A

Technical Seminar report on

Flutter

Submitted in partial fulfillment of the requirements for the award of degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

By

N.VENKATESH

(15AT1A05A3)

Under the guidance of

Dr. K. Seshadri Ramana M.Tech,Ph.D

Professor

Department of Computer Science and Engineering

G.PULLAIAH COLLEGE OF ENGINEERING AND TECHNOLOGY (AUTONOMOUS)

Accredited by NAAC with 'A' Grade and NBA Accredited (EEE,CSE & ECE)

Permanently Affiliated to JNTUA, Ananthapuramu, Approved by AICTE, New Delhi.

(Recognized by UGC under 2(f) & 12(B) and ISO 9001:2008 Certified Institution)

Nandikotkur Road, Kurnool, A.P-518452.

www.gpcet.ac.in

2015-2019

G.PULLAIAH COLLEGE OF ENGINEERING AND TECHNOLOGY (AUTONOMOUS)
Accredited by NAAC with 'A' Grade and NBA Accredited (EEE,CSE & ECE)
Permanently Affiliated to JNTUA, Ananthapuramu, Approved by AICTE, New Delhi.
(Recognized by UGC under 2(f) & 12(B) and ISO 9001:2008 Certified Institution)
Nandikotkur Road, Kurnool, A.P-518452.
www.gpcet.ac.in

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Technical seminar entitled **"Flutter"** is being submitted by **N.Venkatesh (15AT1A05A3)** in partial fulfillment for the award of degree of **BACHELOR OF TECHNOLOGY** in **COMPUTER SCIENCE AND ENGINEERING** to Jawaharlal Nehru Technological University Anantapur, Anantapuramu during the academic year 2018-19 is a record of bonafide work carried out by him/her under my guidance and supervision.

Guide

Dr. K. Seshadri Ramana M.Tech,Ph.D

Professor.

Head of the Department

Dr. S. Prem Kumar Ph.D

Professor & H.O.D.

Date:

ACKNOWLEDGEMENT

I take immense pleasure to express my deep sense of gratitude to my beloved guide **Dr.K.Seshadri Ramana, Professor, Department of CSE, G. Pullaiah College of Engineering and Technology** for his guidance and suggestions, keen interest and thoroughly encouragement extended throughout period of Technical Seminar.

With immense pleasure, I express my deep sense of gratitude to my beloved **Head of the Department, Dr.S.Prem Kumar, Department of CSE, G. Pullaiah College of Engineering and Technology,** who had been a source of inspiration and for his timely guidance in the conduct of my Technical Seminar.

I express gratitude to my beloved **Principal, Dr.C.Srinivasa Rao, G. Pullaiah College of Engineering and Technology** for his encouragement and cooperation in carrying out the Technical Seminar.

I express my deep regards to our **Management** for facilitating the required resources for the successful completion of Technical Seminar.

Finally, yet importantly, I would like to express my heartfelt thanks to my beloved parents for their blessings, my friends/classmates for their help and wishes for the successful completion of this Technical Seminar.

With gratitude,

N.Venkatesh

(15AT1A05A3)

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

In recent years, it is difficult to develop applications for both iOS and Android with in less time. To overcome this, Google introduced a new framework called Flutter. It is a new reactive framework and platform for building high-performance and beautiful mobile apps. It is used extensively at Google to build business-critical apps, and by third-party developers to build popular apps. It is also used as a SDK which provides the support to build beautiful mobile apps in record time. Flutter is highly customizable, which allows it to build apps that are brand-centric, or with the look and feel of native Android and iOS apps from a single code base.

1.INTRODUCTION

1.1 INTRODUCTION

From the past few years mobile app development is a relatively recent field of endeavor. Third-party developers have been able to build mobile apps for less than a decade. The Apple iOS and Google Android SDKs were based on different languages such as Objective-C and Java, respectively.

Flutter takes a different approach in an attempt to make mobile development better. It provides a framework application developer work against and an engine with a portable runtime to host applications. The framework builds upon the Skia graphics library, providing widgets that are actually rendered, as opposed to being just wrappers on native controls.

This approach gives the flexibility to build a cross-platform application in a completely custom manner like the web wrapper option provides, but at the same time offering smooth performance. Meanwhile, the rich widget library that comes with Flutter, along with a wealth of open-source widgets, makes it a very feature-rich platform to work with. Put simply, Flutter is the closest thing mobile developers have had for cross-platform development with little to no compromise.

Like React Native, Flutter provides reactive-style views. Flutter takes a different approach to avoiding performance problems caused by the need for a JavaScript bridge by using a compiled programming language, namely Dart.

Dart is compiled “ahead of time” (AOT) into native code for multiple platforms. This allows Flutter to communicate with the platform without going through a JavaScript bridge that does a context switch. Compiling to native code also improves app startup times.

2. ARCHITECTURE

2.1 ARCHITECTURE

The Flutter framework is organized into a series of layers, with each layer building upon the previous layer.

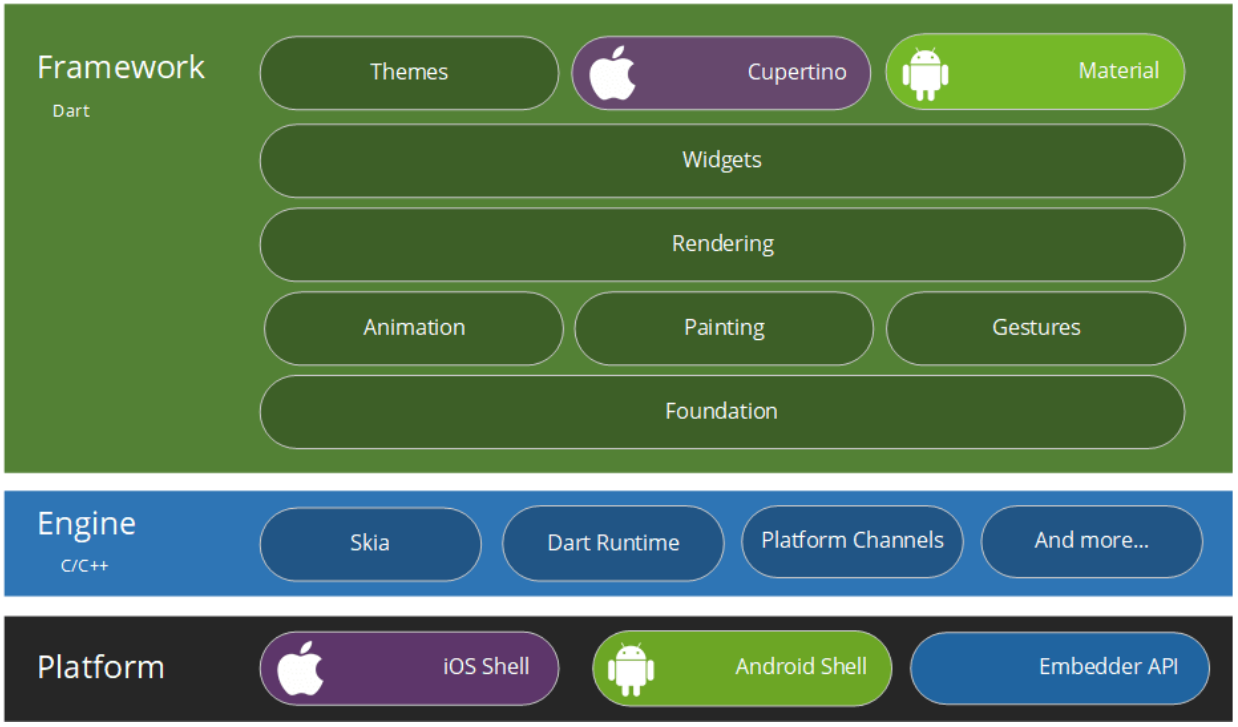

Figure 2.1 Architecture

2.1.1 Dart Framework

Flutter apps are written in the Dart language and make use of many of the language's more advanced features. On Android, and on Windows, macOS and Linux via the semi-official Flutter Desktop Embedding project, Flutter runs in the Dart virtual machine which features a just-in-time execution engine. Due to App Store restrictions on dynamic code execution, Flutter apps use ahead-of-time (AOT) compilation on iOS.

A notable feature of the Dart platform is its support for "hot reload" where modifications to source files can be injected into a running application. Flutter extends this with support for stateful hot reload, where in most cases changes to source code can be reflected immediately in the running app without requiring a restart or any loss of state.

2.1.2 Flutter Engine

Flutter's engine, written primarily in C++, provides low-level rendering support using Google's Skia graphics library. Additionally, it interfaces with platform-specific SDKs such as those provided by Android and iOS.

The Flutter Engine is a portable runtime for hosting Flutter applications. It implements Flutter's core libraries, including animation and graphics, file and network I/O, accessibility support, plugin architecture, and a Dart runtime and compile toolchain. Most developers will interact with Flutter via the Flutter Framework, which provides a modern, reactive framework, and a rich set of platform, layout and foundation widgets.

2.1.3 Platforms

At the platform level, Flutter provides a Shell, that hosts the Dart VM. The Shell, is platform specific, giving access to the native platform APIs and hosting the establishing the platform relevant canvas. There is also an embedder API, if you want to use Flutter like a library, instead of hosting running an app. The Shells, also help provide communication to the relevant IMEs and the systems application lifecycle events.

3. WIDGETS

3.1 Widgets

`

Everything in Flutter is a widget. This includes user interface elements, such as ListView, TextBox, and Image, as well as other portions of the framework, including layout, animation, gesture recognition, and themes, to name just a few.

Widgets are necessary for an app's view and interface. They must have a natural look and feel regardless of screen size. They also must be fast, extensible, and customizable. Flutter takes the everything's a widget approach. It has a rich set of widgets and extensive capabilities for creating complex custom widgets. In Flutter, widgets aren't only used for views. They're also used for entire screens and even for the app itself.

As Flutter's documentation puts it, each widget is an immutable declaration of part of the user interface. Other frameworks separate views, view controllers, layouts, and other properties. Flutter, on the other hand, has a consistent, unified object model: the widget.

A widget can define a structural element (like a button or menu); a stylistic element (like a font or color scheme); an aspect of the layout (like padding); and so on.

Widgets form a hierarchy based on their composition. Each widget nests inside of and inherits properties from its parent. There's no separate application object. Instead, the root widget serves this role.

Flutter has a full set of widgets in Google's Material Design and in Apple's style with the Cupertino pack. Widget rendering happens directly in the Skia engine without using Original Equipment Manufacturer widgets. So we get a smoother UI experience compared with other cross platform frameworks.

3.1.1 Stateless Widgets

A stateless widget is a widget that describes part of the user interface by building a constellation of other widgets that describe the user interface more concretely. The building process continues recursively until the description of the user interface is fully concrete.

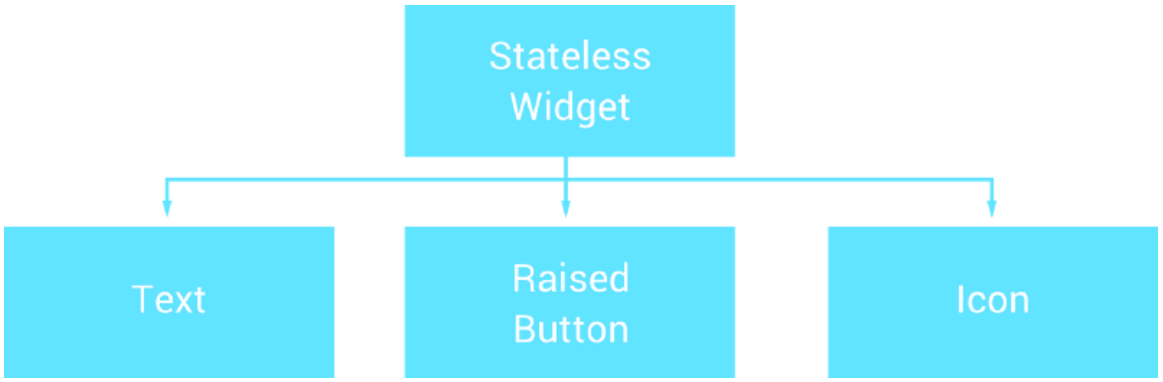

Figure 3.1.1 Stateless Widgets

Stateless widget are useful when the part of the user interface you are describing does not depend on anything other than the configuration information in the object itself and the BuildContext in which the widget is inflated. For compositions that can change dynamically, e.g. due to having an internal clock-driven state, or depending on some system state, consider using StatefulWidget.

The Text widget is instantiated using a constructor and then these properties are used to build the widget to be displayed on the screen. If a widget's parent will regularly change the widget's configuration, or if it depends on inherited widgets that frequently change, then it is important to optimize the performance of the build method to maintain a fluid rendering performance.

So, a stateless widget is **not dynamic**. It doesn't depend on any data other than that that is passed into it, meaning that the only way it can set and how it is represented is when arguments are passed into it's constructor.

3.1.2 Stateful Widgets

A stateful widget is a widget that describes part of the user interface by building a constellation of other widgets that describe the user interface more concretely. The building process continues recursively until the description of the user interface is fully concrete.

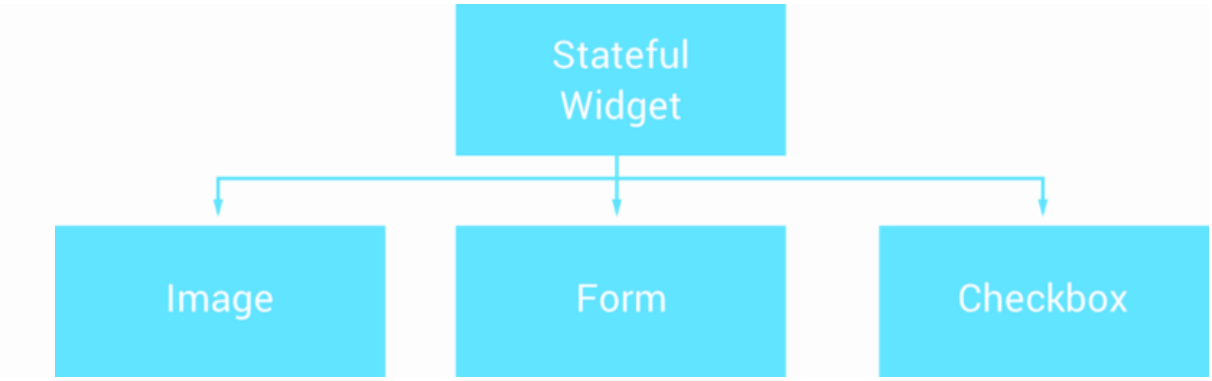

Figure 3.1.2 Stateful Widgets

Stateful widget are useful when the part of the user interface you are describing can change dynamically, e.g. due to having an internal clock-driven state, or depending on some system state. For compositions that depend only on the configuration information in the object itself and the BuildContext in which the widget is inflated, consider using StatelessWidget.

StatefulWidget instances themselves are immutable and store their mutable state either in separate StateObjects that are created by the createState method, or in objects to which that State subscribes, for example Stream or ChangeNotifier objects, to which references are stored in final fields on the StatefulWidget itself.

The State objects associated with StatefulWidget are grafted along with the rest of the subtree, which means the State object is reused (instead of being recreated) in the new location. However, in order to be eligible for grafting, the widget must be inserted into the new location in the same animation frame in which it was removed from the old location.

4. TYPES OF VIEWS

4.1 Web Views

The first cross-platform frameworks were based on JavaScript and WebViews. Examples include a family of related frameworks: PhoneGap, Apache Cordova, Ionic, and others. Before Apple released their iOS SDK they encouraged third party developers to build webapps for the iPhone, so building cross-platform apps using web technologies was an obvious step.

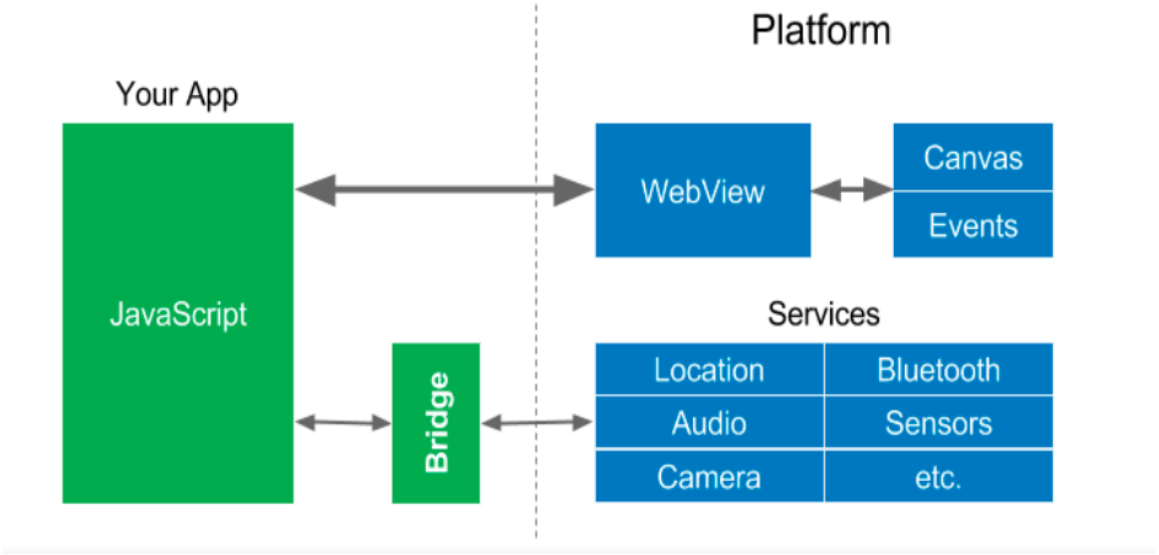

Figure 4.1 Web Views

The app creates HTML and displays it in a WebView on the platform. Note that it is difficult for languages like JavaScript to talk directly to native code (like the services) so they go through a "bridge" that does context switches between the JavaScript realm and the native realm. Because platform services are typically not called all that often, this did not cause too many performance problems.

4.2 Reactive Views

Reactive web frameworks like ReactJS (and others) have become popular, mainly because they simplify the creation of web views through the use of programming patterns borrowed from reactive programming. In 2015, React Native was created to bring the many benefits of reactive-style views to mobile apps.

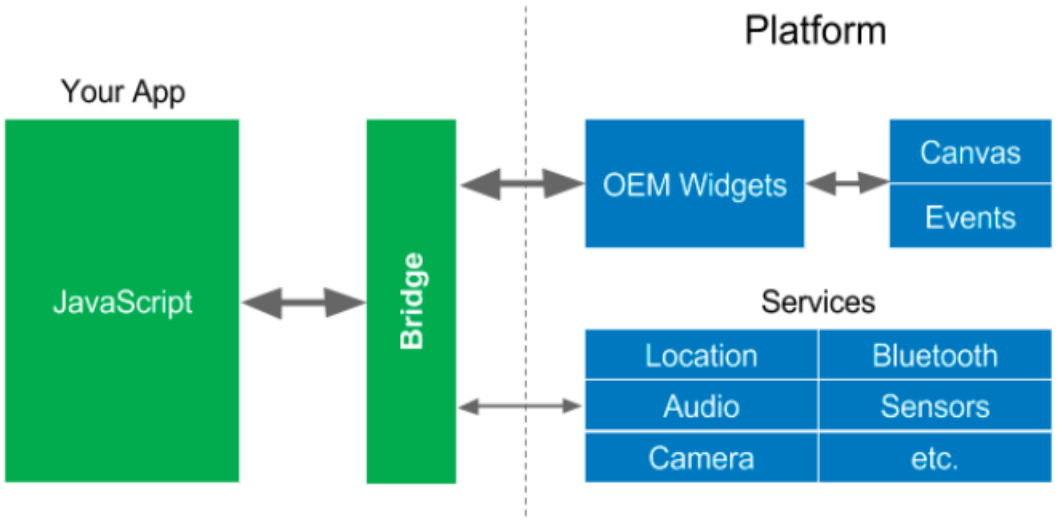

Figure 4.2.1 Reactive Views

React Native is very popular (and deserves to be), but because the JavaScript realm accesses the platform widgets in the native realm, it has to go through the bridge for those as well. Widgets are typically accessed quite frequently up to 60 times a second during animations, transitions, or when the user "swipes" something on the screen with their finger so this can cause performance problems.

Libraries for reactive web views introduced virtual DOM. DOM is the HTML Document Object Model, an API used by JavaScript to manipulate an HTML document, represented as a tree of elements. Virtual DOM is an abstract version of the DOM created using objects in the programming language, in this case JavaScript.

In reactive web views (implemented by systems like ReactJS and others) the virtual DOM is immutable, and is rebuilt from scratch each time anything changes. The virtual DOM is compared to the real DOM to generate a set of minimal changes, which are then executed to update the real DOM. Finally, the platform re-renders the real DOM and paints it into a canvas.

Figure 4.2.2 Reactive Views with real DOM

React Native does a similar thing, but for mobile apps. Instead of DOM, it manipulates the native widgets on the mobile platform. Instead of a virtual DOM, It builds a virtual tree of widgets and compares it to the native widgets and only updates those that have changed.

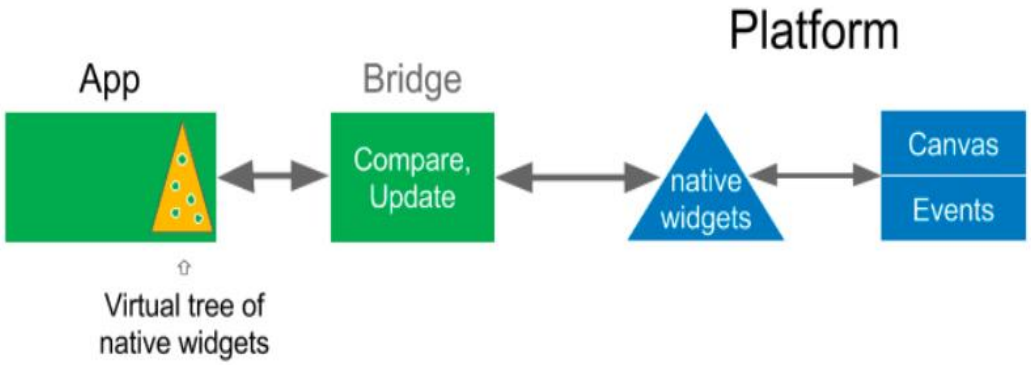

Figure 4.2.3 Reactive Views with native widgets

React Native has to communicate with the native widgets through the bridge, so the virtual tree of widgets helps keep passes over the bridge to a minimum, while still allowing the use of native widgets. Finally, once the native widgets are updated, the platform then renders them to the canvas.

5.DISPLAYING HELLO WORLD APP IN FLUTTER

The hello world program in flutter is represented as below.

```
import 'package:flutter/material.dart';
void main() {
  runApp(
    new Center(
      child: new Text(
        'Hello, world!'
      ),
    ),
  );
}
```

The output of the above program is shown below.

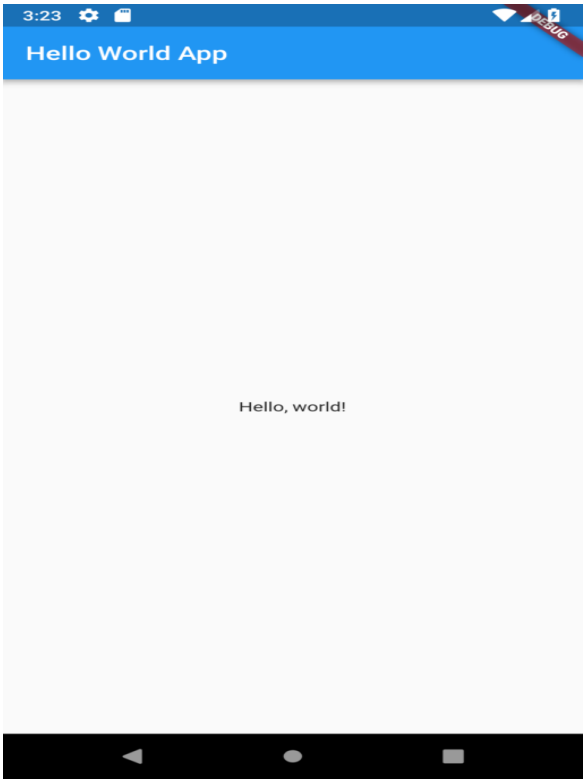

Figure 5.Displaying Hello World app in Flutter

6. COMPARISON TO OTHER DEVELOPMENT PLATFORMS

6.1 Apple or Android Native

Native applications offer the least friction in adopting new features. They tend to have user experiences more in tune with the given platform since the applications are built using controls from the platforms vendors themselves (Apple or Google) and often follow design guidelines set out by these vendors.

One big advantage native applications have is they can adopt brand new technologies Apple and Google create in beta immediately if desired, without having to wait for any third-party integration. The main disadvantage to building native applications is the lack of code reuse across platforms, which can make development expensive if targeting iOS and Android.

6.2 React Native

React Native allows native applications to be built using JavaScript. The actual controls the application uses are native platform controls, so the end user gets the feel of a native app. For apps that require customization beyond what React Native's abstraction provides, native development could still be needed. In cases where the amount of customization required is substantial, the benefit of working within React Native's abstraction layer lessens to the point where in some cases developing the app natively would be more beneficial.

6.3 Xamarin

There are two different approaches that need to be evaluated. For their most cross-platform approach, there is Xamarin.Forms. Although the technology is very different to React Native, conceptually it offers a similar approach in that it abstracts native controls. Likewise, it has similar downsides with regard to customization.

Unlike these alternatives, Flutter attempts to give developers a more complete cross-platform solution, with code reuse, high-performance, fluid user interfaces, and excellent tooling.

7. FUTURE AND NEXT STEPS

Flutter just got into the app development industry and already started to set its popularity offering numerous. No Doubt, the future seems to be bright for Flutter. Initially, Google also launched Flutter Beta 3 with additional exciting features and functions, making it even more powerful and robust than the previous version.

As we know, Flutter has a lot of potentials to offer in the respective mobile app development industry which is strengthen quality and productivity concurrently. Soon, it is expected to take over the app development world.

8. CONCLUSION

Even in beta, Flutter offers a great solution for building cross-platform applications. With its excellent tooling and hot reloading, it brings a very pleasant development experience. The wealth of open-source packages and excellent documentation make it easy to get started with. Looking forward, Flutter developers will be able to target Fuchsia in addition to iOS and Android. Considering the extensibility of the engine's architecture, Flutter land on a variety of other platforms as well. With a growing community, it's a great time to jump in.

REFERENCES

1. *Chris Bracken. "Release v0.0.6: Rev alpha branch version to 0.0.6, flutter 0.0.26 (#10010) · flutter/flutter". GitHub. Retrieved 2018-08-08.*
2. https://github.com/flutter/flutter/releases
3. https://developers.googleblog.com/2018/09/flutter-release-preview-2-pixel-perfect.html
4. https://github.com/flutter/flutter/wiki/Changelog
5. *"FAQ - Flutter". Retrieved 2018-08-08.*
6. *"Google's "Fuchsia" smartphone OS dumps Linux, has a wild new UI". Ars Technica.*
7. *Amadeo, Ron (1 May 2015). "Google's Dart language on Android aims for Java-free, 120 FPS apps". Ars Technica.*
8. *"Google Announced Flutter Release Preview 2". Apptunix.*
9. Jump up to: *a b c d "Technical Overview - Flutter". flutter.io. Retrieved 2017-12-13.*
10. *Stephenwzl (2018-08-01). "Flutter's Compilation Patterns". ProAndroidDev. Retrieved 2018-12-06.*
11. *Lelel, Wm (26 February 2018). "Why Flutter Uses Dart". HackerNoon. Retrieved 5 December 2018.*
12. *"foundation library - Dart API". docs.flutter.io. Retrieved 2017-12-13.*
13. *"Material Design Widgets - Flutter". flutter.io. Retrieved 2017-12-13.*
14. *"Cupertino (iOS-style) Widgets - Flutter". flutter.io. Retrieved 2017-12-13.*