2D rendering engine with excellent support for text.

5.1 - What is inside the Flutter SDK ?

- Heavily optimized, mobile-first 2D rendering engine with excellent support for text
- Modern react-style framework
- Rich set of widgets implementing Material Design and iOS-style.
- APIs for unit and integration tests
- Interop and plugin APIs to connect to the system and 3rd-party SDKs
- Headless test runner for running tests on Windows, Linux, and Mac
- Dart DevTools for testing, debugging, and profiling your app
- Command-line tools for creating, building, testing, and compiling your apps

5.2 - Flutter work with any editors or IDEs

We support plugins for Android Studio, IntelliJ IDEA, and VS Code.

See editor configuration for setup details, and Android Studio/IntelliJ and VS Code for tips on how to use the plugins.

Alternatively, you can use a combination of the flutter command in a terminal and one of the many editors that support editing Dart.

5.3 - Flutter framework

Flutter ships with a modern framework, inspired by React. Flutter's framework is designed to be layered and customizable (and optional). Developers can choose to use only parts of the framework, or a different framework.

5.4 - Flutter widgets

Flutter ships with a set of high quality Material Design and Cupertino (iOS-style) widgets, layouts, and themes. Of course, these widgets are only a starting point. Flutter is designed to make it easy to create your own widgets, or customize the existing widgets.

5.5 - Flutter Material Theming

The Flutter and Material teams collaborate closely, and Material Theming is fully supported. A number of examples of this are shown in the MDC-103 Flutter: Material Theming codelab.

5.6 - Flutter testing framework

Flutter provides APIs for writing unit and integration tests. Learn more about testing with Flutter.

We use our own testing capabilities to test our SDK. We measure our test coverage on every commit.

6 – Technology

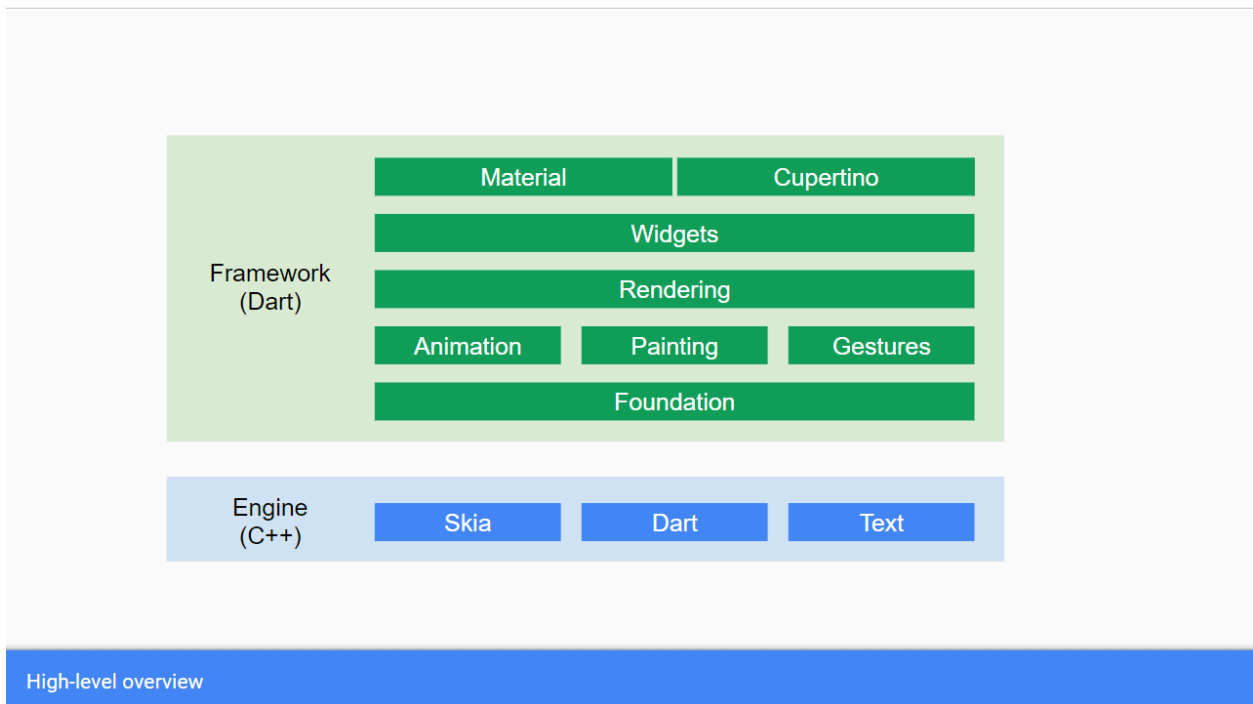
Technology is used in flutter is C, C++, Dart, the architecture diagram for a better.

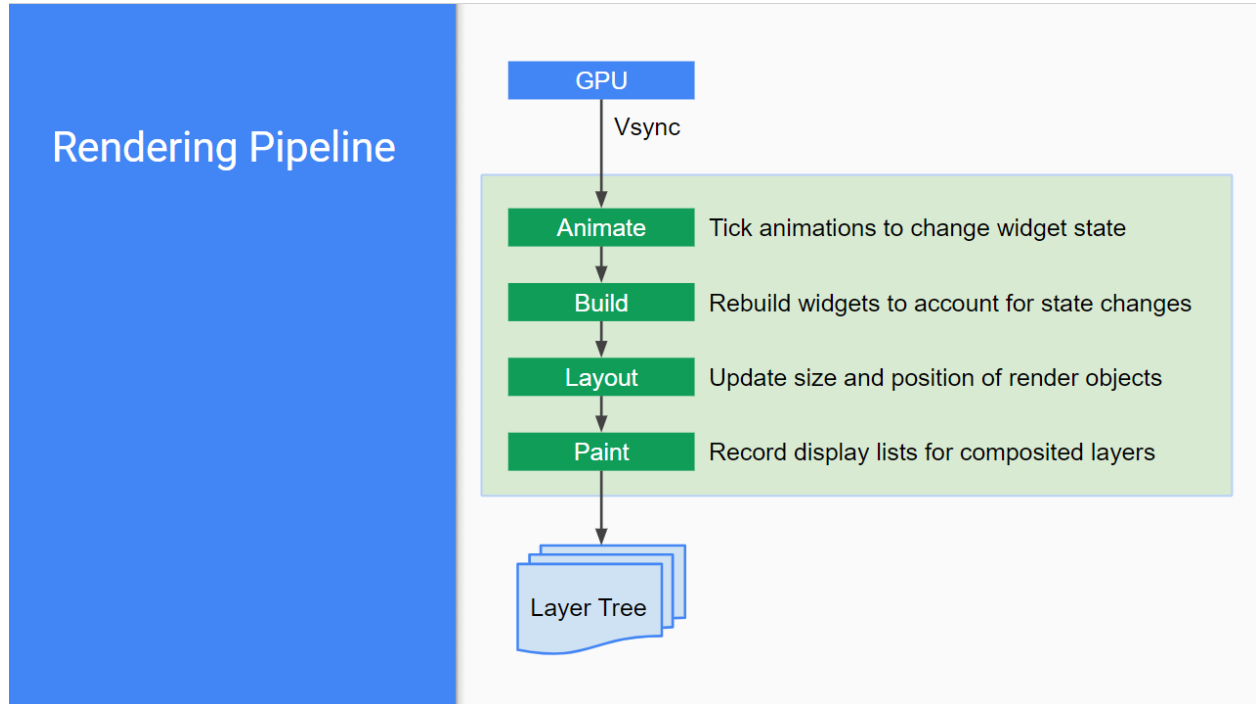
The topic of Technology are as follow:

1. Flutter built with
2. Flutter Run on Android
3. Flutter Run on Ios
4. Flutter use my system's OEM widgets
5. What happens when my mobile OS updates and introduces new widgets
6. Flutter choose to use Dart
7. Flutter engine

6.1.Flutter built with

Flutter is built with C, C++, Dart, and Skia (a 2D rendering engine). See this architecture diagram for a better picture of the main components.





6.2. Flutter Run on Android

The engine's C and C++ code are compiled with Android's NDK. The Dart code (both the SDK's and yours) are ahead-of-time (AOT) compiled into native, ARM, and x86 libraries. Those libraries are included in a "runner" Android project, and the whole thing is built into an APK. When launched, the app loads the Flutter library. Any rendering, input, or event handling, and so on, is delegated to the compiled Flutter and app code. This is similar to the way many game engines work.

Debug mode builds use a virtual machine (VM) to run Dart code (hence the "debug" banner they show to remind people that they're slightly slower) in order to enable Stateful Hot Reload.

6.3 - Flutter Run on Ios

The engine's C and C++ code are compiled with LLVM. The Dart code (both the SDK's and yours) are ahead-of-time (AOT) compiled into a native, ARM library. That library is included in a "runner" iOS project, and the whole thing is built into an .ipa. When launched, the app loads the Flutter library. Any rendering, input or event handling, and so on, are delegated to the compiled Flutter and app code. This is similar to the way many game engines work.

Debug mode builds use a virtual machine (VM) to run Dart code (hence the "debug" banner they show to remind people that they're slightly slower) in order to enable Stateful Hot Reload.

6.4 - Flutter use my system's OEM widgets

No. Instead, Flutter provides a set of widgets (including Material Design and Cupertino (iOS-styled) widgets), managed and rendered by Flutter's framework and engine. You can browse a catalog of Flutter's widgets.

We hope that the end-result is higher quality apps. If we reused the OEM widgets, the quality and performance of Flutter apps would be limited by the quality of those widgets.

In Android, for example, there's a hard-coded set of gestures and fixed rules for disambiguating them. In Flutter, you can write your own gesture recognizer that is a first-class participant in the gesture system. Moreover, two widgets authored by different people can coordinate to disambiguate gestures.

Modern app design trends point towards designers and users wanting more motion-rich UIs and brand-first designs. In order to achieve that level of customized, beautiful design, Flutter is architected to drive pixels instead of the OEM widgets.

By using the same renderer, framework, and set of widgets, it's easier to publish for both iOS and Android concurrently, without having to do careful and costly planning to align two separate codebases and feature sets.

By using a single language, a single framework, and a single set of libraries for all of your UI (regardless if your UI is different for each platform or largely consistent), we also aim to help lower app development and maintenance costs.

6.5-What happens when my mobile OS updates and introduces new widgets ?

The Flutter team watches the adoption and demand for new mobile widgets from iOS and Android, and aims to work with the community to build support for new widgets. This work might come in the form of lower-level framework features, new composable widgets, or new widget implementations.

Flutter's layered architecture is designed to support numerous widget libraries, and we encourage and support the community in building and maintaining widget libraries.

6.6 - Flutter choose to use Dart

Flutter used four primary dimensions for evaluation, and considered the needs of framework authors, developers, and end users. We found some languages met some requirements, but Dart scored highly on all of our evaluation dimensions and met all our requirements and criteria.

Dart runtimes and compilers support the combination of two critical features for Flutter: a JIT-based fast development cycle that allows for shape changing and stateful hot reloads in a language with types, plus an Ahead-of-Time compiler that emits efficient ARM code for fast startup and predictable performance of production deployments.

In addition, we have the opportunity to work closely with the Dart community, which is actively investing resources in improving Dart for use in Flutter. For example, when we adopted Dart, the language didn't have an ahead-of-time toolchain for producing native binaries, which is instrumental in achieving predictable, high performance, but now the language does because the Dart team built it for Flutter. Similarly, the Dart VM has previously been optimized for throughput but the team is now optimizing the VM for latency, which is more important for Flutter's workload.

Dart scores highly for us on the following primary criteria:

Developer productivity. One of Flutter's main value propositions is that it saves engineering resources by letting developers create apps for both iOS and Android with the same codebase. Using a highly productive language accelerates developers further and makes Flutter more attractive. This was very important to both our framework team as well as our

developers. The majority of Flutter is built in the same language we give to our users, so we need to stay productive at 100k's lines of code, without sacrificing approachability or readability of the framework and widgets for our developers.

Object-orientation. For Flutter, we want a language that's suited to Flutter's problem domain: creating visual user experiences. The industry has multiple decades of experience building user interface frameworks in object-oriented languages. While we could use a non-object-oriented language, this would mean reinventing the wheel to solve several hard problems. Plus, the vast majority of developers have experience with object-oriented development, making it easier to learn how to develop with Flutter.

Predictable, high performance. With Flutter, we want to empower developers to create fast, fluid user experiences. In order to achieve that, we need to be able to run a significant amount of end-developer code during every animation frame. That means we need a language that both delivers high performance and delivers predictable performance, without periodic pauses that would cause dropped frames.

Fast allocation. The Flutter framework uses a functional-style flow that depends heavily on the underlying memory allocator efficiently handling small, short-lived allocations. This style was developed in languages with this property and doesn't work efficiently in languages that lack this facility.

6.6 - Flutter choose to use Dart

In July 2019, we measured the download size of a minimal Flutter app (no Material Components, just a single Center widget, built with `flutter build apk --split-per-abi`), bundled and compressed as a release APK, to be approximately 4.3 MB for ARM, and 4.6 MB for ARM 64.

In ARM, the core engine is approximately 3.2 MB (compressed), the framework + app code is approximately 920.6 KB (compressed), the LICENSE file is 54.3 KB (compressed), necessary Java code (classes.dex) is 113.6 KB (compressed).

In ARM64, the core engine is approximately 3.5 MB (compressed), the framework + app code is approximately 872 KB (compressed), the LICENSE file is 54.3 KB (compressed), necessary Java code (classes.dex) is 113.6 KB (compressed).

These numbers were measured using apkanalyzer, which is also built into Android Studio.

On iOS, a release IPA of the same app has a download size of 10.9 MB on an iPhone X, as reported by Apple's App Store Connect. The IPA is larger than the APK mainly because Apple encrypts binaries within the IPA, making the compression less efficient (see the iOS App Store Specific Considerations section of Apple's QA1795).

The release engine binary includes LLVM IR (bitcode). Xcode uses this bitcode to produce a final binary for the App Store containing the latest compiler optimizations and features. The profile and debug frameworks contain only a bitcode marker, and are more representative of the engine's actual binary size. Whether you ship with bitcode or not, the increased size of the release framework is stripped out during the final steps of the build. These steps happen after archiving your app and shipping it to the store.

Of course, we recommend that you measure your own app. To do that, see [Measuring your app's size](#).

7.What is dart ?

Dart is an open-source general-purpose programming language. It is originally developed by Google and later approved as a standard by ECMA. Dart is a new programming language meant for the server as well as the browser. Introduced by Google, the Dart SDK ships with its compiler – the Dart VM. The SDK also includes a utility -dart2js, a transpiler that generates JavaScript equivalent of a Dart Script. This tutorial provides a basic level understanding of the Dart programming language.

Dart is an object-oriented language with C-style syntax which can optionally trans compile into JavaScript. It supports a varied range of programming aids like interfaces, classes, collections, generics, and optional typing.

Dart can be extensively used to create single-page applications. Single-page applications apply only to websites and web applications. Single-page applications enable navigation between different screens of the website without loading a different webpage in the browser. A classic example is GMail — when you click on a message in your inbox, browser stays on the same webpage, but JavaScript code hides the inbox and brings the message body on screen.

Google has released a special build of Chromium – the Dart VM. Using Dartium means you don't have to compile your code to JavaScript until you're ready to test on other browsers.

The following table compares the features of Dart and JavaScript.

Google AdWords => Google Fuchsia

AdSense => Mandrill
performance reports

Google internal => Google internal
sales tool CRM

Dart Documentation:

<https://www.dartlang.org/community/who-uses-dart>

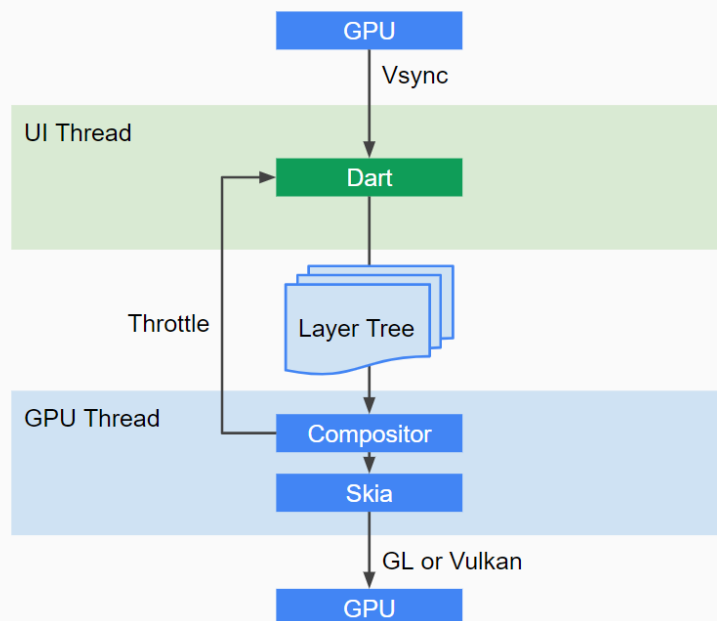
8. Capabilities

1. devices and OS versions does Flutter run on

- Mobile operating systems: Android Jelly Bean, v16, 4.1.x or newer, and iOS 8 or newer.

- Mobile hardware: iOS devices (iPhone 4S or newer) and ARM Android devices.
- Flutter supports building ahead-of-time (AOT) compiled libraries for x86_64, armeabi-v7a, and arm64-v8a.
- Apps built for ARMv7 or ARM64 run fine (using ARM emulation) on many x86 Android devices.
- We support developing Flutter apps with Android and iOS devices, as well as with Android emulators and the iOS simulator.
- We test on a variety of low-end to high-end phones and tablets, but we don't yet have an official device compatibility guarantee.

Graphics Pipeline





Google Fuchsia

Fuchsia is a capability-based, real-time operating system (RTOS) currently being developed by Google

<https://github.com/fuchsia-mirror/>

9. How Flutter works ?

Flutter is approachable to programmers familiar with object-oriented concepts (classes, methods, variables, etc) and imperative programming concepts (loops, conditionals, etc).

No prior experience is required in order to learn and use Flutter.

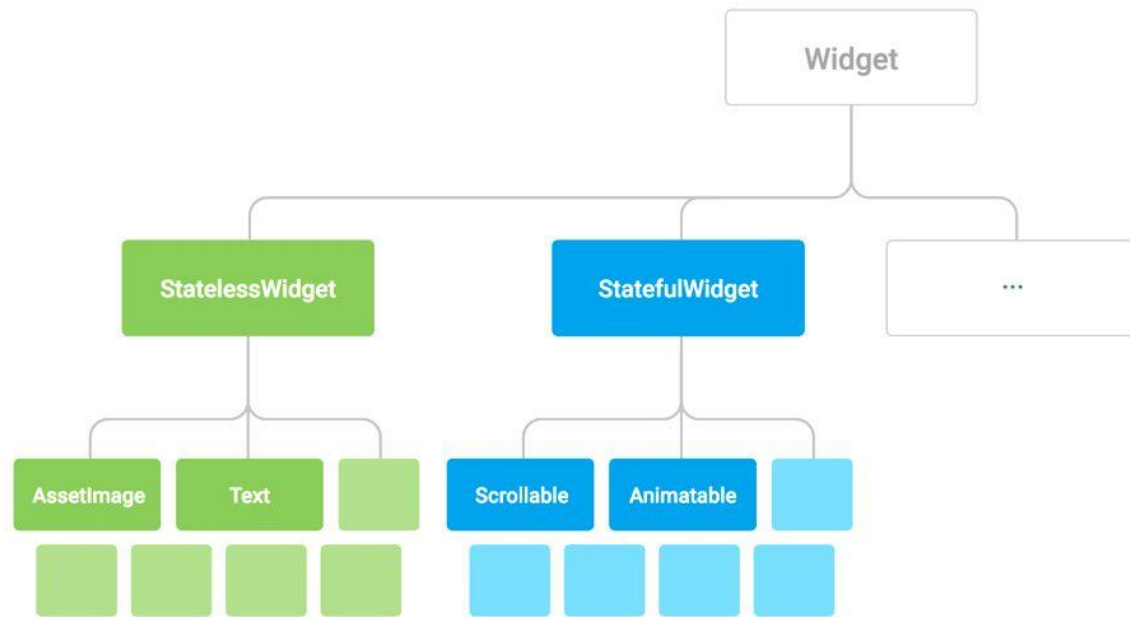
We have seen people with very little programming experience learn and use Flutter for prototyping and app development.

Flutter is optimized for 2D mobile apps that want to run on both Android and iOS. Flutter is also great for interactive apps that you want to run on your web pages or on the desktop. (Note that web support is in beta, and desktop support is in alpha.)


Apps that need to deliver brand-first designs are particularly well suited for Flutter. However, apps that need to look like stock platform apps can also be built with Flutter.

You can build full-featured apps with Flutter, including camera, geolocation, network, storage, 3rd-party SDKs, and more.

Everything is a Widget



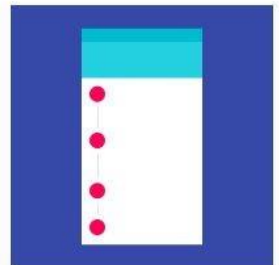
Layout



ListTile

A single fixed-height row that typically contains some text as well as a leading or trailing icon.

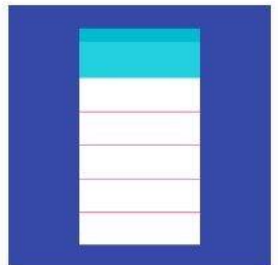
[Documentation](#)



Stepper

A material stepper widget that displays progress through a sequence of steps.

[Documentation](#)



Divider

A one logical pixel thick horizontal line, with padding on either side.

[Documentation](#)

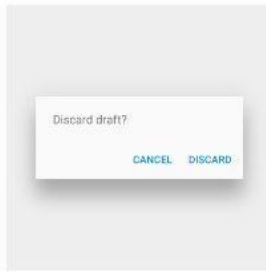
Dialogs, alerts, and panels



SimpleDialog

Simple dialogs can provide additional details or actions about a list item. For example they can display avatars icons clarifying subtext or orthogonal actions...

[Documentation](#)



AlertDialog

Alerts are urgent interruptions requiring acknowledgement that inform the user about a situation. The AlertDialog widget implements this component.

[Documentation](#)



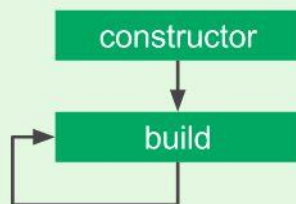
BottomSheet

Bottom sheets slide up from the bottom of the screen to reveal more content. You can call `showBottomSheet()` to implement a persistent bottom sheet or...

[Documentation](#)

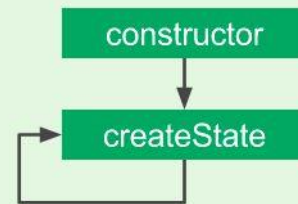
Reactive Module

StatelessWidget

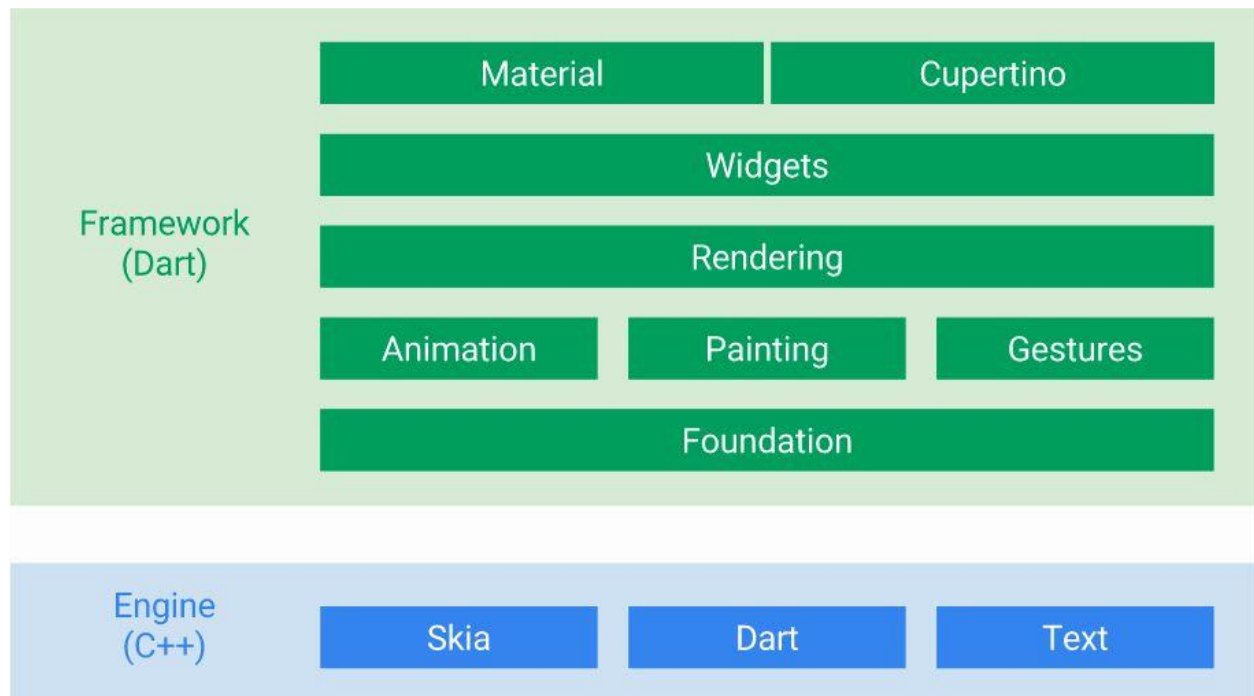
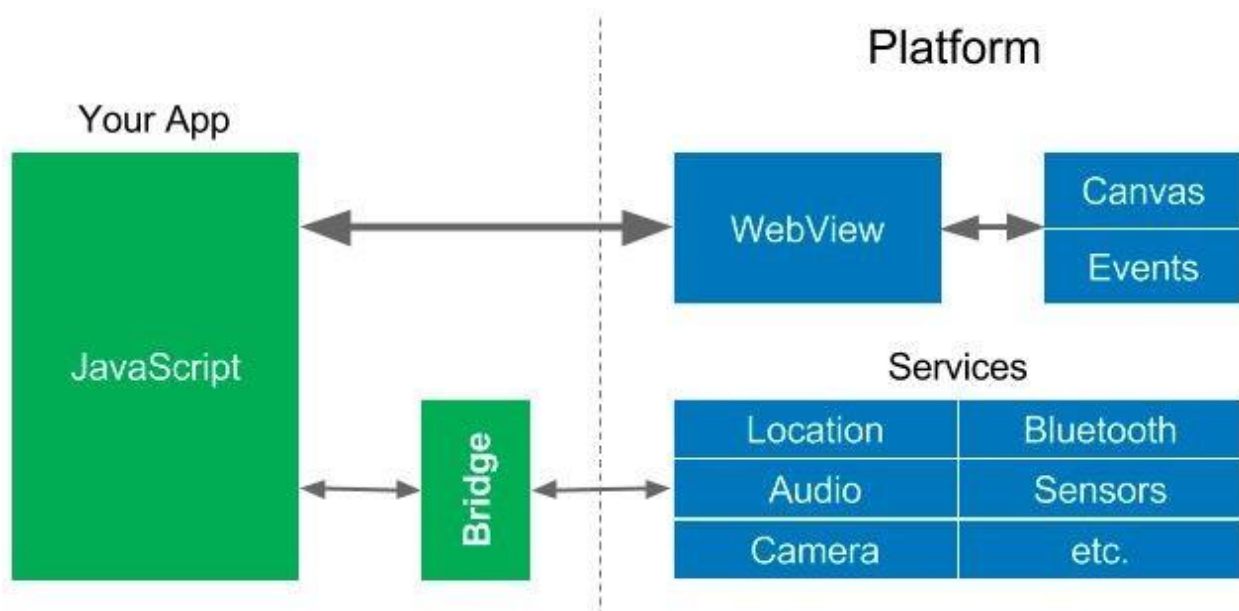


A single StatelessWidget can build in many different BuildContexts

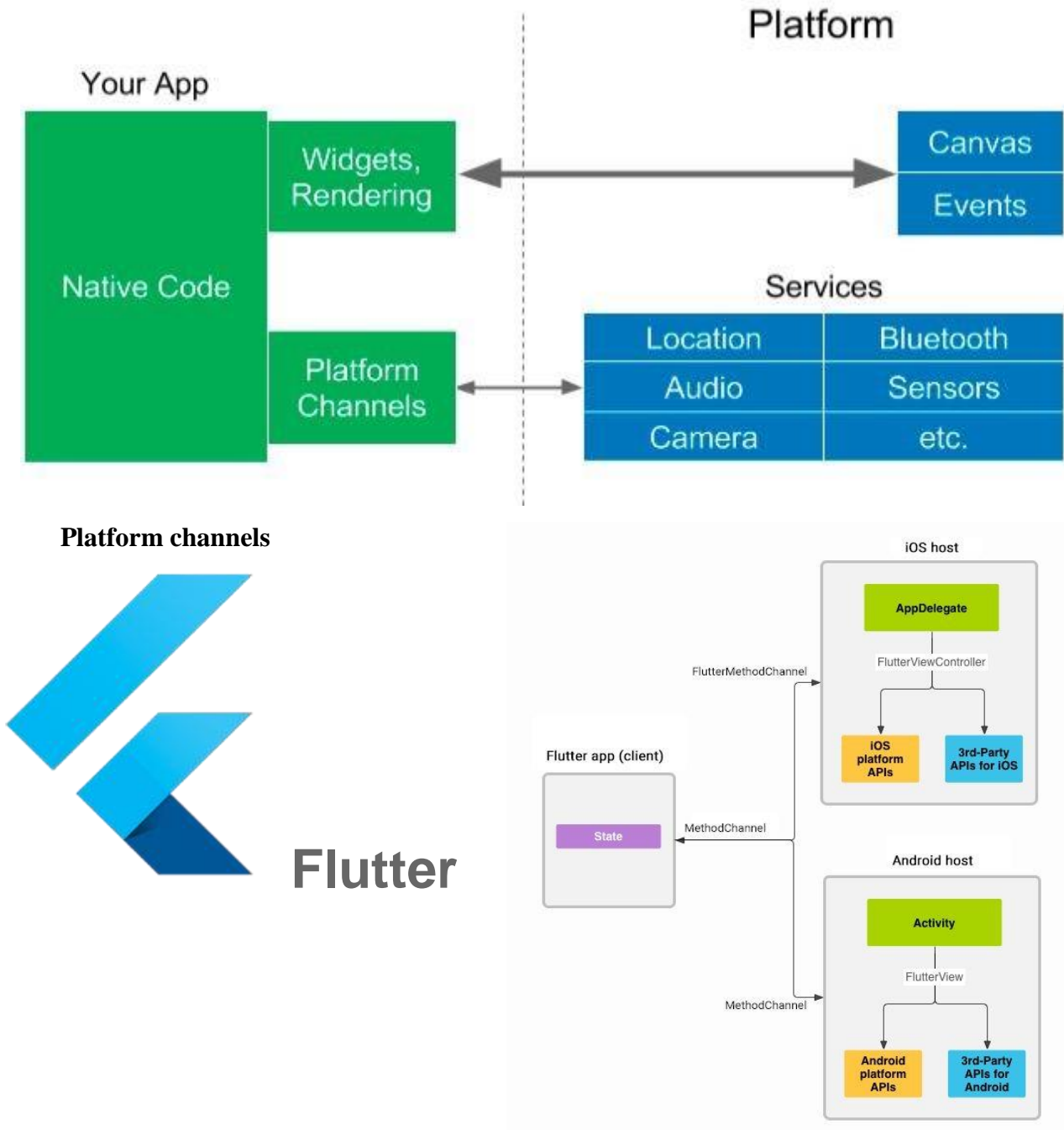
StatefulWidget



A StatefulWidget creates a new State object for each BuildContext

Framework Module :**WebViews**

Flutter :



HTML/CSS Analogs in Flutter:

```
<div class="greybox">
  Lorem ipsum
</div>

.greybox {
  background-color: #e0e0e0; /* grey 300 */
  width: 320px;
  height: 240px;
  font: 900 24px Georgia;
}
```

```
var container = new Container( // grey box
  child: new Text(
    "Lorem ipsum",
    style: new TextStyle(
      fontSize: 24.0
      fontWeight: FontWeight.w900,
      fontFamily: "Georgia",
    ),
  ),
  width: 320.0,
  height: 240.0,
  color: Colors.grey[300],
);
```

IDM Support ?



Terminal :

```
$ flutter create demoapp
```


What can do Flutter ?

- Right-to-left layouts
- Accessibility
- Customized UI
- Background execution
- Firebase integration
- New pub site
- iPhone X support
- Animated GIF support
- Video
- Widget inspector
- Android Studio support

20.What programming paradigm does Flutter's framework use?

Flutter is a multi-paradigm programming environment. Many programming techniques developed over the past few decades are used in Flutter. We use each one where we believe the strengths of the technique make it particularly well-suited. In no particular order:

- **Composition**

The primary paradigm used by Flutter is that of using small objects with narrow scopes of behavior, composed together to obtain more complicated effects. Most widgets in the Flutter widget library are built in this way. For example, the `Material FlatButton` class is built using a `MaterialButton` class, which itself is built using an `IconTheme`, an `InkWell`, a `Padding`, a `Center`, a `Material`, an `AnimatedDefaultTextStyle`, and a `ConstrainedBox`. The `InkWell` is built using a `GestureDetector`. The `Material` is built using an `AnimatedDefaultTextStyle`, a `NotificationListener`, and an `AnimatedPhysicalModel`. And so on. It's widgets all the way down. Functional programming.

Entire applications can be built with only `StatelessWidgets`, which are essentially functions that describe how arguments map to other functions, bottoming out in primitives that

compute layouts or paint graphics. (Such applications can't easily have state, so are typically non-interactive.) For example, the `Icon` widget is essentially a function that maps its arguments (color, icon, size) into layout primitives. Additionally, heavy use is made of immutable data structures, including the entire `Widget` class hierarchy as well as numerous supporting classes such as `Rect` and `TextStyle`. On a smaller scale, Dart's `Iterable` API, which makes heavy use of the functional style (`map`, `reduce`, `where`, etc), is frequently used to process lists of values in the framework.

- **Event-driven programming**

User interactions are represented by event objects that are dispatched to callbacks registered with event handlers. Screen updates are triggered by a similar callback mechanism. The `Listenable` class, which is used as the basis of the animation system, formalizes a subscription model for events with multiple listeners.

Class-based object-oriented programming

Most of the APIs of the framework are built using classes with inheritance. We use an approach whereby we define very high-level APIs in our base classes, then specialize them iteratively in subclasses. For example, our render objects have a base class (`RenderObject`) that is agnostic regarding the coordinate system, and then we have a subclass (`RenderBox`) that introduces the opinion that the geometry should be based on the Cartesian coordinate system (x/width and y/height).

- **Prototype-based object-oriented programming**

The `ScrollPhysics` class chains instances to compose the physics that apply to scrolling dynamically at runtime. This lets the system compose, for example, paging physics with platform-specific physics, without the platform having to be selected at compile time.

Imperative programming

Straightforward imperative programming, usually paired with state encapsulated within an object, is used where it provides the most intuitive solution. For example, tests are written in an imperative style, first describing the situation under test, then listing the invariants that the test must match, then advancing the clock or inserting events as necessary for the test.

Reactive programming

The widget and element trees are sometimes described as reactive, because new inputs provided in a widget's constructor are immediately propagated as changes to lower-level widgets by the widget's build method, and changes made in the lower widgets (for example, in response to user input) propagate back up the tree via event handlers. Aspects of both functional-reactive and imperative-reactive are present in the framework, depending on the needs of the widgets. Widgets with build methods that consist of just an expression describing how the widget reacts to changes in its configuration are functional reactive widgets (for example, the `Material Divider` class). Widgets whose build methods construct a list of children over several statements, describing how the widget reacts to changes in its configuration, are imperative reactive widgets (for example, the `Chip` class).

- **Declarative programming**

The build methods of widgets are often a single expression with multiple levels of nested constructors, written using a strictly declarative subset of Dart. Such nested expressions could be mechanically transformed to or from any suitably expressive markup language. For example, the `UserAccountsDrawerHeader` widget has a long build method (20+ lines), consisting of a single nested expression. This can also be combined with the imperative style to build UIs that would be harder to describe in a pure-declarative approach.

- **Generic programming**

Types can be used to help developers catch programming errors early. The Flutter framework uses generic programming to help in this regard. For example, the `State` class is parameterized in terms of the type of its associated widget, so that the Dart analyzer can catch mismatches of states and widgets. Similarly, the `GlobalKey` class takes a type parameter so that it can access a remote widget's state in a type-safe manner (using runtime checking), the `Route` interface is parameterized with the type that it is expected to use when popped, and collections such as `Lists`, `Maps`, and `Sets` are all parameterized so that mismatched elements can be caught early either during analysis or at runtime during debugging.

- **Concurrent programming**

Flutter makes heavy use of Futures and other asynchronous APIs. For example, the animation system reports when an animation is finished by completing a future. The image loading system similarly uses futures to report when a load is complete.

- **Constraint programming**

The layout system in Flutter uses a weak form of constraint programming to determine the geometry of a scene. Constraints (for example, for cartesian boxes, a minimum and maximum width and a minimum and maximum height) are passed from parent to child, and the child selects a resulting geometry (for example, for cartesian boxes, a size, specifically a width and a height) that fulfills those constraints. By using this technique, Flutter can usually lay out an entire scene with a single pass.

21. References

1. <https://flutter.dev/docs/resources/faq>
2. <https://www.google.com/>
3. <https://www.netguru.com/codestories/introduction-to-flutter-part-1>
4. [https://en.wikipedia.org/wiki/Flutter_\(software\)](https://en.wikipedia.org/wiki/Flutter_(software))