Why you choose Flutter over Native Android or iOS Development?

Top 10 reasons that why android developers around the world are shifting towards Flutter,

1. Developer Friendly: So, Flutter is very developer friendly as most of the widget it provides are pre-built like rows, columns and containers and lot more. So, we just need to modify and use it in our application which helps to build beautiful user interface, more quickly and with less code.

2. More Efficient: So, when it comes to efficiency, Flutter is just wow means the work that Java can complete in 179 file and 12176 lines, Dart can complete in just 31 files and 1735 lines and that's like more than 10 times efficient. Similarly, Flutter is way more efficient as compared to all other platforms used for Android Development which saves a lot of time and effort.

3. Cross Platform: As Flutter is a cross-platform from a single code base, you can make applications for iOS, Android, Web as well as Desktop which reduces a lot of work if you want to create a product for all the platforms and thus Flutter proves itself as the best solution.

4. Hot Reload: So, while developing one of the pains is when you have to run the changes in code. In other platforms it takes 10 to 15 seconds to save the code, compile it and see the output on the screen, but in the case of Flutter, it just takes fraction of seconds and that's the power of Flutter, which makes it fast and smooth.

5. Dart Language: So, Flutter uses Dart programming language which is very easy to learn and powerful language and is very similar to modern object-oriented programming languages. So, if you have any prior experience to programming, it's very easy to get started with Dart.

6. Open Source: So, Flutter has one of the largest open-source communities that has emerged in past few years, so it provides enriched documentation about each and every thing in Flutter and with all the kind and helping people in the community, it become really easy to get started with Flutter as they are always up to help you by either giving some advice or referring some documentation.

7. Freelance Opportunities: So, freelance opportunities for Flutter are increasing rapidly. Last year there was a 320 percent increase in the freelance opportunities for Flutter, which clearly indicates the demand for Flutter and it's only gonna get bigger and that's what attract developers towards it.

8. Single File: So, while developing Android applications from other platforms we have maintain two files separately, one for user interface and one for its functionality, but in case if Flutter, we just have to maintain a single file which reduces a lot of work as from a single file, you can maintain both its UI as well as Functionalities in an easy way.

9. Packages and Plugins: So, Flutter provides us with a lot of packages and plugins for Android Development like AR core for augment reality, ML kit for Machine Learning and a lot more. So, we can make applications for whatever we want with ease, and it works really smooth with the integration of different technologies like Machine Learning, Augmented reality and a lot more which makes it really easy to explore different technologies with Flutter.

10. Compatible with All IDE: So, when it comes to development, IDE plays a really important role and in case of Flutter you can get started with the IDE that you are most comfortable with as Flutter integrates with all the major development tools like Android Studio, Visual Studio, Xbox, Eclipse, NetBeans and a lot more.

So, Flutter has really come out as a winner as compared to all other platforms for Android and iOS Development as Flutter apps look native, smooth and slick, just as Native Android and that's what make Flutter the first choice for Developers.

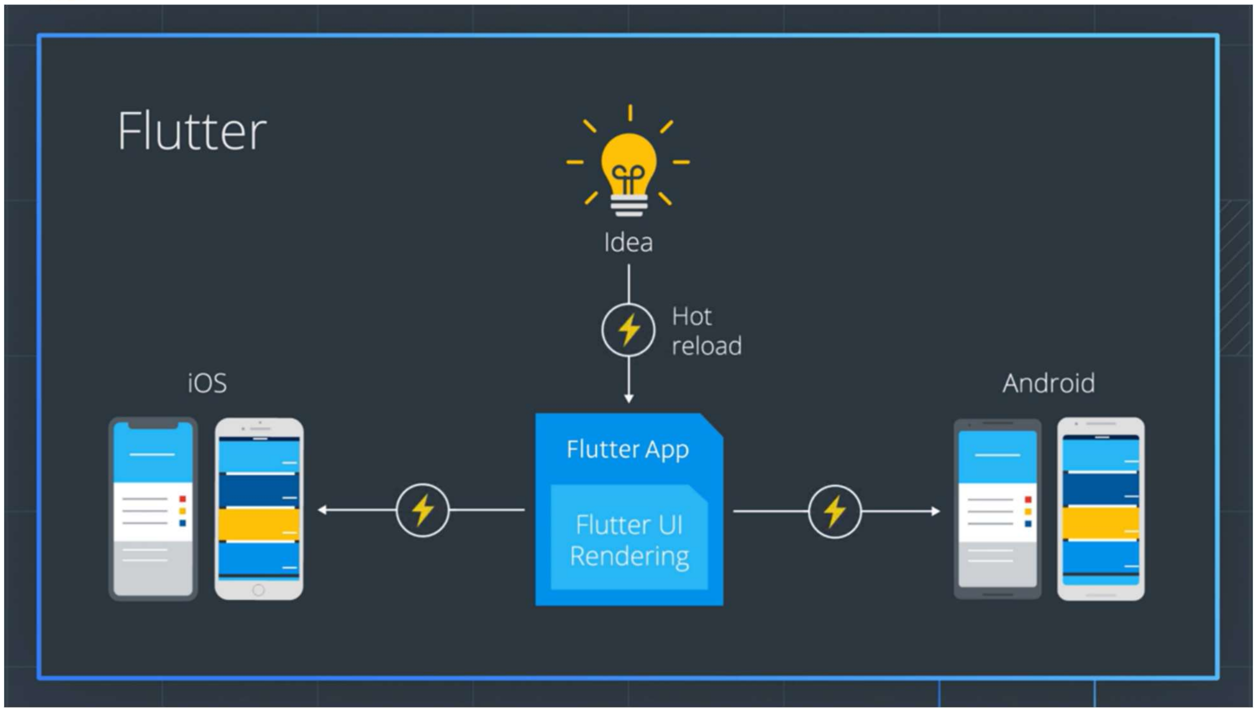
Why Flutter?

Generally, Users expect their apps to have beautiful designs, smooth animations and great performance, but that's not enough to make a successful app.

To deliver an app that users love, developers need to create new features quickly without compromising on quality or performance. This is where Flutter comes in.

Flutter aims to deliver the best of mobile application development: performance and platform integrations of native mobile with high-velocity development and multiplatform reach of portable UI toolkits.

So, Flutter is built to help developers increase app quality, increased development velocity and reach more users.



Flutter is unique because one code base compiles to native arm code meaning the machine code for each platform. This enables a fast startup and predictable fast performance. It uses its own mobile first GPU accelerated renderer for fast consistent UI across platforms and devices. It draws all its own widgets with a layered extensible customizable framework. There is no bridging between the framework and the widgets. So, rendering is efficient, and animations are smoother.

Why Dart?

Flutter is written in Dart, a terse, strongly typed, object-oriented programming language.

Primarily, its performant, both during active development and in production. It supports both just-in-time and ahead-of-time compilation.

Just-in-time compilation enables Flutter to recompile code directly on the device while the app is running. This leads to a very fast development cycle and enables sub-second, stable, hot reloading of the app.

Ahead-of-time compilation means that the libraries and functions used by your app code are compiled directly to native ARM code. This is great for release builds as the resulting native code starts quickly and has predictable performance.

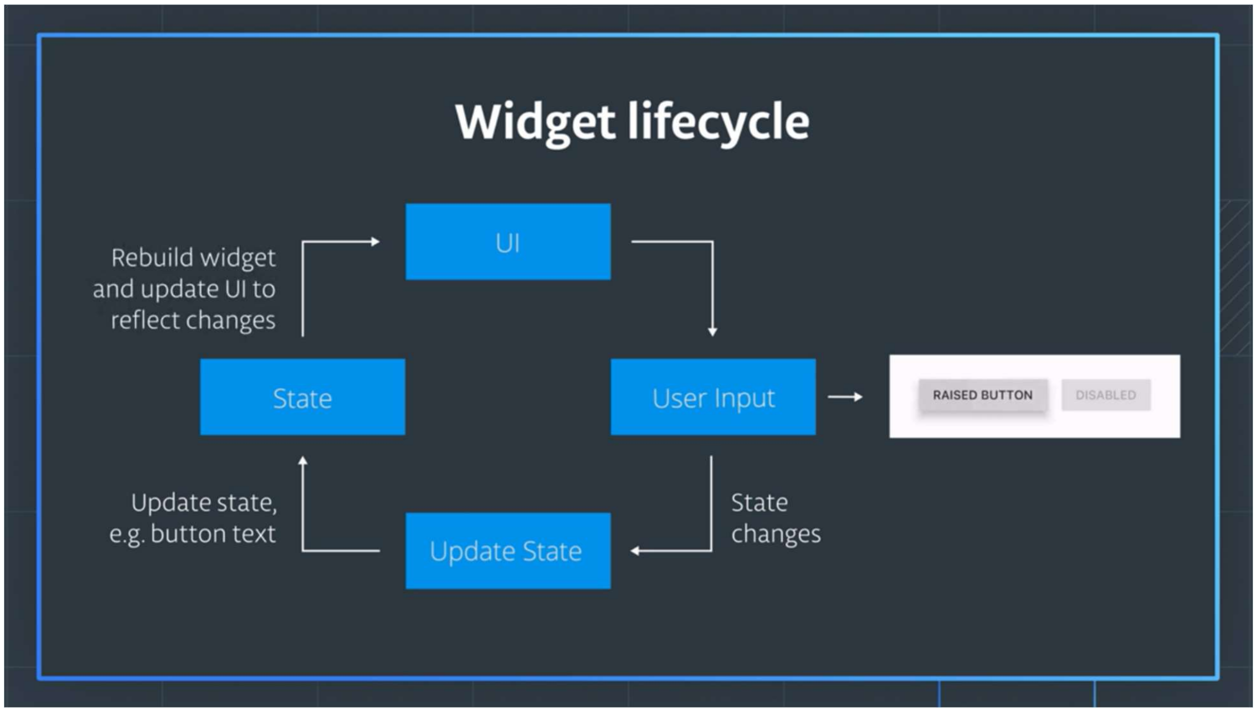
Dart also has static types and a sound type system which help you catch bugs at compile time and manage your code as it grows.

What is Flutter's Reactive Framework and Rendering Engine?

Flutter includes a modern reactive framework and a 2D rendering engine. With Flutter's reactive framework, a view is built as an immutable tree of widgets.

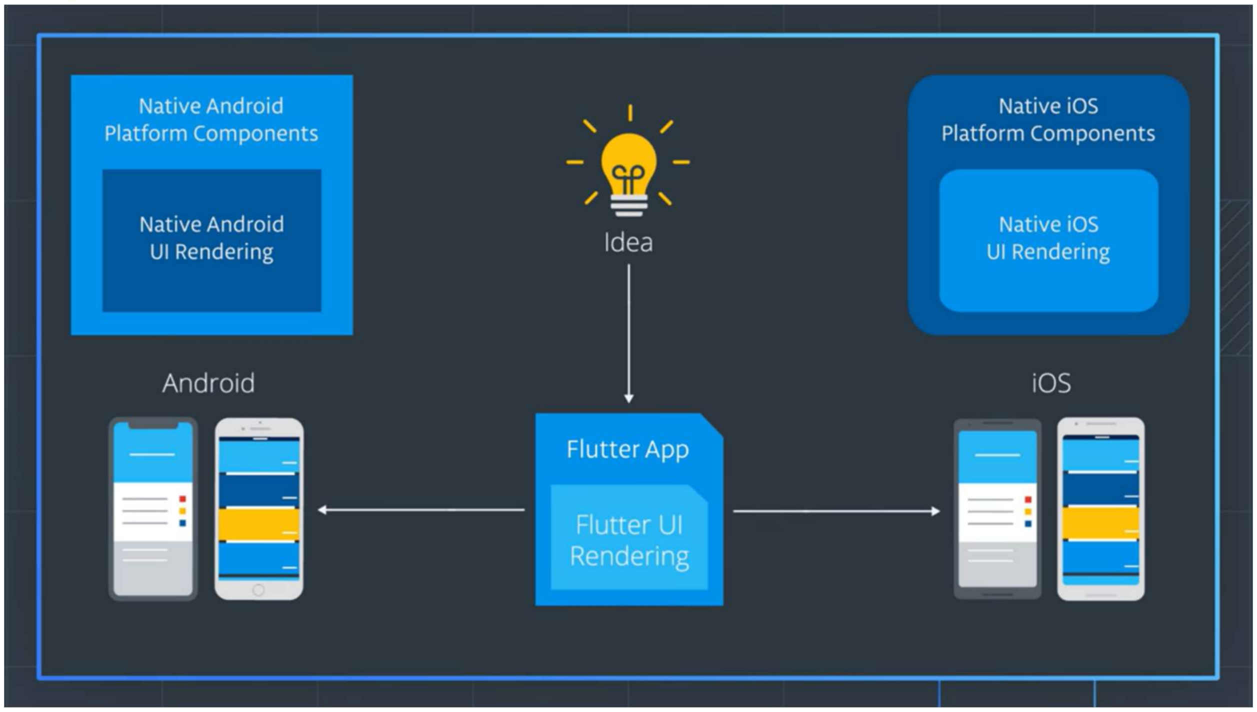
Widgets are the foundation of Flutter apps. A widget is a description of part of a user interface and nearly everything is a widget, written in Dart. There are no separate files for layout customization or text alignment. All the code for your widget is defined in one Flutter widget.

Widget lifecycle



When a widget state changes such as due to user input or animations, the widget rebuilds itself according to the new state. This saves developer's time because the UI can be described as functions of state. We don't have to write code for solely updating the UI when the state changes. The framework creates a difference to determine the minimal changes needed to update the render tree.

Notably, the **rendering engine** itself is part of your app, we don't need to bridge the UI rendering code to the native platform. Layout and rendering calls happen much more frequently than platform-specific calls such as those to the device camera. By doing all the rendering work on the app side Flutter can quickly render and re-render your widget tree allowing rich motion and smooth scrolling.


Native Android Platform Components
Native Android UI Rendering
Idea
Native iOS Platform Components
Native iOS UI Rendering
Android
iOS
Flutter App
Flutter UI Rendering

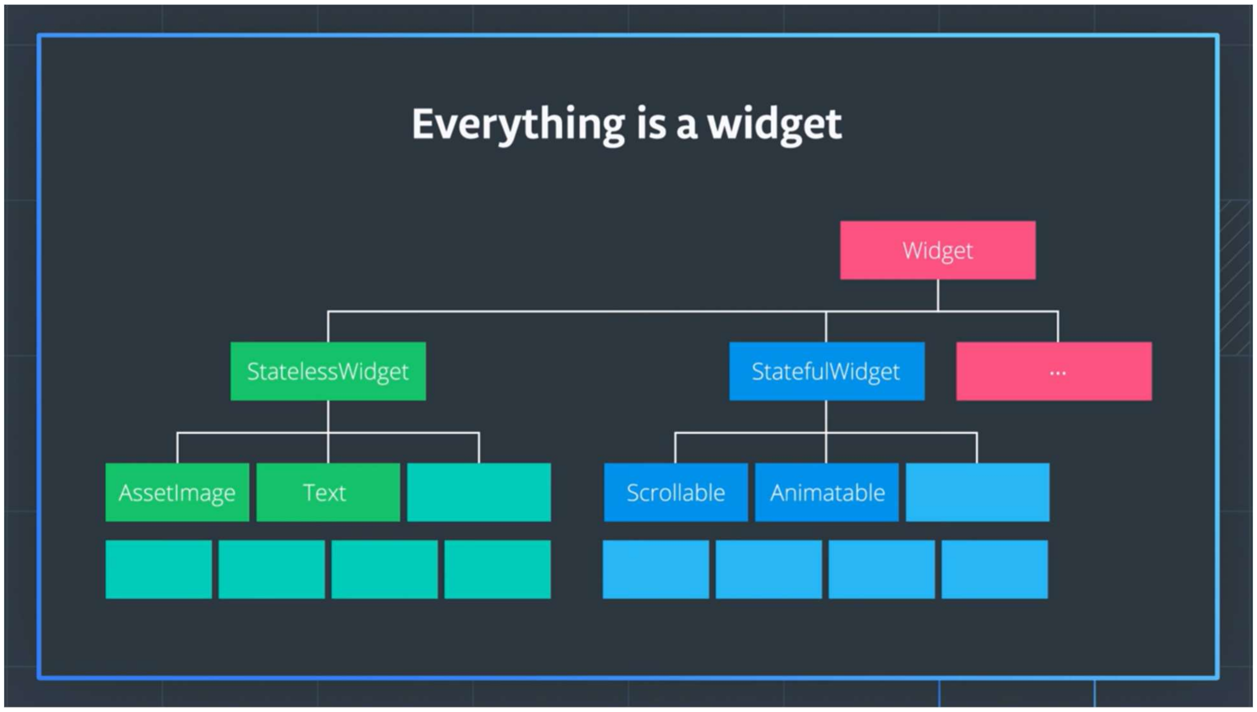
This rendering engine is built in **Skia**, a 2D graphics rendering library and Dart. It displays widgets for both iOS and Android devices. The iOS and Android platforms just have to provide a **canvas** for us to place our widgets on and rendering engine inside. This is where it's useful to have ahead of time compilation to native code.

What are widgets in Flutter?

Widgets are the foundation of Flutter apps. A widget is a description of part of a user interface.

Flutter would be nothing without widgets if Flutter UI is composed exclusively of them. For example, we can have a container widget inside of which is an image widget. Flutter comes with many premade, and customizable widgets like Container, Row, Column, Icon, IconButton, OffStage, Stack, Expanded etc. The package ecosystem also provides an ever-expanding collection of community made libraries and widgets.

We can customize our widgets, use widgets from other packages or just use the built-in ones. The built-in widgets include material components which make it straightforward to add themes and colors, layout the app and add user interaction.

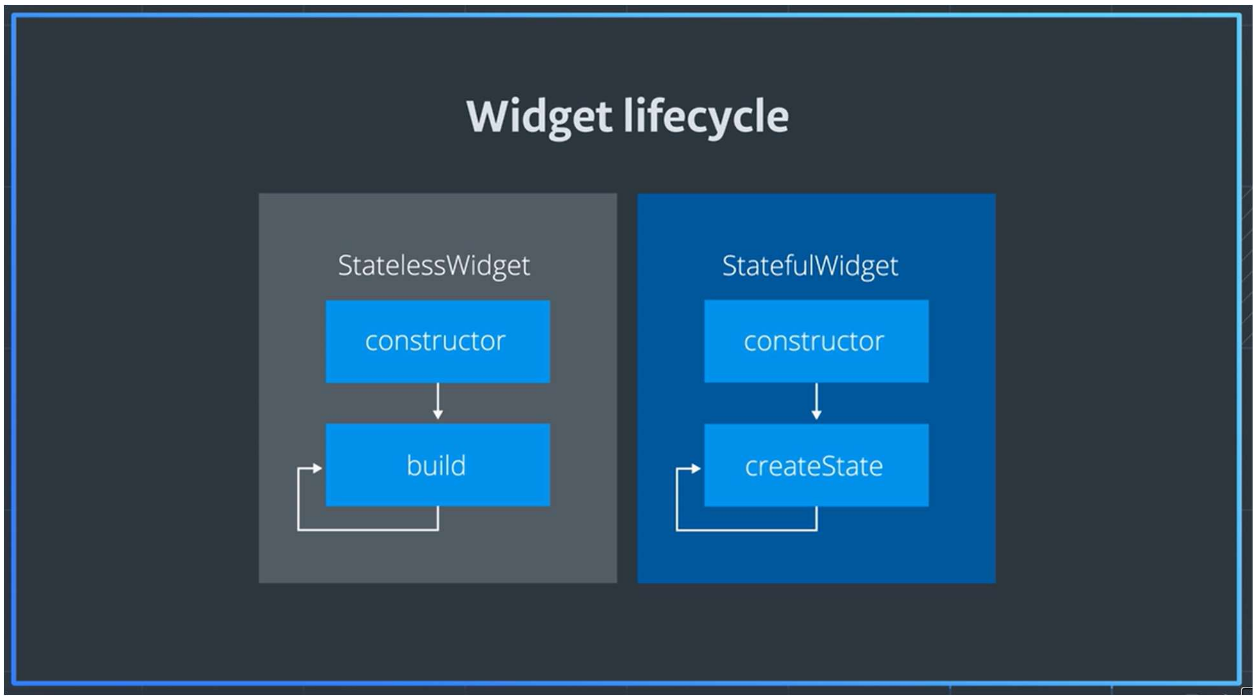


As mentioned, widgets are written entirely with Flutter code. There are both stateless and stateful widgets.

Stateless widgets are immutable. This means that all their fields have to be final when we initialize one at compile time. We can pass in custom properties when creating our widget such as a background color, but once created those properties can't change.

We **instantiate** each widget by parsing in some parameters. Some are required and some are optional.

Stateless widgets are not interactive. So, there are also a class of widgets called **stateful** widgets that create state objects means widgets can change, allowing for user interactivity.



To save state and respond to user interaction and events, we use a **Stateful** Widget. While the Stateful widget instance itself is also immutable, it creates **state object** using **createState**.

The **State Object** stores the widget state and can change throughout the widget lifetime. It contains a set state function that you call each time you want to change any state

Let's take a look at the **Container**, one of the most versatile widgets. You can use it to subdivide your app into sections, create various shapes, decorate existing widgets and more. We can customize our container by filling in the properties for example we can add width, height and color. We can also fill in property such as padding, margin and alignment.

Most widgets have an optional child or children property allowing us to nest widgets inside other widgets. The **child property** exists on widgets that only expect one child such as a padding or material app. The **children property** is present in widgets that take in a list of children such as columns, ListViews and stacks.

Filling in a child is also optional. We are allowed to create a widget without children, and this is the default behavior, so you don't have to specify that child equals null.

Flutter Application Overview

The entry point to a Flutter app is its **main** function.

To show something on the app screen, we import the Flutter **material package**, a package is a library of functions that you can use, and this library comes with pre-made widgets known as "**Material Components**". So, many building blocks of our app and many predefine colors are taken from the material package.

By importing this package, we are able to run our app using **runApp** function. runApp takes any widget as its argument.

Then the Scaffold is another one and that is usually created inside of the **Material app**. The scaffold offers drawers, app bars, bottom navigation, tabs and floating action buttons.

Stacks lay widgets on top of each other.

Bottom sheets or a tab that can be dragged opened to display its contents.

How can we retrieve text and update the state of your app?

Flutter widget often have a built-in property that lets you define what to do in response to a user's action. There are 3 ways to retrieve the text the user has entered:

1. onChanged
2. onSubmitted
3. Controller

The **onChanged** property takes in a function. Every time the text changes, this function is called. Inside the function, you can set state, check the input for validation errors, or count the number of characters. Whatever you want.

For **example**, if you're implementing a search field you might want to update the search results as the user types in their query presses the return or enter key on the keyboard. You can also pass in a function to do this. This is useful for text you want to evaluate only after the user indicates they have finished typing.

Finally, the **controller** property allows for more customization and control over the user input you create and pass in a text editing controller and add a listener that listens to this controller. This not only allows you to respond as the user types in each character, but also lets you modify or overwrite the contents of the text field.

You could use this to implement autocomplete, for **example**.

Finally, you should always be defensive against what a user can put inside your text box. Sure, the keyboard type property may have been set to number or email address, but it's always wise to add some **input validation**.

Flutter Gestures

Gestures represent semantic actions that are recognized from multiple individual pointer events. These can include **taps**, **drags** and **scales**.

Gestures can dispatch multiple events corresponding to the lifecycle of the gesture. If we want to add our own interactivity to any widget, we can wrap the widget in a **GestureDetector**.

File Assets

An **asset** is a file that is bundled and deployed with your app and is accessible and read-only at runtime. Our assets are stored in an assets directory that we create in our project directory. We define the assets in the projects **pubspec.yaml** file.

During a build, Flutter places assets into a special archive called the **Asset Bundle**, which apps can read from at runtime. Flutter comes with the **root bundle** object to access the main asset bundle. You can call this directly.

However, it is recommend retrieving the asset bundle for the current bill context using **DefaultAssetBundle**. This allows a parent widget to substitute in different asset bundles in runtime for localization and for running tests.

We use **DefaultAssetBundle.off** to indirectly load an asset. For **example**, a JSON file from the app's runtime root bundle.

What is Flutter?

- Flutter is a UI toolkit for creating fast, beautiful, natively compiled mobile applications with one programming language and a single codebase.
- It is an open-source development framework developed by Google.
- Flutter is not a language; it is an SDK.
- Flutter apps use Dart programming language for creating an app.
- 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.
- The first alpha version of Flutter was released in May 2017.

What is Dart?

- Dart is a general-purpose, object-oriented programming language with C-style syntax.
- It is open-source and developed by Google in 2011.
- The purpose of Dart programming language is to create a frontend user interfaces for the web and mobile apps.
- It is an important language for creating Flutter apps.
- The Dart language can be compiled both AOT (Ahead-of-Time) and JIT (Just-in-Time).

What do you understand by the Stateful and Stateless widgets?

Stateful Widget

- A Stateful widget has state information. It is referred to as Dynamic because it can change the inner data during the widget lifetime.
- A widget that allows us to refresh the screen is called a Stateful widget.
- This widget does not have a build ( ) method. It has createState ( ) method, which returns a class that extends the Flutters State Class.
- The examples of the Stateful widget are Checkbox, Radio, Slider, InkWell, Form and TextField.

Stateless Widget

- The Stateless widget does not have any state information. It remains static throughout its lifecycle.
- The examples of the Stateless widget are Text, Row, Column, Container etc.
- If the screen or widget contains static content, it should be a Stateless widget, but if you want to change the content, it needs to be a Stateful widget.

What is pubspec.yaml file?

It is project’s configuration file that will use a lot during working with the Flutter project. It allows how your application works. It is also allowing us to set the constraints for the app.

This file contains:

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

What are the packages and plugins in Flutter?

- A package is a group of similar types of classes, interfaces and sub-packages.
- The packages and plugins help us to build the app without having to develop everything from packages.

- In Flutter, it allows you to import new widgets or functionality into the app.
- The packages and plugins have a very small distinction.
- Generally, packages are the new components, or the code written in dart languages, whereas plugins allow more functionality on the device by using the native code.
- In the DartPub, packages and plugins are both referred to as packages.

What are the advantages of Flutter?

Cross-platform Development

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.

Faster Development

The performance of the Flutter application is fast. Flutter compiles the application by using the arm C/C++ library that makes it closes to machine code and gives the app a better native performance.

Good Community

Flutter has good community support where the developers can ask the issues and get the result quickly.

Live & Hot Reloading

It makes the app development process extremely fast. This feature allows us to change or update the code are reflected as soon as the alternations are made.

Minimal Code

Flutter app is developed by Dart programming language, which uses JIT and 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.

UI Focused

It has an excellent user interface because it uses a design-centric widget, high-development tools, advanced APIs and many more features.

Documentation

Flutter has very good documentation support. It is organized and more informative. We can get everything that we want to be written in one place.

Why does the first Flutter app build take so long?

When you build the Flutter app the first time, it will take a longer time. It is because the Flutter built the device specific APK or IPA file. Thus, the Gradle and XCode are used to build the file, taking a long time.

What is Tween Animation?

- It is the short form of in-betweening.
- In a tween animation, it is required to define the start and endpoint of animation. It means the animation begins with the start value, then goes through a series of intermediate values and finally reached the end value.
- It also provides the timeline and curve, which defines the time and speed of the transition.
- The widget framework provides a calculation of how to transition from the start and endpoint.

Explain Hot Reload in Flutter?

The hot reload feature allows you to quickly and easily perform an experiment in the project. It helps to build UI, add new features, fix bugs, and make app development fast. To perform hot reloading of Flutter app, do the following steps:

- Run the app in a supported Flutter editor or terminal window.
- Modify any of the Dart files in the project.
- If you use an IDE that supports Flutter, then select Save All or click the Hot Reload button on the toolbar. Immediately, you can see the result in your emulator or real device.

Name the popular database package used in the Flutter?

The most used and popular database packages used in the Flutter are as follows:

- Sqflite database: It allows to access and manipulate SQLite database.
- Firebase database: It will enable you to access and manipulate the cloud database.

Which type of animation allows you to represent real-world behavior?

The Physics-based animation allows you to represent real-world behavior in Flutter.

What is the difference between “main ( )” and “runApp ( )” functions in Flutter?

We can differentiate the main and runApp functions in Flutter as below:

- The main ( ) function is responsible for starting the program. Without the main ( ) function, we cannot write any program on Flutter.
- The runApp ( ) function is responsible for returning the widgets that are attached to the screen as a root of the widget tree and will be rendered on the screen.

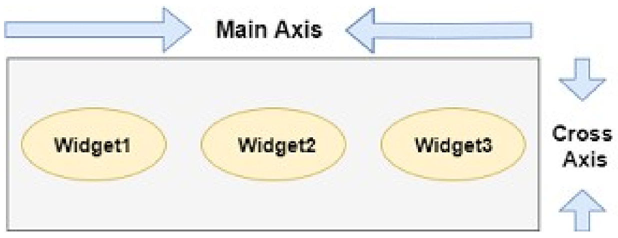
What is the difference between Hot Restart and Hot Reload?

Hot Reload	Hot Restart
It works with a small r key on the terminal or commands prompt.	It mainly works with State's value.
The hot reload feature allows us to quickly compile the newly added code in the file and sent them to Dart Virtual Machine (DVM). After DVM completes the updation, it immediately updates the UI of the app.	It allows developers to get a fully compiled application because it destroys the preserves State values and sets them to their defaults. On every Hot Restart, our app widget tree is completely rebuilt with the new typed code.
It helps to build UI, add new features, fix bugs and make app development fast.	It takes more time than Hot Reload to compile and update the app.

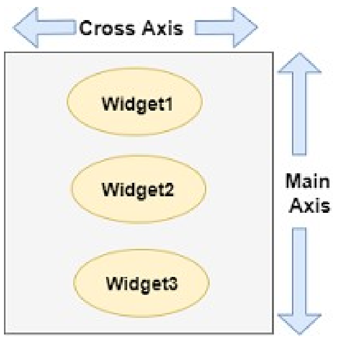
When should you use mainAxisAlignment and crossAxisAlignment?

We can use the crossAxisAlignment and mainAxisAlignment to control how a row and column widgets align its children based on our choice.

The row's cross-axis will run vertically, and then main axis will run horizontally.



The column's cross-axis will run horizontally, and the main axis will run vertically.



What is the difference between SizedBox Vs Container?

The Container in Flutter is a parent widget that can contain multiple child widgets and manage them efficiently through width, height, padding, background color etc. If we have a widget that needs some background styling may be a color, shape or size constraints, we may wrap it in a container widget.

The SizedBox widget in Flutter is a box that comes with a specified size. Unlike Container, it does not allow us to set color or decoration for the widget. We can only use it for sizing the widget passed as a child. It means it forces its child widget to have a specific width or height.

What is Stream in Flutter?

- A Stream is a sequence of asynchronous events. It provides an asynchronous sequence of data.
- It is the same as a pipe where we put some value on the one end, and if we have a listener on the other end, it will receive that value.
- We can keep multiple listeners in a stream, and all of those will receive the same value when put in the pipeline.
- We can process a stream by using the await for or listen ( ) from the Stream API. It has a way to respond to errors. We can create streams in many ways, but they can be used in the same way.

Example:

```
Future<int> sumStream(Stream<int> stream) async {
  var sum = 0;
  await for (var value in stream) {
   sum = sum + value;
  }
  return sum;
 }
```

Explain the different types of Streams?

Streams can be of two types, which are:

Single Subscription Streams

- It is the most common type of stream that contains a sequence of events, which is the parts of a larger whole. It will deliver the events in the correct order and without missing any of them.
- If any of the events are missing, then the rest of the stream makes no sense. This stream is mainly used to read a file or receive a web request. It will listen once, and if it is listening again, it means missing an initial event.

- When it starts listening, the data will be fetched and provided in chunks.

Broadcast Streams

It is a type of stream used for individual messages that can be handled one at a time without the knowledge of the previous events. It can have multiple listeners to listen simultaneously, and we can listen again after cancelling the previous subscription. This mouse events in a browser is a kind of this stream.

Why is the build ( ) method on State and not StatefulWidgets?

The main reason behind this is that the StatefulWidget uses a separate State class without building a method inside its body. It means all fields inside a Widget are immutable and includes all its sub-classes.

On the other hand, the StatelessWidget has its build and associated methods inside its body. It is due to the nature of StatelessWidget, which is rendered completely on the screen using the provided info. It also doesn't allow any future changes in its state information.

The StatefulWidget allows us to change the State information during the course of the app. Therefore, it is not suitable for storage in a build method to satisfy Widget class conditions where all fields are immutable. This is the main reason to introduce the State class. Here, we only need to override the createState ( ) function to attach the defined State with the StatefulWidget, and then all expected changes happen in a separate class.

What are the different build modes in Flutter?

The Flutter tooling supports three modes while compiling the application. These compilation modes can be chosen by depending on where we are in the development cycle. The name of the modes is:

- Debug
- Profile
- Release

Explain the difference between "??" and "?" operators?

??operator	? operator
The "??" operator is used to evaluate and returns the value between two expressions.	The "?" operator is used to evaluate and returns the value between two expressions based on the given condition.
It can be used as below: expr1 ?? expr2	It can be used as below: Condition ? expr1 : expr2

This operator first checks the expression 1 and, if it is non-null, returns its value; otherwise, it will evaluate and returns the value of expression 2.	This operator first checks the condition, and if it is true, it will evaluate expr1 and returns its value 9if the condition is matched). Otherwise, it evaluates and returns the value of expr2.

Why do we need mixins?

Dart does not support multiple inheritances. Thus, to implement the multiple inheritances in Flutter/Dart, we need mixins provide a way to write the reusable class's code in multiple class hierarchies.

Why we use a Ticker in Flutter?

- Ticker in Flutter is a refresh rate of our animation. It is a class that sends a signal at a regular interval, i.e., around 60 times per second.
- We can understand it with our watch, which tics at regular intervals. At each tick, Ticker provides a callback method with the duration since the first ticks at each second, after it was started.
- Even if the tickers started at different times, it always synchronized automatically.

What are keys in Flutter, and when to use it?

- Keys in Flutter are used as an identifier for Widgets, Elements and Semantics Nodes.
- We can use it when a new widget tries to update an existing element; then, its key should be at the same as the current widget key associated with the element.
- Keys should not be different amongst the Elements within the same parent.
- The subclasses of Key must be a Global key or Local Key.
- Keys are useful when we try to manipulate (such as adding, removing or reordering) a collection of widgets of the same type that hold some state.

How would you execute code only in debug mode?

To execute the code only in debug mode, we need to first import the dart foundation as below:

```
import 'package:flutter/foundation.dart' as Foundation;
```

Next, we need to use the kReleaseMode as below:

```
if (Foundation.kReleaseMode){ // is Release Mode ??

  print('release mode');
} else {

  print('debug mode');

}
```

What is profile mode, and when do you use it?

Profile mode is used to measure the performance of our applications. In this mode, some debugging ability is maintained to profile your app's performance. This mode is disabled on the emulator and simulator because they are not representative or real performance.

We can use the below command to compile the profile mode:

```
flutter run --profile
```

What is release mode, and when do you use it?

Release mode allows us to optimize the codes and generate them without any debug data in a fully optimized form. In this mode, many of the application's code will be entirely removed or rewritten.

We use this mode when we are ready to release the app. It enables maximum optimization and minimal footprint size of the application.

We can use the below command to compile the release mode:

```
flutter run --release
```

What is BuildContext?

- BuildContext in Flutter is the part of the widgets in the Element tree so that each widget has its own BuildContext.
- We mainly use it to get a reference to another widget or theme. For example, if we want to use a material design element, it is required to reference it to the scaffold.
- We can get it using the Scaffold.of (context) method.

What is the difference between WidgetsApp and MaterialApp?

WidgetsApp	MaterialApp
WidgetsApp is used for basic navigation. It includes many foundational widgets together with the widgets library that Flutter uses to create the UI of our app.	MaterialApp, along with the material library, is a layer that is built on the top of WidgetsApp and its library. It implements Material Design that provides a unified look and feels to our app on any platform.
WidgetsApp class is the base class for MaterialApp class.	It offers many interesting tools such as Navigator or Theme for developing the application.
It wraps several widgets that are required for building the application.	It wraps several widgets that are required for building material design applications.

What types of tests can you perform in Flutter?

Testing is an activity used to verify and validate the application, which is bug-free and meets the user requirements. Generally, we can use these three types of tests in Flutter:

Unit Tests

It tests a single function, method or class. Its goal is to ensure the correctness of code under a variety of conditions. This testing is used for checking the validity of our business logic.

Widget Tests

It tests a single widget. Its goal is to ensure that the widget's UI looks and interests with other widgets as expected.

Integration Tests

- It validates a complete app or a large part of the app. Its goal is to ensure that all the widgets and services work together as expected.
- Flutter also provides one additional testing known as a golden test. Its goal is to ensure that you have an image of a widget or screen and check to see whether the actual widget matches it or not.

What are Null-aware operators?

Dart provides some useful information to handle the null values.

1. The "??=" assignment operator that assigns a value to a variable only when that variable is null.

```
int a; // Initial value of a is null.
a ??= 5;
print(a); // It will print 5.
```

2. The "??" null-aware operator that is used to evaluate and returns the value between two expression. It first checks the expression 1 and if it is non-null, returns its value; otherwise, it will evaluate and returns the value of expression 2:

```
print(3 ?? 5); // It will print 3.
print(null ?? 5); // It will print 5.
```

Why is the Android and iOS folder in the Flutter project?

Android: This folder holds a complete Android project. It is used when you create 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 is further deployed to the Android Virtual Device.

iOS: This folder holds a complete Mac project. It is used when you build the Flutter application for iOS. It is similar to the Android folder, which 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 and Xcode IDE.