FLUTTER AND DART THE COMPLETE GUIDE

FREDRICK MARTINS

Contents

Introduction

Some years ago, Google unveiled Dart, a general-purpose programming language. Version 1.0, released in late 2013, was meant as a replacement for JavaScript in browsers; it also had its own virtual machine (VM) for running native applications in the Chrome browser.

Dart allows Flutter to update the display at 60 or even 120 frames per second (if the hardware supports it), which are frame rates more typical of games. Flutter

has adopted a similar architecture style to React Native, except that compiled Dart code deals directly with pixels. It does this through its own widgets, not those supplied in iOS or Android.

There was a saying twenty years ago that “everything in Windows is a window,” and in Flutter, the same is true: “Everything is a widget.” For iOS, Flutter includes Cupertino, a package of widgets for iOS-look-alike controls.

Flutter Widgets are immutable and only exist until they need to be changed; then a new tree of widgets is created. As a result, Flutter is able to run at high frame rates consistently, and uses Dart’s fast garbage collection to make that happen.

What is Dart?

Dart is the programming language used to code Flutter apps. Dart is another product by Google and released version 2.1, before Flutter, in November. As it is starting out, the Flutter community is not as extensive as ReactNative, Ionic, or Xamarin.

Dart looks a bit like C and is an object-oriented programming language. So, if you prefer the C languages or Java, Dart is the one for you, and you’ll likely be proficient in it.

Dart is a programming language that was initially used internally at Google to build web, server and mobile applications

Dart is not only used for mobile app development but is a programming

language. Approved as a standard by Ecma (ECMA-408), it’s used to build just about anything on the web, servers, desktop and of course, mobile applications.

Dart, when used in web applications, is transpiled to JavaScript so it runs on all web browsers. The Dart installation comes with a VM as well to run the .dart files from a command-line interface. The Dart files used in Flutter apps are compiled and packaged into a binary file (.apk or .ipa) and uploaded to app stores.

What does coding in Dart look like?

Like most ALGOL languages (like C# or Java) :

The entry point of a Dart class is the main () method. This method acts as a starting point for Flutter apps as well.

The default value of most data types is null.

Dart classes only support single inheritance. There can be only one superclass for a particular class but it can have many implementations of Interfaces.

The flow control of certain statements, like if conditions, loops (for, while and do-while), switch-case, break and continue statements are the same.

Abstraction works in a similar manner, allowing abstract classes and interfaces.

Unlike them (and sometimes a bit like JavaScript) :

Dart has type inference. The data type of a variable need not be explicitly declared, as Dart will “infer ”what it is. In Java, a variable needs to have its type explicitly given during declaration. For example, String something; But in Dart, the keyword is used instead like so, var something;The code treats the variable according to whatever it contains, be it a number, string, bool or object.

All data types are objects, including numbers. So, if left uninitialized, their default value is not a 0 but is instead null.

A return type of a method is not required in the method signature.

The type num declares any numeric element, both real and integer.

The super() method call is only at the end of a subclass’s constructor.

The keyword new used before the constructor for object creation is optional.

Method signatures can include a default value to the parameters passed. So, if one is not included in the method call, the method uses the default values instead.

It has a new inbuilt data type called Runes, that deal with UTF-32 code points in a string. For a simple example, see emojis and similar icons.

And all these differences are just a few in the many that you can find in the Dart Language tour, which you can check out here.

Dart also has inbuilt libraries installed in the Dart SDK, the most commonly used being:

dart:core for core functionality; it is imported in all dart files.

dart:async for asynchronous programming.

dart:math for mathematical functions and constants.

dart:convert for converting between different data representations, like JSON to UTF-8.

Using Dart in Flutter

Flutter has more app-specific libraries, more often on user interface elements like:

- Widget: common app elements, like the Text or ListView.
- Material: containing elements following Material design, like FloatingActionButton.
- Cupertino: containing elements following current iOS designs, like CupertinoButton.

Fundamental Aspects of Dart that Make it Indispensable for Flutter.

a. Combines Ahead-of-Time and Just-in-Time Compilation

In creation, developers typically need to decide on a certain kind of compilation for their programming language. That is, whether the program is to be executed before or during runtime. Ahead-of-Time (AOT) compiled programs generally run quicker since they are compiled before execution. However, in reality, with ahead of time compilation, the development itself is decelerated.

Just-in-Time (JIT) compilation, on the other hand, helps quicken development processes but reduces the application's initialization speed, as the compiler executes the code at or just before runtime.

Overcoming these issues, Flutter uses JIT compilation for development and AOT for launching the application, thus ensuring an optimal end-user experience.

b. Eliminates XML Files

Dart has a declarative and programmable layout that is easy to read and visualize. Hence, Flutter doesn't require a separate declarative layout language like XML. It is easy for Flutter to provide advanced tooling since all the layout in one language and in a central place.

c. Eliminates the JavaScript Bridge

The application runs seamlessly on a user's gadget as Dart compiles and executes

directly into native code, without an intermediary bridge (e.g., JavaScript to Native). This allows Dart to render hitchless animations, while user interfaces are better even at 60fps. Dart can further perform object allocation and garbage collection without the acquisition of locks.

d. Scaffolding

The scaffold in Flutter is very distinct from that of iOS or React Native or even the Android UI. Among other things, it stretches to fill in the space available. Usually, this means it will fill the entire screen of its device window.

The scaffold incorporates APIs for an app bar, a floating button, drawers, and the bottom sheets to enforce the graphic interface arrangement of the primary content design.

e. Incorporates HTTP

To host HTTP resources from the internet, Dart incorporates an abstraction named Future. Here, the http.get() function returns a ‘Future’ that contains a ‘Response’. While the Future is a core class in Dart to deal with asynchronous functions, a future-object reflects a possible value or error and will be visible in the Future at any point.

The Upsides of the Flutter-Dart Combination for the Business

1. Portability

Dart provides a standalone VM that makes use of the language itself as its

intermediate language (literally serving as an interpreter and making it a smoother process to port any programming language to any new hardware platform).

As Flutter isn't just a platform but a full SDK, it can function seamlessly with only a screen on nearly every device. By now, third-party terminals for developing Flutter applications for Mac OS, Windows, and Linux have all been created. These terminals include embedding input functions for the APIs, mouse, keyboard, and various plugins.

2. Accessibility

An essential component of the Dart intl package, Flutter provides widgets that simplify accessibility through a broad user base. It recognizes 24 languages, currencies, units of measurement, date and time, interface options (for recognizing variations in language writing style), and so on. Flutter also guarantees web accessibility, while its UI enables:

- Large fonts
- Screen readers
- Sufficient contrast

Dart vs JavaScript Comparison

Ease of use

JavaScript has been around longer in the industry and is a mature and stable

language. JavaScript is very easy to use. It has numerous frameworks and libraries available online, so developers can use the existing code for developing apps faster. However, in order to learn the JavaScript syntax, we need to have a basic technical knowledge of programming in general.

Dart is a fairly new language for most of the developers outside of Google. Although Google has put a lot of effort into documenting the Dart programming language, it's still hard for developers to find solutions for specific problems. In terms of coding style and syntax, it has Java-like syntax, so developers from OOPS background can master and use Dart easily once they learn the basics.

Popularity

At the moment, JavaScript is everywhere. There is no device in the world that doesn't run JavaScript. There are many companies that are using JavaScript frameworks for developing web and mobile applications. JavaScript can also be used for writing server-side applications and backends, so more and more developers have got hooked on JavaScript as a language during their career.

Before Google announced Flutter, Dart was nowhere to be found. However, since the announcement of Flutter in Google I/O, Dart has got drastic attention among the mobile developers as an alternative to React Native. The developers who didn't like JavaScript as the programming language found Dart as an alternative option. As big companies like Google, , etc adopted Flutter, the popularity of Dart raised considerably, but it's not near as popular as JavaScript.

Productivity

JavaScript has countless frameworks and new JavaScript frameworks land in the market almost every year. As there is a need for developers to share and distribute code, there are thousands of JS packages available online. With the use of an existing package and experience of the developers, it's easy for new developers to learn and adopt the JavaScript programming language. As

JavaScript is a fast, lightweight and dynamic programming language, it boosts the developer productivity. Solutions to common problems can be found online easily, which is another reason that developers prefer JavaScript over other programming languages. Although JavaScript has countless libraries and frameworks available, not all of them are equally good. Also new JavaScript frameworks are released after a regular interval, so the developers constantly need to learn new frameworks, which hinders their productivity.

Dart has great, but new or even experienced developers can get confused with the Dart language features. The Dart syntax is clean and mostly looks similar to Java, so developers with Java background can pick up the code syntax easily. However, developers may struggle a lot to find solutions to the problems in Dart.

Learning curve

Learning JavaScript is not an easy task for non-programmers, but knowing the basic programming concepts makes JavaScript easy to learn. There are lots of online course and tutorials out there for the developers who wish to learn JavaScript.

Learning Dart can be daunting task for beginners as Dart is not a commonly used programming language and there are very limited courses or books available online for the Dart programming language. However, Google has extensive documentation of Dart on its official website, which helps programmers from other OOPS languages to pick up the Dart concepts easily. Some of its syntax might be unusual and come as a surprise, but with little bit of practice, developers will soon feel comfortable with Dart.

Speed

JavaScript is an interpreted language, so it might feel lighter and faster. It’s

actually faster than other compiled languages like. However, Dart proved to be much faster when benchmarked against JavaScript. You can refer to the benchmark of Dart against NodeJS .

Dart can be compiled both, and which helps building apps in several ways as using JIT compilation can speed up development and AOT compilation can be used during the release process for better optimization. This technique has been used in Flutter app development.

Frontend vs Backend

JavaScript was originally used for frontend web development with HTML and CSS. However, with the rise of the framework, JavaScript is now widely used for server-side and backend development as well.

Dart is currently actively used with Flutter for developing the frontend of cross-platform mobile apps. Dart can be also used for , but there is no mention of Dart being used for backend development.

Type safety

Being an interpreted language, JavaScript supports both dynamic and the application developer can type any code and JavaScript allows it, so JavaScript is not a type-safe language. Programming errors can only be found at the runtime.

Dart supports both loose and strong prototyping. As Dart is a compiled language, most of the programming errors can be found during the compilation. In that respect, Dart is more type-safe than JavaScript.

Web vs Mobile

JavaScript has dominated both mobile and web app development with different frameworks. At one point, with the release of React and React Native from Facebook, JavaScript become a no-brainer for developing web and mobile apps for startups and small businesses. There is a large number of JavaScript frameworks still available in the market for developing web apps, progressive web apps and hybrid mobile apps, etc.

Similarly to JavaScript, Dart can be used for both mobile and web development. Dart became popular along with the framework for developing cross-platform mobile apps. Dart can be also used for developing , but is actually used for this purpose very rarely. There are some and for developing web apps with Dart for those who would like to give it a try. Nevertheless, as Flutter's popularity is growing fast, developers are starting to pay attention to Dart. There are also new Dart landing in the market. In the end, the future of Dart depends on the success of Flutter.

Editor/IDE support

There are some great IDEs and editors out there for JavaScript development. JavaScript doesn't always require a full-fledged IDE as developers can use lightweight editors. However, there are some IDEs like WebStorm and Visual Studio Code that can work well for JavaScript application development.

Dart code can be also developed with lightweight editors like Sublime or VIM, but there are and that come with the Dart plugin and are the most commonly used IDEs for Dart application development, especially for Flutter app development.

Commercial use

JavaScript is extensively used in big companies for developing both web and cross-platform mobile apps. Facebook is the pioneer in writing the React and the React Native framework that they used internally, but there are many other big companies that use JavaScript, including Instagram, Reddit, eBay, Slack, Airbnb, etc.

Dart was born in Google, so obviously it's being used inside Google, but there are some big brands like Alibaba that also adopted Flutter and Dart for developing cross-platform mobile apps. There are some other big companies that use Dart, including Blossom, WorkTrails, Whale, Mobile, etc.

Dart vs JavaScript: Pros and cons

Pros of Dart

- Open-source
- Backed by Google and runs easily on Google Cloud Platform
- Dart is approximately two times faster than JavaScript
- Dart is type-safe and compiled with both AOT and JIT compilers

Cons of Dart

- Dart is fairly new to the programmers and rarely used in the market.
- Dart has very limited resources online and it's hard to find solutions to problems.

Pros of JavaScript

JavaScript can be used for both web and mobile apps.

It can be used for both frontend and backend, so JavaScript can run on every device.

JavaScript has a huge community and great frameworks available online.

JavaScript is friendly with other languages, so many other apps can use JavaScript.

Cons of JavaScript

- JavaScript has some libraries that are not of good quality.
- Being a dynamic language, programmers can make big mistakes easily.
- There are constant changes as a new framework lands after a regular interval.

What is Flutter?

Flutter is a free and open-source mobile UI framework created by Google and released in May 2017. In a few words, it allows you to create a native mobile application with only one codebase. This means that you can use one programming language and one codebase to create two different apps (for iOS and Android).

Flutter consists of two important parts:

An SDK (Software Development Kit): A collection of tools that are going to help you develop your applications. This includes tools to compile your code into native machine code (code for iOS and Android).

A Framework (UI Library based on widgets): A collection of reusable UI elements (buttons, text inputs, sliders, and so on) that you can personalize for your own needs.

To develop with Flutter, you will use a programming language called Dart. The language was created by Google in October 2011, but it has improved a lot over these past years.

Dart focuses on front-end development, and you can use it to create mobile and web applications.

If you know a bit of programming, Dart is a typed object programming

language. You can compare Dart's syntax to JavaScript.

“Flutter is Google’s UI toolkit for building beautiful, natively compiled applications for mobile, web, and desktop from a single codebase.” - Google, flutter.dev

How Flutter Works: Widget Strategy and Dart Programming Language

The framework of Flutter, written in the Dart programming language, has the Flutter engine, Foundation library, and widgets. The approach to development in Flutter differs from others by its declarative UI writing. Here, there is a need to start from the end, meaning before starting the development of some element, the user needs to have in mind a complete picture of what kind of UI it will be. Many developers distinguish this UI writing as a more clear one, but it also causes certain difficulties for developers at first.

The main idea of Flutter is that developers can build the entire user interface by simply combining different widgets. The application interface consists of various nested widgets, which can be any object. This applies to anything from buttons to padding, and by combining widgets, the developer can customize the application radically. Widgets can influence each other and use built-in functions to respond to external changes in the state. Widgets are important elements of the user interface and comply with the design specifications of Android, iOS and conventional web applications.

With Flutter, developers can create custom widgets, which can be easily combined with existing ones. Note that there are no OEM widgets, but Flutter Gallery provides developers with their own ready-made widgets — a set of application examples that show how to use standard widgets — which look like

native Android and iOS design languages (Material and Cupertino).

Flutter also provides developers the ability to view widgets in a reactive style. For the record, Flutter isn't the first one to do this but Flutter is the only mobile SDK that offers a reactive look without the need for a JavaScript bridge. Moreover, Dart comes with a repository of software packages to enhance the capabilities of applications. For example, it offers several packages that help to access Firebase so that developers can create serverless applications. Another package allows access to the Redux data warehouse or facilitates access to platform services and equipment such as cameras.

Why Choose Flutter on Your Next Project

There are things that are of great value to the business — platform stability, its performance, a guarantee of successful support, and improvement of technologies and products. Any problems and shortcomings in any of these aspects may lead to risks, including direct and indirect financial losses.

With this in mind, Flutter lowers risks for your business due to the following points:

The best in class (cross-platform) performance and resource consumption due to the compilation of native code and high-performance rendering engine. The first provides an easy way of establishing communication between platform-native code and Dart through platform channels. Thus, developers can implement into a Flutter app anything that a native app can do, just with a little more effort on the native side. Because of the engine (Flutter uses Skia for rendering itself), a UI built in Flutter can be launched on virtually any platform, assuming this platform supports Flutter. Putting it differently, developers no longer have to adjust the UI

to transfer it to a platform, which simplifies the development process vastly.

A good UX is incredibly dependent on the performance of the application. Flutter application performance in most cases will be indistinguishable from the native application. This is because Flutter doesn’t rely on intermediate code representations or interpretation. Application on Flutter is compiled directly into machine code, eliminating any performance errors in the interpretation process. This provides it with the highest performance and makes it the most resource saving software among cross-platform technologies.

Better developer productivity is achieved due to Flutter being primarily designed for quicker code writing. It consists of ready-to-use widgets, its syntax requires less code to be written, and hot reload speeds increase the searching for and correction of bugs. All this results in fewer man-hours for developers.

Also, finding Flutter engineers in 2020 is not a problem — the community of fans is growing, especially among Android developers. According to Stack Overflow Developer Survey 2020, 68.8% of developers — which is among the top 3 — love to use Flutter and have expressed interest in continuing to develop with it.

Quicker time to market. Due to the greater productivity of Flutter developers, it takes less time to create an application, which means that compared to other programming languages and frameworks, applications in Flutter are written quicker and enter the market earlier with equal effort. Thus, the less coding and support effort needed, the quicker the time to market.

Low-cost app development. Flutter provides more efficient development work and, accordingly, to develop an application requires less man-hours. At the same time, the cost of an hour is at the average market (and sometimes even lower) level. As a result, the cost of the application on Flutter is lower than when using other cross-platform languages or native development.

The Flutter App

Building the user interface of a Flutter app makes use of Widgets.

Widgets work in a similar way to React. A widget uses different components to describe what the UI should look like. They can be either Stateful or Stateless. In Stateful components, the widget rebuilds due to state changes, to accommodate the new state.

When we look at the current code for the Home page, we see that it's a Stateful page. If the counter variable increases, the framework tries to find the least expensive way to re-render the page. In this case, find the minimal difference between the current widget description and the future one. It takes into account the changed state.

The Scaffold class is a material design layout structure and is the main container for the Home page. The AppBar, also a material design element is the title bar found at the top of the page. All other components, like the floating button and two text tags, fall under the body of the page. The Center class is a layout class that centers its child components vertically and horizontally.

The Column class, another layout widget, lists each child element vertically.

Each of its child elements is added to an array and put underneath the children: section.

The two texts speak for themselves. The first displays the text ‘You have pushed.’ The second one displays the current value in the _counter variable.

The FloatingActionButton is part of the Material design widgets. It displays a + icon and triggers the increment of the _counter variable.

Apps Built with Flutter

Although Flutter is young enough, it is already used by such global services as Google Ads, Alibaba, AppTree, Reflectly and many others.

“Flutter significantly reduced the time we need to develop new features from 1 month down to 2 weeks.” says Bruce Chen, Senior Development Engineer, Alibaba.

Flutter has also been picked as the solution for Nubank for all app development. Nubank is the largest digital bank in the world outside of Asia. With over 20 million customers, they could easily see the benefits of unified app development.

As for the question, “What apps can be made with Flutter?”, the answer is quick and simple — any. You can develop applications for any purpose — from small business (online stores, banks) to large ones (contact centers, courier control, organization of internal processes). However, Flutter is the best option for start-ups and R&D projects, primarily because of the ability to quickly prototype and check business ideas.

Setting up Flutter

So, to get this thing into gear, follow the Flutter docs. It gives details on installing the Flutter SDK and setting up your preferred IDE; mine would be VS code. Setting up VS code with the Flutter extension is helpful. It comes with inbuilt commands, as opposed to using the terminal.

Follow the docs again to create your first app. In my case, run the extension command Flutter: New Project. Afterward, type the project name and pick the destination folder.

If you prefer using the terminal, move to the destination folder of the app. Then use the command flutter create <app_name> to create the app folder. This generates the entire app folder, including the Android and iOS project folder. To open these folders, use Android Studio and XCode, for building the app.

In the root of the project, you find pubspec.yaml. This file contains the app's dependencies. This includes both external libraries/modules and assets like images and config files. It works like a package.json, containing all external modules of the app. To install these packages, enter the package name and version under the dependencies: section of the pubspec.yaml. Run the command flutter packages get. Include the assets of the app inside the flutter: section of the same file.

The entry point of the app is main.dart, found inside the lib folder. This folder also contains all Dart classes (app pages or reusable components). On creation of the app, the main.dart file comes with a simple pre-written code. Before running this code, a device is either connected to the PC, with USB debugging enabled. Afterward, run the command flutter run on the terminal.

Flutter vs. React Native: In a nutshell

What is it? A portable UI toolkit for building natively-compiled apps across mobile, web, and desktop* from a single codebase A framework for building native applications using React

Official release	December 2018, Google I/O
	March 2015, F8 Conference
Created by	
Google	Facebook
Free and open source	
Yes	Yes
Programming language	

Dart	JavaScript
Popularity	81,200 Stars on Github (December 2019)
	83,200 stars on Github (December 2019)
Hot Reload	
Yes	Yes
Native performance	
Great	Great
UI	

Flutter apps look as good on the up-to-date operating systems as they do on older versions.

Since they only have one codebase, the apps look and behave similarly across iOS and Android – but thanks to Material Design and Cupertino widgets, they can also imitate the platform design itself. How’s that possible?

Flutter contains two sets of widgets which conform to specific design languages: Material Design widgets implement Google’s design language of the same name; Cupertino widgets imitate Apple’s iOS design.

This means that your Flutter app will look and behave naturally on each platform, imitating their native components.

Application components look just like native ones (e.g. a button on an iOS device looks just like a native iOS button, and the same on Android).

The fact React Native uses native components under the hood should give you confidence that, after any OS UI update, your app's components will be instantly upgraded as well.

That said, this can break the app's UI but it happens very rarely.

If you want your app to look near-identical across platforms – as well as on older versions of an operating system (as Flutter achieves) – then consider using third-party libraries (like this one). They will enable you to use Material Design components, in place of native ones.

Sharing code

Currently on iOS and Android – but the long-term vision for Flutter is to offer an integrated solution that allows developers to write one code for both desktop & mobile, and for the web.

Flutter for Web support is available as a tech preview but still, this isn't an alpha channel yet.

When it comes to developing desktop apps with Flutter, APIs are in their early

stages of development and so will be probably released, just further down the line.

iOS and Android – but there are select libraries that allow you to use the same code to build iOS, Android, web, and Windows10 apps.

You can also extract shared code in mobile, desktop, and web apps, to a separate repository; treat it as a separate project; then inject it in the same way as another dependency.

This allows a developer to focus on writing code for a specific platform without having to consider compatibility with another one.

Top apps made with this technology

Xianyu app by Alibaba, Hamilton app for Hamilton Musical, Google Ads app Instagram, Facebook, Facebook Ads, Skype, Tesla

Time-to-market

Typically much faster than native development.

Possibly as fast as development with Flutter.

However…

React Native uses bridge and native elements, so it may reqquire separate optimization for each platform – a problem that widget-based Flutter doesn't run into. It may make the app development with React Native longer.

Competitive advantage

I. Great look and feel thanks to rich widgets;
II. Rapidly growing community, and popularity;
III. Excellent documentation with strong support from the Flutter team (which makes it easy to start developing with Flutter);
IV. Improving Flutter for Web, offering the potential for one codebase across mobile and web platforms
V. Difficult to beat time-to-market length
VI. Stability (5+ years on the market);
VII. Many successful, prominent market players using React Native;
VIII. Mature, vast community;
IX. Easy-to-learn technology;

Plenty of tutorials and libraries, which allow quick and easy development;

Code can be easily reused for both web app and desktop app development.

When it is not the best fit

If…

• Your app needs to support 3D Touch (for now, Flutter doesn't support 3D – but it features on the Flutter team's long-term roadmap)

• The design of your app is platform-specific

• Your app requires multiple interactions with an OS; or requires rare, little-known native libraries

• You need a minimalistic UI, but rely on significant use of the phone hardware (e.g. an application that plays music, or only takes pictures)

• You want to create an instant app (small-sized app)

If your app sounds like any of the above, it's probably better you choose native app development.

If…

• Your app needs to handle less common, or ultra-specific tasks (like calculations) in the background

• You require custom communication via Bluetooth (which can be tricky to implement using React Native)

• You want to create an app for Android only

In truth, if you want to build an iOS app and you know JavaScript, consider React Native – but if you want an Android-only app, it's likely better to build natively with another team. Why? Right now, iOS has better support than Android.

If your app sounds like any of the above, it's probably better you consider choosing native app development.

Flutter’s Disadvantages

While Flutter has a lot of benefits that businesses can take advantage of, there are some areas in which it still needs work.

Lack of third-party libraries. Flutter is a newer technology. As such, the volume of third-party libraries currently available for Flutter is limited. Third-party libraries help speed up development time significantly, so this is a definite downside to developing in Flutter.

Large file size. Many, if not most, of the apps developed through Flutter are destined for mobile devices only. Although current mobile devices have large storage capacities, file size is still important. For example, the creation of a hello world app in Flutter could account for 4.7MB to 6.7MB. The same app created in native Java is closer to 500KB.

New skills required. While Flutter is easy to use and can be learned by non-programmers, it does require developers to learn Dart first. This adds an additional phase of learning, which can increase the time and money for any project. That being said, if a developer knows Java/C#, he or she can easily upskill to Dart. Moreover, Flutter’s Dart programming language is pretty easy to learn for those with little programming experience.

Altering the current code

When you click the button, the _counter variable value increases. This re-renders the page and the new value is displayed on the body of the page.

I’m going to change that up a bit. For every button click, we will display a custom Card component with the item number.

Creating the Custom Card Component

So, to start off, we make a new .dart file inside the lib folder. I created mine in a subfolder commonComponents and named it customCard.dart.

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

class CustomCard extends StatelessWidget {  CustomCard({@required this.index});

final index;


  @override

  Widget build(BuildContext context) {

return Card(

child: Column(

children: <Widget>[Text('Card $index')],

        )

    );

  }

}
```

This component will be a stateless widget and will only display the value that we send to it, in the Text widget.

Displaying a List of Custom Cards

Import the above component to the main.dart like so:

```
import 'commonComponents/customCard.dart';
```

I then replace the code of the home page body, from the one above to this:

```
body: Center(
child: Container(
child: ListView.builder(
itemCount: _counter,
itemBuilder: (context, int index) {
return CustomCard(
index: ++index,
                );
            },
        )
    ),
),
```

It now displays a List of CustomCard elements, up to the number of times the button is clicked. The itemCount is used to define the number of elements the ListView must display. The itemBuilder returns the actual item that is displayed.

And that’s a simple example of using Flutter.

Why you should learn Flutter?

Simple to learn and use

Flutter is a modern framework, and you can feel it! It’s way simpler to create mobile applications with it. If you have used Java, Swift, or React Native, you'll notice how Flutter is different.

I personally never liked mobile application development before I started using Flutter.

What I love about Flutter is that you can create a real native application without a bunch of code.

Quick Compilation: Maximum Productivity

Thanks to Flutter, you can change your code and see the results in real-time. It’s called Hot-Reload. It only takes a short amount of time after you save to update the application itself.

Significant modifications force you to reload the app. But if you do work like

design, for example, and change the size of an element, it’s in real-time!

Ideal for startup MVPs

If you want to show your product to investors as soon as possible, Flutter is a good choice.

Here are some reasons to use it for your MVP:

- It’s cheaper to develop a mobile application with Flutter because you don’t need to create and maintain two mobile apps (one for iOS and one for Android).
- One developer is all you need to create your MVP.
- It’s performant – you won't notice the difference between a native application and a Flutter app.
- It’s beautiful – you can easily use widgets provided by Flutter and personalize it to create a valuable UI for your customers

Good documentation

It’s important for new technology to have good documentation. But it’s not always the case that it has it!

You can learn a lot from Flutter's documentation, and everything is very detailed with easy examples for basic use cases. Each time I've had a problem with one of my widgets in my code, I have been able to check the documentation and the answer was there.

A growing community

Flutter has a robust community, and it's only the beginning!

As you may know, I love to share my knowledge and useful content on programming on my website. I need to know I'm working on a technology full of potential with a lot of backers.

When I started using Flutter, the first thing I did was search for communities, and to my surprise… there are a considerable number of places to exchange info on Flutter.

I will give you some examples of places I love to check daily. Feel free to send me a message on Twitter with your suggestions.

i. Flutter Awesome: An awesome list that curates the best Flutter libraries and tools. This website publishes daily content with lots of examples, application templates, advice, and so on.
ii. Awesome Flutter: A GitHub repository (linked to Flutter Awesome) with a list of articles, videos, components, utilities, and so on.
iii. It's all widgets!: An open list of apps built with Flutter.
iv. Flutter Community: A Medium publication where you can find articles, tutorials, and much more.
v. Supported by Android Studio and VS Code

vi. Flutter is available on different IDEs. The two main code editors for developing with this technology are Android Studio (IntelliJ) and VS Code.

vii. Android Studio is a complete software with everything already integrated. You have to download Flutter and Dart plugins to start.

viii. VS Code is a lightweight tool, and everything is configurable through plugins from the marketplace.

I use Android Studio because I don’t need to configure a lot of things to work.

You are free to choose your preferred IDE!

Bonus

Freelance

If you want to start doing some freelance work, you should think about using Flutter.

In 2020, I believe that this technology is going to explode. And that means a lot of people are going to search for developers who know how to use it.

The biggest platform for freelancers in France, called Malt, recently published the tech trends of this year. Flutter has grown by +303% on this platform between 2018 and 2019.

Flutter 2020 Major Updates

While Flutter is new, its creators are not resting on their laurels. They are continually innovating and improving Flutter to make it a more useful and powerful tool for developers. Some key improvements that are worth calling out in 2020 are as follows:

Branching model – Flutter will be changing the way they release versions. From April onwards, they will create a new branch at the beginning of each month as a beta release. During that month and the months following, they will work to stabilize this release and will promote it to the newest version quarterly.

Alignment with Dart – In a similar fashion, the Dart release process has also been changed. Now Dart offers a beta channel too, and the release discrepancy between Flutter and Dart will be aligned.

Adobe XD support for Flutter — Adobe XD to Flutter plugin is now available for public testing with early access. “The ability to export designs to Flutter further reduces the latency between creative ideas and product development, as an XD-prototype can now become working Flutter code within minutes. Adobe XD supports design on Windows or macOS, and includes a free starter plan to get you up and running,” announced Tim Sneath.

Flutter Windows Alpha — Adding Windows to Flutter, with support for Windows 7 and above, gives adventurous developers something to get started with. This alpha release offers a solid foundation that must be stabilized over the coming months.

Sound null safety — When a developer chooses null safety, the types in code cannot be null by default, which means that values cannot be null unless the

developer says they can be null. With this update, the runtime null-dereference errors become edit-time analysis errors.

Mobile Autofill Support — Text autofill support in Flutter apps. With Flutter 1.20, Flutter has added the basic autofill functionali

Flutter’s Popularity and Perspective

Flutter’s popularity is easy to understand, and while half a million developers use Flutter monthly, the top five territories for Flutter developers are the US, China, India, Brazil, and the EU.

“You see today, how Flutter is already enabling developers all over the world to deliver beautiful apps to hundreds of millions of people throughout the globe. But this is also just the beginning.“ says Eric Seidel, Engineering Manager at Google. “Flutter is highly portable and already works on many form factors beyond phones… So if I look forward a few years, I see Flutter running in a ton more places.”

A technical preview of Flutter Web was introduced by Google, which allows developers to run Flutter applications in a browser in a clean form without changing the source code. It means that with Flutter, developers can 100% go further beyond app development on mobile. Moreover, this marks Flutter’s transition from a cross-platform mobile application framework to a full-blown cross-platform development tool. However, for now, NIX experts see in Flutter web development more difficulties, since it takes a lot of time to get the correct display of elements and the operation of the application from a single base. We are looking forward to new web updates.

Despite the fact that everything, except the mobile part of the framework, is not yet considered ready for production, an experienced Flutter developer can make

a pure flutter application, running today on any major platform — including Android, iOS, Web browser, Windows, macOS, Linux and even embedded devices — and the application will function properly without changing and customizing the code for each platform separately.

A New Programming Style

Although Dart is a typical static-typed OOP language, using it in Flutter is a bit different; for mobile developers used to Java or Swift, it requires a different approach. There is no declarative syntax akin to .xaml files in Xamarin Forms or .nib in iOS. Everything is defined by creating Dart code for widgets.

At first glance, this might sound quite tedious, but hot reload makes it very fast to experiment with complicated user interfaces. The downside is that you have some large files to define a widget; for example, in Red Brogdon's open-source flutterflip reversi clone, the main game board widget is around 170 lines long.

Another plus: There are no issues regarding Dart formatting, as the Dart plugin includes a Dart formatter; a single right-click in one of the IDEs tidies up your code.

Conclusion

Flutter/Dart work very well together. The only problems I've experienced are importing projects between IDEs; the location of Dart and Flutter SDKs varies between everyone's PCs, and so those need to be configured.

Flutter is one of the most innovative mobile technologies on the market right now. For businesses looking to create applications on both iOS and Android, Flutter is a great option. If you are seeking apps with amazing UI and high performance — Flutter is the best option as well.

Flutter is not a universal remedy for everything, but it is a 100% promising framework, considering the breadth of coverage and speed of implementation.