

EXPERIMENT 5A

NAME: Shivam Pawar

UID: 2019230068

NAME: Vishal Salvi

UID: 2019230069

NAME: Shreyas Patel

UID: 2018130043

CLASS: TE COMPS

BATCH: C

DATE:

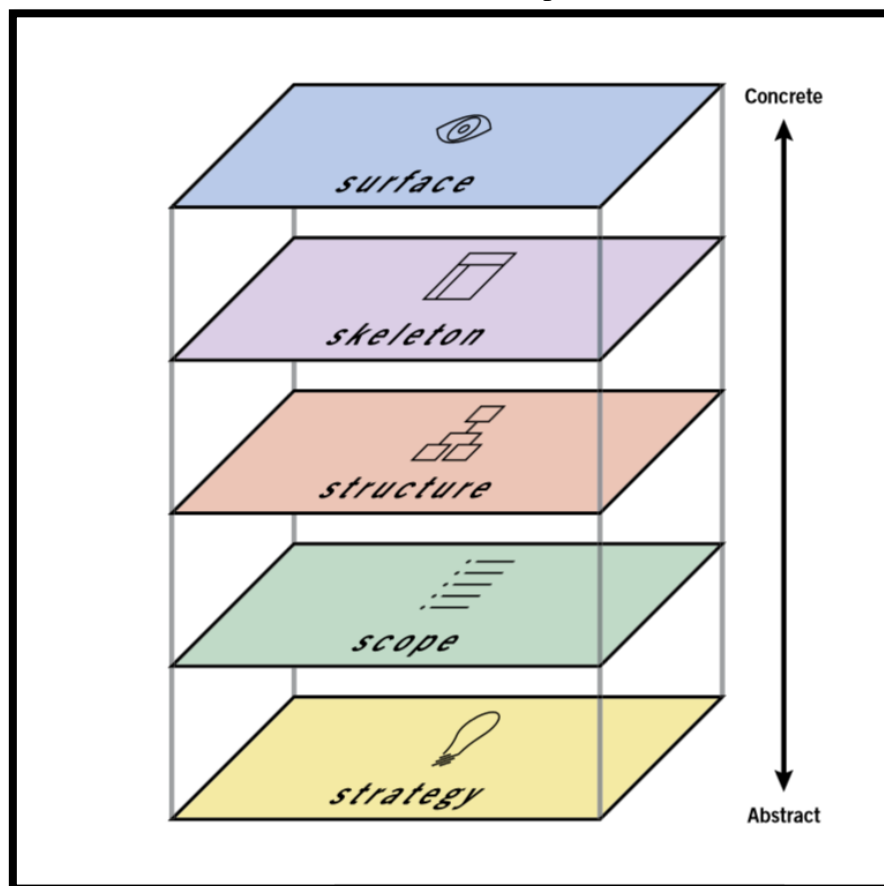
Aim: To develop app for mobile device

5.1 Create and map the database design in the five planes of UXD

Theory:

The Five Elements of UX

There are five dependent layers, each level builds on the level before it, and they start with abstract level towards concrete one (from bottom to top).



The Five Elements of UX

1. Strategy



Strategy

The reason for the product, application or the site, why we create it, who are we doing this for, why people are willing to use it, why they need it. The goal here is to define the user needs and business objectives.

This could be done through **Strategic Research Process**, where you interview users, and all stakeholders in addition to review the competing products or companies.

2. Scope



Scope

Defines the functional and content requirements. What are the features, and content contained in the application or product. The requirements should fulfil and be aligned with the strategic goals.

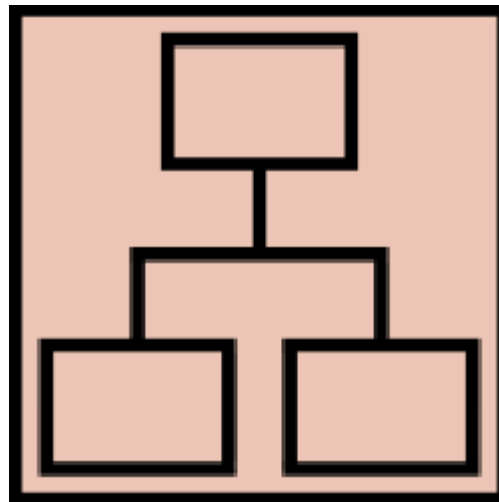
Functional Requirements It's the requirements about the functions, or features in the product, how features work with each other, and how they interrelate with each other. These features is what user need to reach the objectives.

Content Requirements It's the information we need in order to provide the value. Information like text, images, audio, videos, ...etc. Without defining the content, we have no idea about the size or time required to complete the project.

Functional Vs Content requirements

The feature is having a media player for songs, while the content is the audio files for these songs.

3. Structure



Structure

Defines how user interact with the product, how system behave when user interact, how it's organized, prioritized, and how much of it. Structure is split into two components, **Interaction Design & Information Architecture**.

Interaction Design Given the functional requirements, It defines how user can interact with the product, and how the system behaves in response to the user interactions.

Information Architecture Given the content requirements, It defines the arrangement of content elements, how they are organized, to facilitate human understanding.

Good Interaction design

- helps people to accomplish their goals.
- effectively communicates interactivity and functionality (what user can do).
- informs user about state changes (file has been saved, or any feedback), while they interact.
- prevents user error or mistakes, like the system asks user to confirm potentially harmful action (i.e. deletion).

Good Information Architecture

- organizes, categorizes, and prioritizes the information based on user needs and business objectives.
- makes it easy to understand and move through information presented.
- flexible to accommodate growth and adapt to change.
- appropriate for the audience.

4. Skeleton

Skeleton determines the visual form on the screen, presentation and arrangement of all elements that makes us interact with the functionality of the system that exist on the interface. Also how the user moves through the information, and how information is presented to make it effective, clear, obvious.

Wireframes are widely used to create a visual format, which is a Static diagrams that represent a visual format of the product, including content, navigation and ways for interactions.

Skeleton is split into three components *Interface Design, Navigation Design, & Information Design*.

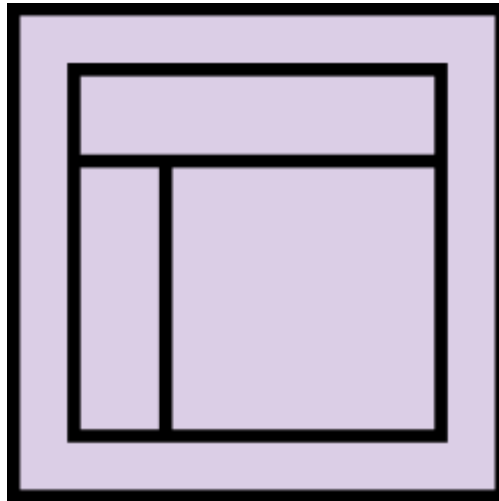
Interface Design presenting and arranging interface elements to enable users to interact with the functionality of the system.

Navigation Design how to navigate through the information using the interface.

Information Design defines the presentation of information in a way that facilitates understanding.

So, the Skeleton should answer these questions:

- What visual form of all things that will be presented on screen
- How interactions will be presented and arranged
- How will users move around the site, or application
- How content will be presented clearly

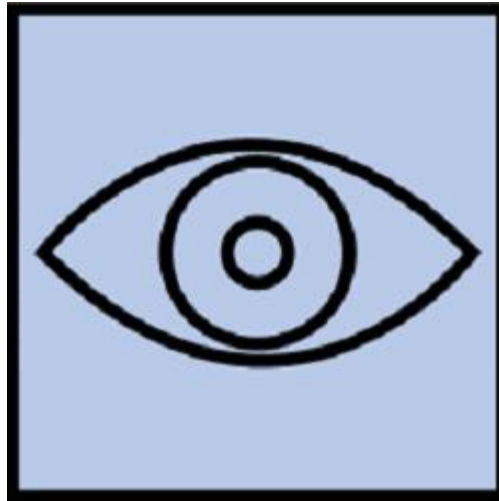


Skeleton

5. Surface

It's the sum total of all the work and decisions we have made. It determines how does the product will look like, and choosing the right layout, typography, colors, ...etc.

In Surface, we are dealing with _Visual Design(Sensory Design), It's concerned about the visual appearance of content, controls, which gives a clue of what user can do, and how to interact with them. It should make things easier to understand, increase cognitive ability to absorb what users see on the screen.



Surface

How UX Elements Work Together

Each layer is dependent on the other layers below it. If you screw up in Strategy, you will pay for it during the whole project. When you make a decision or choice in a layer, this decision will affect on your future decisions in the next layers.

Your decisions may change over time, and if you consider decisions to be fixed, you will end up build something that nobody wants. If you are in Surface, and you need to enhance a functionality, you can go back to Structure, and make it better.

To summarize how those 5 elements work together, we start by the **Strategy**, which is the foundation of any successful UX. Strategy becomes **Scope** when user & business needs translated to requirements for content & functionality.

Scope is given **Structure** when we define the ways of interaction with the system functionality, the system response, and how information is organized. Sketching each screen of the system(i.e. using wireframes) to present the areas of interactions and structure defined in Structure, and how information will be presented clearly, is what we do in **Skeleton**. Finally, In **Surface**, we take all the work and decisions we have made into the final visual presentation.

Flutter:

Flutter is a cross-platform UI toolkit that is designed to allow code reuse across operating systems such as iOS and Android, while also allowing applications to interface directly with underlying platform services. The goal is to enable developers to deliver high-performance apps that feel natural on different platforms, embracing differences where they exist while sharing as much code as possible.

During development, Flutter apps run in a VM that offers stateful hot reload of changes without needing a full recompile. For release, Flutter apps are compiled directly to machine code, whether Intel x64 or ARM instructions, or to JavaScript if targeting the web. The framework is open source, with a permissive BSD license, and has a thriving ecosystem of third-party packages that supplement the core library functionality.

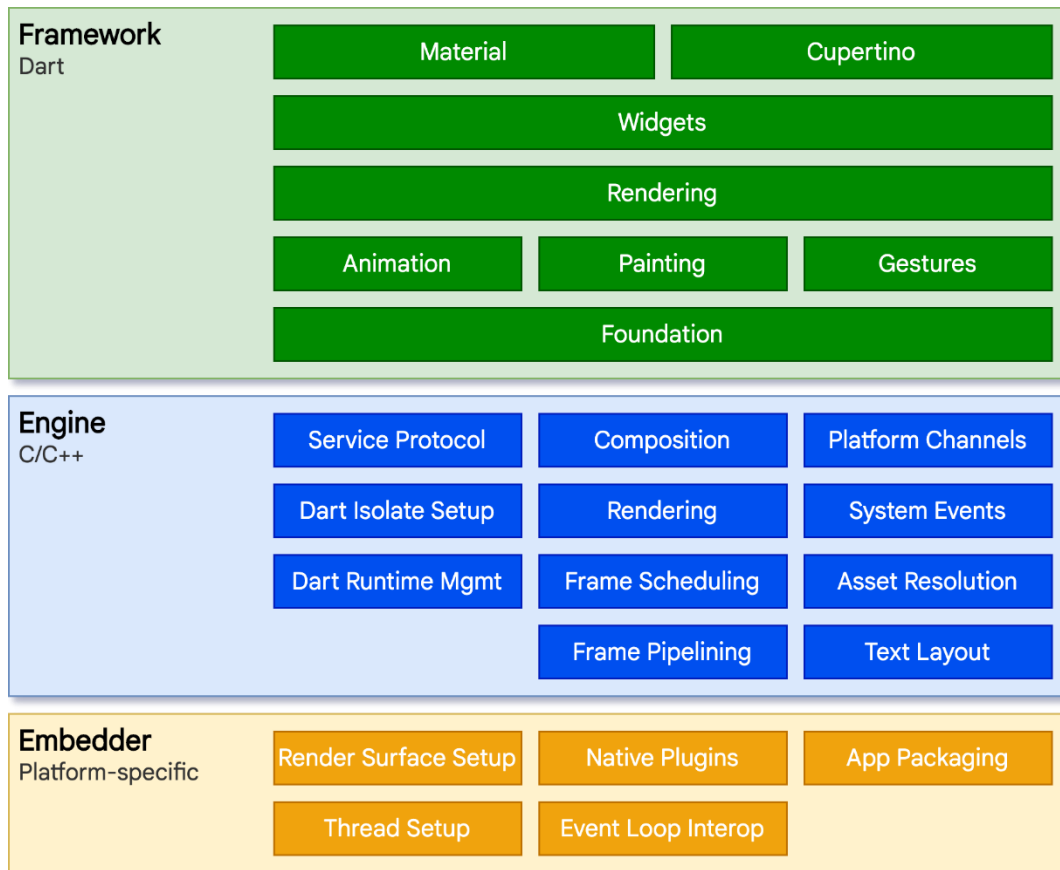
This overview is divided into a number of sections:

1. The **layer model**: The pieces from which Flutter is constructed.
2. **Reactive user interfaces**: A core concept for Flutter user interface development.
3. An introduction to **widgets**: The fundamental building blocks of Flutter user interfaces.
4. The **rendering process**: How Flutter turns UI code into pixels.
5. An overview of the **platform embedders**: The code that lets mobile and desktop OSes execute Flutter apps.
6. **Integrating Flutter with other code**: Information about different techniques available to Flutter apps.
7. **Support for the web**: Concluding remarks about the characteristics of Flutter in a browser environment.

Architectural layers

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

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



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

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

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

- Basic **foundational** classes, and building block services such as **animation**, **painting**, and **gestures** that offer commonly used abstractions over the underlying foundation.
- The **rendering layer** provides an abstraction for dealing with layout. With this layer, you can build a tree of renderable objects. You can manipulate these objects dynamically, with the tree automatically updating the layout to reflect your changes.
- The **widgets layer** is a composition abstraction. Each render object in the rendering layer has a corresponding class in the widgets layer. In addition, the widgets layer allows you to define combinations of classes that you can reuse. This is the layer at which the reactive programming model is introduced.

- The **Material** and **Cupertino** libraries offer comprehensive sets of controls that use the widget layer's composition primitives to implement the Material or iOS design languages.

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

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

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.

Why you should learn Flutter?

I selected some of the reasons why I like Flutter and why I want to use it next year. I will give you details and my feedback below.

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.

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 my top 4 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!

- [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.
- [Awesome Flutter](#): A GitHub repository (linked to Flutter Awesome) with a list of articles, videos, components, utilities, and so on.
- [It's all widgets!](#): An open list of apps built with Flutter.
- [Flutter Community](#): A Medium publication where you can find articles, tutorials, and much more.

Supported by Android Studio and VS Code

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

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

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

About The Platform

Flutter is very new, but a promising platform, that has attracted the attention of large companies who've released their apps already. It is interesting because of its simplicity compared to developing web applications, and because of its speed as compared with native applications. High performance and productivity in Flutter are achieved by using several techniques:

- Unlike many other popular mobile platforms, Flutter **doesn't use JavaScript in any way**. Dart is the programming language. It compiles to binary code, and that's why it runs with the native performance of Objective-C, Swift, Java or Kotlin.
- Flutter **doesn't use native UI components**. That may sound awkward at first. However, because components are implemented in Flutter itself, there is no communication layer between the view and your code. Due to this, games hit the best speed for their graphics out of the smartphones. So buttons, text, media elements, background are all drawn by Flutter's graphics engine. As an aside, it should be mentioned that the bundle of the Flutter "Hello, World" application is quite small: iOS $\approx 2.5\text{Mb}$ and Android $\approx 4\text{Mb}$.
- Flutter **uses a declarative approach**, inspired by the React web framework, to build its UI based on widgets (named "components" in the world of the web). To get more out of widgets, they are **rendered only when necessary**, usually when their state has been changed (just like the Virtual DOM does for us).
- In addition to all of the above, the framework has **integrated Hot-reload**, so typical for the web, but still missing on native platforms. This allows the Flutter framework to automatically rebuild the widget tree, allowing you to quickly view the effects of your changes.

There is an excellent article on the practical use of these features from an Android developer who recreated his application from Java to Dart and shared his impressions.

I wanted to share with you some numbers from his article.

- **Java** (before): Number of files = 179 and Lines of code 12,176
- **Dart** (after): Number of files = 31 and Lines of code 1,735.

You can read more about the [technical details of the platform](#) or look at [examples of applications](#).

About Dart

Dart is a programming language that we'll use to develop our application in Flutter. Learning it isn't hard if you have experience with Java or JavaScript. You will quickly get it.

I tried to write an article on Dart for you, to describe the minimal scope that is required for Flutter. After several attempts, I was still failing to write it so that it was short and covered the core concepts at the same time. Authors of [A Tour of the Dart Language](#) coped well with this task!

Benefits of Flutter App Development Services

Flutter has attained superiority in the market because it has mesmerized a large community of developers and other app owners. Let's look at the advantages of the Flutter app development services for developing a mobile application.

1. Hot Reload

The best part of this feature - Hot Reload is that developers and designers can easily identify all the changes and improvements that have been made to the code right away in the app. That's why Hot Reload strengthens a bond between developers and designers when they are looking for improvements on how the app looks and checks effect immediately.

2. High Performance



There're many factors that impact the performance of an app, including CPU usage, frame number per second, request number per second, average response time, and many more. The rate of Flutter is 60fps, at which contemporary screens display a smooth and clear picture.

With this frame rate, a human eye can identify any lag. If you compare it with React Native and Xamarin, this framework is ahead with 220-millisecond launch time and 58fps.

3. Immediate Updates

Flutter offers hot reload functionality that allows you for instant updates without the need for plugins. A hot reload also allows you to view updates in real-time. If you face an error while running the code, the framework lets you fix it immediately and to carry on without having to restart it.

With hot reload, you can improve your productivity and it also allows for experimentation without lengthy delays and assists with fast iterations.

4. Custom Widgets for Quick UI Coding



Flutter has ready-designed and custom widgets. These widgets are used to create an excellent app interface and its appearance. While many approaches could be utilized by different objects like controllers, views, and layout, this framework features a unified and consistent object model. Every object in this tool is a widget - fonts, color schemes, menus, buttons, and padding also. By combining the widgets to form layouts, you can utilize widgets on any customization level. The widgets of Flutter are consistent and have extensive capabilities.

5. Mild Learning Curve

Learning a Dart programming language is the easiest thing to learn. Many developers with little coding knowledge can develop prototypes and apps with the framework. The mobile app development experience will not weigh into this development.

You will find many video lessons, documents, a starting guide, and practical lessons over the internet.

Disadvantages of Flutter App Development Services

Likewise, Flutter has downfalls too. The Flutter technology is not matured at the moment, which means it still has a lot of room to grow, expand, and get better.

1. Large File Sizes

One big loophole that cannot be ignored is the large file size of apps developed in Flutter. Now for some cases, these file sizes could be a significant issue and cause a developer to choose an alternative tool for the development. As we can see and find a sufficient memory storage space on the phone and that does not happen everywhere in the world with most of the users. Many older devices are unable to store additional apps without users being forced to pick and choose between an app or photos/music on their device. However, this file size offers you improved runtime and performance so it's not easy to understand the audience you are appealing to.

2. Lack of Third-party Libraries

Third-party libraries and packages have a significant impact on software development as it enables some features for developers. These third-party libraries are normally free, open-source, pre-tested, and easily available. You may not find every single feature you need for the development, for now.

However, since Flutter is new for mobile app development, it's not easy to find such free packages and libraries. The tool is still in the growing phase and improving. Hence, you will have to wait for this tool to use or choose an alternative for long-term development.

3. Issues with iOS

Flutter is developed by Google. This is why developers are worried about its implementation for iOS. Since Google is directly interested in fixing bugs in the shortest amount of time, building Android apps on Flutter is fast and enjoyable.

One of the latest updates in Flutter is a pixel-perfect iOS appearance. iPhone settings were created on the framework to enable the Cupertino widgets. But based on iOS 10 and iOS 11 features were updated later and released for a while.

4. Dart

Flutter is using a Dart programming language. However, it has both benefits and drawbacks. This object-oriented programming language is not as great as other languages like C#, Java, Objective C, and JavaScript.

Not many freshers will be able to develop an app using this language. So, this is an essential factor to keep in mind while developing a cross platform application.

Is Flutter Good For Development?

So, by identifying its advantages and disadvantages, we have concluded that Flutter has many more pros than cons for business and development teams. You can surely build beautiful, high-performance, and amazing cross platform mobile applications that fit your custom needs and requirements. It's worth considering Flutter, especially if you want to develop both for iOS and Android. Because it is cost effective to hire a flutter developer than hiring two native app developers (one for Android and one for iOS).

What's more than saving money and time. You can surely give it a try. For that, The One Technologies is the right choice for you cross platform mobile application development. We develop outstanding mobile apps for your industry.

When should you use Flutter, then?

It's clear to see that Flutter is not mature enough to handle more complex projects, at least for now. At the same time, however, **it's a good solution for an MVP (especially for startups)**. Actually, it's a common pattern with all relatively new technologies.

Essentially, whenever you have an idea for a mobile app but you're not exactly sure whether it's a good one: build your MVP with Flutter to cut costs and see your idea in action. If the MVP becomes successful, you should start thinking about "turning it" into native mobile apps instead.

Let's face it: developing two separate apps from the start would take much more time and money. That's also one of the reasons why startups with limited resources turn to cross-platform solutions like Flutter. Reusing code helps them bring their ideas to life without making substantial investments.

More established enterprises also seem to appreciate Flutter's [ability to build highly-branded experiences that support multiple platforms](#). If that's something that might interest you, you should consider giving this cross-platform technology a try.

Which is better: Flutter or React Native?

Flutter is said to be a strong contender to React Native. For now, though, React Native is more mature and stable - not to mention that it takes advantage of the most popular programming language, JavaScript, and already has a large community of users and supporters behind it.

At the same time, however, Flutter seems to be growing at an unprecedented rate. It also happens to be faster than React Native - there's no need to go through a JavaScript bridge, and thanks to the use of Dagger it's easy to write & compile code at speed. It might be just a matter of time before it's used more widely.

This doesn't mean that other cross-platform technologies become obsolete, though. After all, the nature of both Flutter and React Native allows to reduce time-to-market and makes developing mobile apps more efficient, which attracts increasingly more developers and app owners.

Here's a short comparison of Flutter vs React Native:

	Flutter	React Native
Definition	Google's UI toolkit for developing natively compiled applications for mobile, web, and desktop from a single codebase	A framework for building native iOS and Android applications with JavaScript
Creators	Google	Facebook
Official release	December 2018, Google I/O	March 2015, F8 Conference
Programming language	Dart	JavaScript
Free & open-source	+	+
When to choose it	<ul style="list-style-type: none"> - If you want to reduce time-to-market as much as possible - If you plan to scale the app across different platforms and operating systems - If you like experimenting with new, not fully mature technologies 	<ul style="list-style-type: none"> - If your developers are already fluent in JavaScript - If you prefer creating your app's UI with native components - If you appreciate having a vast number of tutorials and libraries at your disposal

What is React Native?

React Native (also known as RN) is a popular **JavaScript-based mobile app framework** that allows you to build natively-rendered mobile apps for iOS and Android. The framework lets you create an application for various platforms by using the same codebase.

React Native was first released by Facebook as an open-source project in 2015. In just a couple of years, it became one of the top solutions used for mobile development. React Native development is used to power some of the **world's leading mobile apps**, including Instagram, Facebook, and Skype. We discuss these and other examples of React Native-powered apps further in this post.

There are several reasons behind React Native's global success.

Firstly, by using React Native, companies can create code just once and use it to power both their iOS and Android apps. This translates to huge time and resource savings.

Secondly, React Native was built based on **React – a JavaScript library**, which was already hugely popular when the mobile framework was released. We discuss the differences between React and React Native in detail further in this section.

Thirdly, the framework **empowered frontend developers**, who could previously only work with web-based technologies, to create robust, production-ready apps for mobile platforms.

Interestingly, as with many revolutionary inventions, React Native was developed as a response to...a big technological mistake.

The history of React Native

When Facebook first decided to make its service available on mobile devices, instead of building out a native app like many top tech players at the time, they decided to run with a mobile webpage **based on HTML5**. However, the solution didn't stand the test of time, leaving much room for UI and performance improvements. In fact, in 2012, Mark Zuckerberg admitted that “the biggest mistake we made as a company was betting too much on HTML as opposed to native.”

Soon after, in 2013, Facebook developer Jordan Walke made a groundbreaking discovery – he found a method of generating **UI elements for iOS apps** by using JavaScript. This sparked a fire, and a special Hackathon was organized to further discover how much mobile development could be done using (so far, traditionally web-based) JavaScript solutions.

That's how React Native came to life. Initially developed just for iOS, Facebook quickly followed it up with Android support, before taking the framework public in 2015.

Just three years later, React Native was already **the second biggest project on GitHub**, as measured by the number of contributors. In 2019, it stood strong and came sixth, with over 9,100 contributors.

React vs. React Native

In the most simple terms, React Native isn't a 'newer' version of React, although React Native *does* use it.

React (also known as ReactJS) is a JavaScript library used for building the frontend of a website. Similarly to React Native, it was also developed by the Facebook engineering team.

Meanwhile, React Native – which is powered by React – lets developers use a set of UI components to quickly compile and launch iOS and Android apps.

Both React and React Native use a mixture of JavaScript and a special markup language, JSX. However, the syntax used to render elements in JSX components differs between React and React Native. Additionally, React uses some HTML and CSS, whereas React Native allows the use of native mobile user interface elements.

Here's an example of code from a Stack Overflow discussion:

“React JSX renders HTML-like components like <h1>, <p>, etc. [Meanwhile] react-native renders native app view components like <View> , <Text>, <Image>, <ScrollView>, so you can't directly reuse your UI component code unless you rework/replace all the elements.”

Hence, while the two frameworks are related to one another, they're used for different purposes. Knowledge of React won't be enough for iOS and Android mobile app development.

Before we proceed to analyze the advantages and disadvantages of React Native, let's first take a look at what cross-platform development is all about.

What is cross-platform development?

Cross-platform development is the practice of building software that is compatible with more than one type of hardware platform. A cross-platform application can run on Microsoft Windows, Linux, and macOS, or just two of them. A good example of a cross-platform application is a web browser or Adobe Flash that performs the same, irrespective of the computer or mobile device you run it on.

Cross-platform is considered the holy grail of software development – you can build your codebase once and then run it on any platform, as opposed to software built natively for a specific platform. Developers are able to use the tools they're proficient in, like JavaScript or C#, to build platforms they're foreign to. Software owners are also keen on it as product development, in terms of time to market and costs, is cut in half.

What are some of the characteristics of cross-platform development?

Wider audience

You don't have to decide which audience to target, i.e., iOS or Android users, as cross-platform software runs on both, which gives you access to a wider user base.

Platform consistency

There are some navigation and design differences between iOS and Android, which – in cross-platform development – are dealt with by default, thanks to the shared codebase. This helps with creating a consistent app brand identity on both platforms with less effort than if built on native.

Reusable code

This is one of the greatest advantages of cross-platform development – you can build just one codebase for both Android and iOS at the same time. Native app development requires writing code separately and frequently needs two different software developers to perform the job – one for iOS and one for Android.

Quicker development

Since only one codebase is required to handle iOS and Android, and everything is in one place, product development is much quicker. Cross-platform applications are built as single projects, even though they support different devices, and a large amount of code can be reused between platforms.

Reduced costs

Building cross-platform applications **can be 30% cheaper** than building native apps, all thanks to the ability to reuse code and faster development, which directly impacts the cost. What you've read so far might lead you to think that cross-platform development is flawless – it's not, it has some disadvantages. Let us get into them right now.

Requires more expertise to ensure high performance

It is a common myth that cross-platform apps perform worse than their native counterparts. For instance, both Flutter and React Native aim to run at 60 frames per second. In most cases, cross-

platform applications can perform to the same standard as native apps provided that the developers have enough skill and expertise.

Harder code design

Since cross-platform apps must be responsive to various devices and platforms, it makes coding more complex. This results in more work for developers who have to include exceptions for different devices and platforms to account for the differences – especially when it comes to more complex features.

Long feature release time

With every new feature release for Android or iOS, it takes a while to update both apps to support the new feature. Native apps are provided with the updates quicker.

While we're on cross-platform development, it's worth having a quick look at some of the cross-platform frameworks.

- **React Native** – developed and presented to the world by Facebook in 2015, it works just like React, but allows you to build apps for both mobile and desktop. The beauty of it is that you can code in JavaScript without having to master any specific coding languages a platform might require like Java, Swift, or Objective-C. React Native is focused on building a great user experience for mobile devices, which makes it a suitable option for apps that require high responsiveness and intuitive use.
- **Flutter** – released in 2017 by Google, it can be used beyond cross-platform mobile development. Flutter is perfect for experimenting with new features and fixing small bugs thanks to its fast refresh feature. It lets developers instantly verify the changes made by the most current updates without the need to restart the app after editing the source code.
- **Xamarin** – developed by Microsoft, this free and open-source solution allows 75-90% of the code to be shared between different systems. It's written in C#, which requires developers to know the language – although it's more stable, it's also harder to pick up than JavaScript. Interestingly, Microsoft has itself made a turn towards RN in recent years. As of 2019, there were 38 Microsoft-developed iOS and Android apps that leveraged React Native.

How does React Native work?

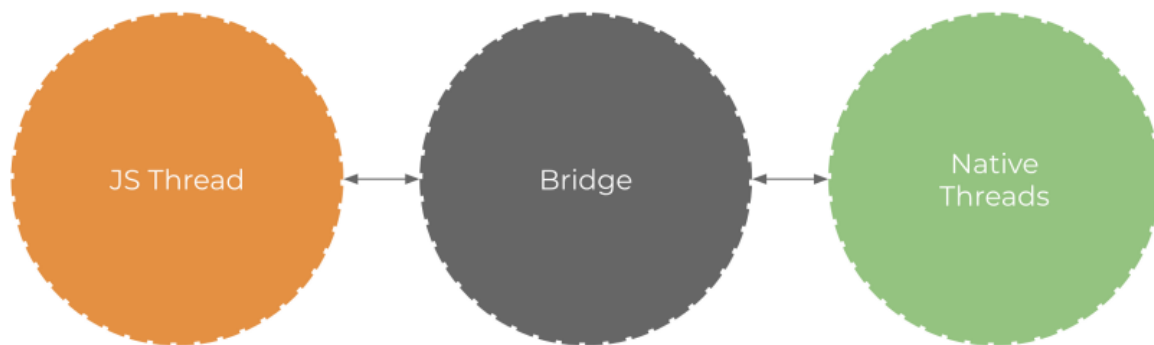
Now that we've discussed cross-platform development, let's take a look at the mechanics of React Native, and how it's different from anything we've seen before.

Don't worry if you're not a technical person – we're going to explain this in layman terms.

As mentioned earlier, React Native is written with a mixture of JavaScript and JXL, a special markup code resemblant of XML. The framework has the ability to communicate with both realms – JavaScript-based threads and existent, native app threads.

How does this communication work? React Native uses a so-called “bridge”. While JavaScript and Native threads are written in completely different languages, it's the bridge feature that makes bidirectional communication possible.

Here's a great visualization of the bridge concept:



Source: [*Hackernoon*](#)

This means that – if you already have a native iOS or Android app – you can still use its components or shift to React Native development.

What makes React Native unique?

The difference between React Native and other cross-platform development solutions (for example, [Cordova](#) and [PhoneGap](#)) is that React Native doesn't render WebViews in its code. It runs on actual, native views and components. This is one of the reasons for React Native's spectacular success.

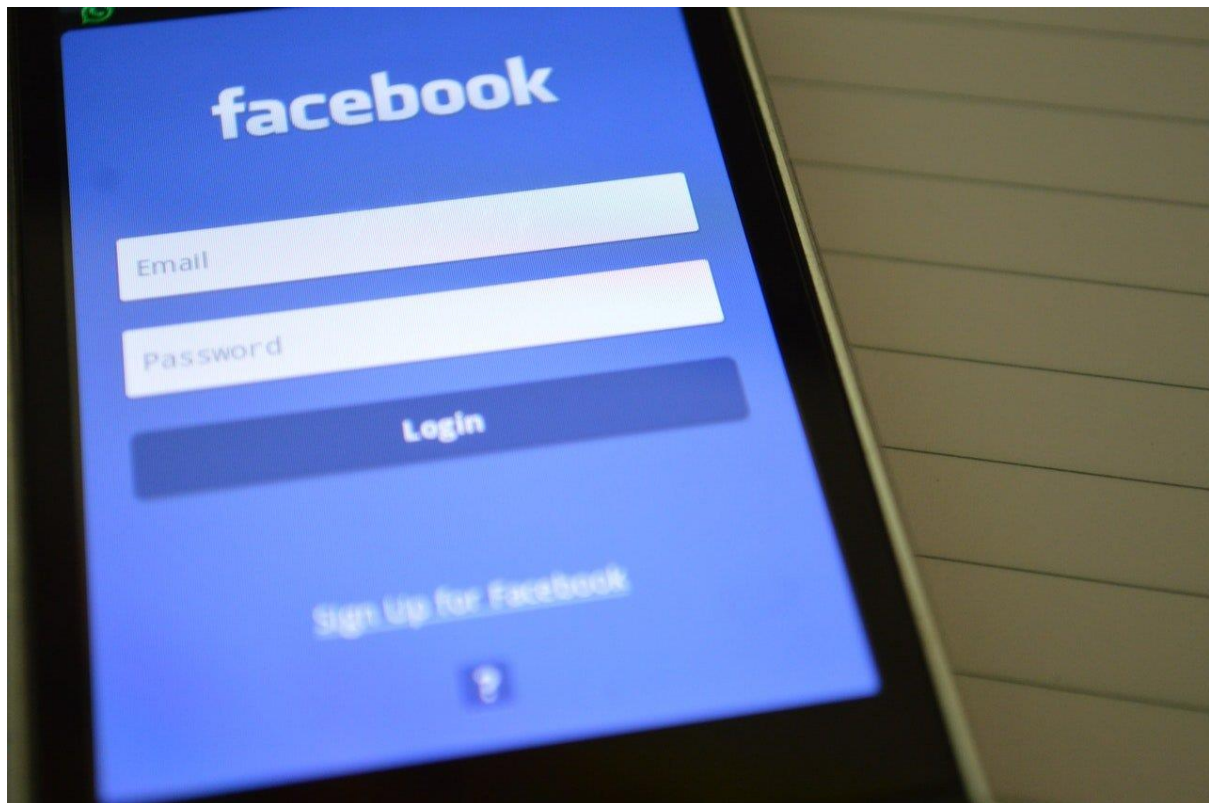
Examples of apps built with React Native

Now that you know what React Native is and how it works, it's time to have a look at [the products built with](#) it. Here is our selection of popular React Native apps.

Facebook

Facebook is one of the most popular React Native apps, and it's no surprise we're mentioning it first, as it gave birth to this programming language and is the main force behind its development.

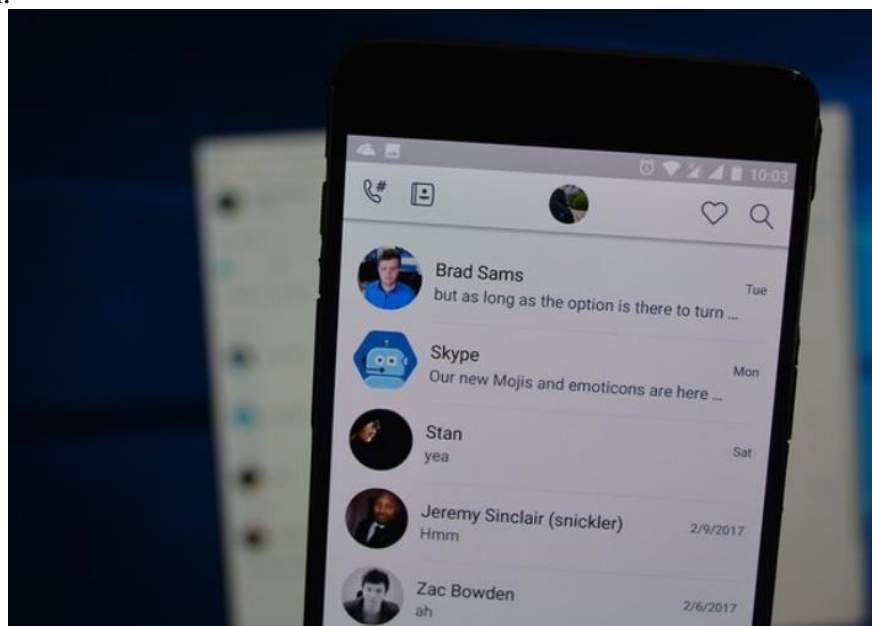
Facebook aimed to bring all the **web development benefits to mobile**, like quick iterations and having a single product development team, and this is how React Native came to life. The company used it to develop its own Ads Manager app in iOS and Android – both versions were created by the same dev team.



Skype

Skype is another good example of a React Native mobile app. In 2017, Skype announced that it was building a completely new app based on React Native. This brought a lot of excitement from its users, as the older version suffered from a few issues.

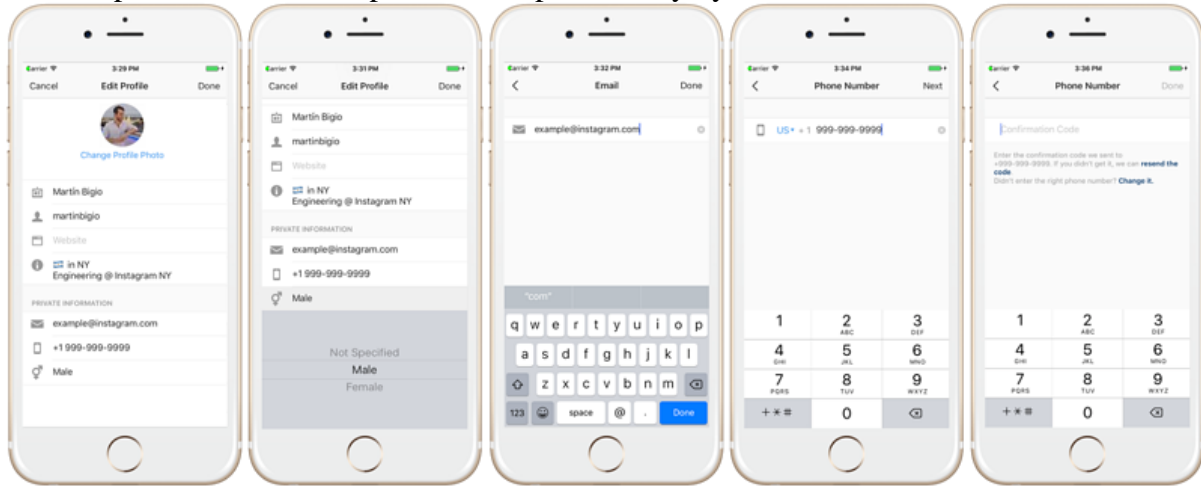
The new app was **completely redesigned**, from the icons to the new messaging interface, which now has three conversation sections: find, chat, and capture. Microsoft, which owns Skype, decided to use React Native not only in the mobile app but also in the desktop version of the platform.



Source: Twitter

Instagram

Instagram decided to integrate React Native into its existing native app, starting with the Push notification view that originated as the WebView. Luckily, it wasn't necessary to build the navigation infrastructure, as the UI was simple enough to cope without one. Using React Native allowed product teams to improve developer velocity by 85-99%.



Source: [Instagram.engineering](https://instagram.engineering)

Walmart

Another interesting React Native example? Walmart's iOS and Android apps. The American grocer has been known for making **bold technological decisions** – and one of them was rewriting its mobile apps entirely into React Native.

Previously, some parts of the Walmart app featured embedded web views, which – as Walmart Labs pointed out – fell below **“the standard that both we and our customers demand.”**

After shifting to React Native, the performance of both iOS and Android **apps improved immensely** – to a nearly native-level. Ninety-five percent of the codebase is shared for Android and iOS; furthermore, there's a single team that manages and develops both apps.



Source: [Google Play](https://play.google.com/store/apps/details?id=com.walmart.android)

Here are some other benefits that Walmart noticed after introducing React Native:

- A short time to market.
- Both platforms can be updated on the same day.
- As React Native is written in JavaScript, it's easy to onboard other teams.
- The UI of iOS and Android apps is platform-specific, giving the apps a native feel and a smooth UX.

It's best to sum up Walmart's opinion of React Native with their own words:

“From startups to Fortune 500 companies, if you're considering taking on a new mobile project, consider using React Native — we know you won't regret it.”

Benefits of React Native

We've discussed the products built using React Native, so let's move onto the advantages of React Native development and why you should choose it as a solution to build your mobile app.

Code reusability – cross-platform development

Being able to reuse code is the biggest advantage of React Native, and it indicates that apps can **run effectively on multiple platforms** – which is something that CEOs and Product Owners truly appreciate. They can integrate 90% of the native framework for reusing the code for both operational systems.

Engineers at Discord say, “we tried React Native the day it was released for Android. We were surprised by how easily and quickly we were able to make our comprehensive iOS app run on Android — took only two days and it is built!”

Another great piece of news is that it's possible to use the web application code for mobile app development if they're both using React Native. It also **speeds up development time** as it includes pre-developed components, which are included in the open-source library.

Large developer community

React Native is an open-source JavaScript platform that allows developers to contribute their knowledge to the framework's development, which is freely accessible to all.

If any developer experiences a problem while developing an app, then they can turn to the community for support (as of mid-2020, there are nearly **50,000 active contributors** to the React Native tag in Stack Overflow).

There will always be someone who'll be able to help them resolve their issues – this also has a positive impact on improving coding skills.

Cost efficiency

Another advantage of React Native development is greater cost efficiency. As mentioned earlier, this is because developers are able to use the same code to build applications for iOS and Android.

It means you don't have to hire two separate iOS and Android dev teams to finalize your project; a **small team is enough** to build it. The cost of developing apps in React Native is much lower than apps built using languages that don't allow for cross-platform development.

Fast refresh

Fast refresh allows developers to run the app while updating it to new versions and modifying the UI. Changes are visible immediately, and the developer is spared from rebuilding the entire app.

This leads to two significant benefits: **time savings** – as programmers save time on compilation and **increased productivity** – since they don't lose any state while incorporating changes into the app.

Simple UI

React Native development uses React JavaScript to **build the app's interface**, which makes it more responsive and faster with reduced load time, resulting in an overall better user experience. Thanks to the reactive UI and component-based approach, the framework is perfect for building apps with both simple and complex designs.

Fast applications

Some claim that React Native code might have a detrimental effect on an app's performance. Even though JavaScript won't run as fast as native code, this difference is unnoticeable to the human eye. To further prove it, we decided to run a test comparing two versions of a simple application **written in React Native and Swift** – both achieved similar performance results.

Future-proof

Considering the pace at which the framework took over the market and its simple approach to resolving development problems, the future of React Native for cross-platform apps looks bright. Even though it has a few disadvantages, which we'll discuss in the next section, its speed and convenience of development compensate for them.

With all this in mind, let's now take a look at why React Native potentially might not be a great fit for you.

React Native: Risks and Drawbacks

Here are the top four potential drawbacks you need to be aware of before you decide on developing a React Native app.

Lack of some custom modules

While React Native has been around for several years now, some custom modules either leave room for improvement or are entirely missing. This means that you might need to **run three separate codebases** (for React Native, iOS, and Android) instead of just one. That being said, it's not a common occurrence. Unless you're developing your app from scratch or trying to hack an existing one, you likely won't come across these issues.

Compatibility & debugging issues

While it may come as a surprise – after all, React Native is used by top tech players – **it's still in beta phase**. Your developers might come across various issues with package compatibility or debugging tools. If your developers aren't proficient in React Native, this might negatively impact your development as they spend time on lengthy troubleshooting.

Scalability

Most of the time, React Native will work very well for you even if your app eventually grows into a highly-sophisticated, **complex solution**. After all, companies like Facebook and Skype have found much success with the framework and have been using it consistently for many years. That being said, some companies have decided to back out from using React Native.

Airbnb, for instance, decided to use the framework for its mobile app back when the company was just an emerging startup. Over time, however, React Native proved to be unfit for the company's growth plans, and Airbnb resorted to developing two native apps. With the current advancements in RN, and with the right software architecture choices, scalability issues can easily be prevented.

Native developers' help needed

Remember the “bridging” feature we mentioned earlier in this post? As it showed, React Native bridges **JavaScript with native mobile code**. This means that if you put a developer who doesn't have knowledge of native mobile development in charge, they will have a hard time incorporating native code into the RN codebase. As a result, you'll need some assistance from Android or iOS developers to guide them through the process. If you're a small company, you might not want to hire native mobile developers, as this generates additional costs.

One way of tackling this is by engaging a software consultancy to give you a helping hand with the native iOS and Android elements.

Alternatives for React Native

Now that you have a good understanding of what React Native is, it's worth taking a look at some of its alternatives.

Flutter

We already mentioned it Flutter earlier in this article, where we compared it to React Native.

Ionic

Ionic is a complete open-source SDK designed for hybrid mobile development, introduced in 2013 by Drifty. It uses technologies like HTML, CSS, and JavaScript, as well as platforms like PhoneGap and Cordova, to create a native-like experience.

Ionic is built **on top of Angular**, and therefore if you're familiar with it, it'll be easy for you to pick up Ionic. It's packed with numerous built-in-components, which speed up development, making it smoother and easier. Additionally, it's a good option for fast prototyping as it offers a hybrid approach to product development.

In terms of performance, it's slower than React Native as it uses WebView, but the good news is, you can test the code on any browser.

Apache Cordova

Apache Cordova is a mobile application development framework originally introduced by Nitobi. It allows developers to build mobile apps using **CSS3, HTML5, and JavaScript** and not rely on the platform-specific APIs included in Android, iOS, or Windows Phone. Just like Ionic, Apache Cordova also uses WebView, which creates some limitations.

For example, iOS apps that run inside the default WebView engine run more slowly than the same app in the Safari mobile browser. What's more, as JavaScript is single-threaded, having too many things going on in the application code might lead to problems, like slow animations and reduced app responsiveness.

According to Johannes Stein, freelance software engineer – “By using Cordova, you can quickly turn your existing single page application into a mobile application for different platforms, at the cost of interactions not necessarily having the native feeling to their specific platform.”

PhoneGap

PhoneGap is a distribution of Apache Cordova, meaning that it's powered by Cordova but has some extra tools you can use, which are provided by Adobe.

**“It promises you an <easy life> as a mobile app developer, enabling you to use any JavaScript library and framework that you're comfortable working with.”
– Engineers at Optasy.**

PhoneGap is easy to work with, which makes it developer-friendly. They have a lot of frameworks and libraries at their disposal. It's based on the “write once, run on every platform” motto, so you can take advantage of cross-platform development.

Just select your favorite web technology and your app will run on all available platforms, without the need to build separate versions for each one.

Unfortunately, apps built with PhoneGap might suffer from **poorer user experience**, as web technology was created for, well, the web, not mobile apps. This makes handling animations problematic. Also, you're risking experiencing the same issues that web apps experience, which include browser-specific bugs.

Summary

React Native is an exciting framework that enables web developers to create robust mobile applications using their existing JavaScript knowledge. It offers faster mobile development, and more efficient code sharing across iOS, Android, and the Web, without sacrificing the end user's experience or application quality. The tradeoff is that it's new, and still a work in progress. If your team can handle the uncertainty that comes with working with a new technology, and wants to develop mobile applications for more than just one platform, you should be looking at React Native.

Installation:

Step 1: Install create-react-native-app

After installing NodeJS and NPM successfully in your system you can proceed with installation of create-react-native-app (globally as shown below).

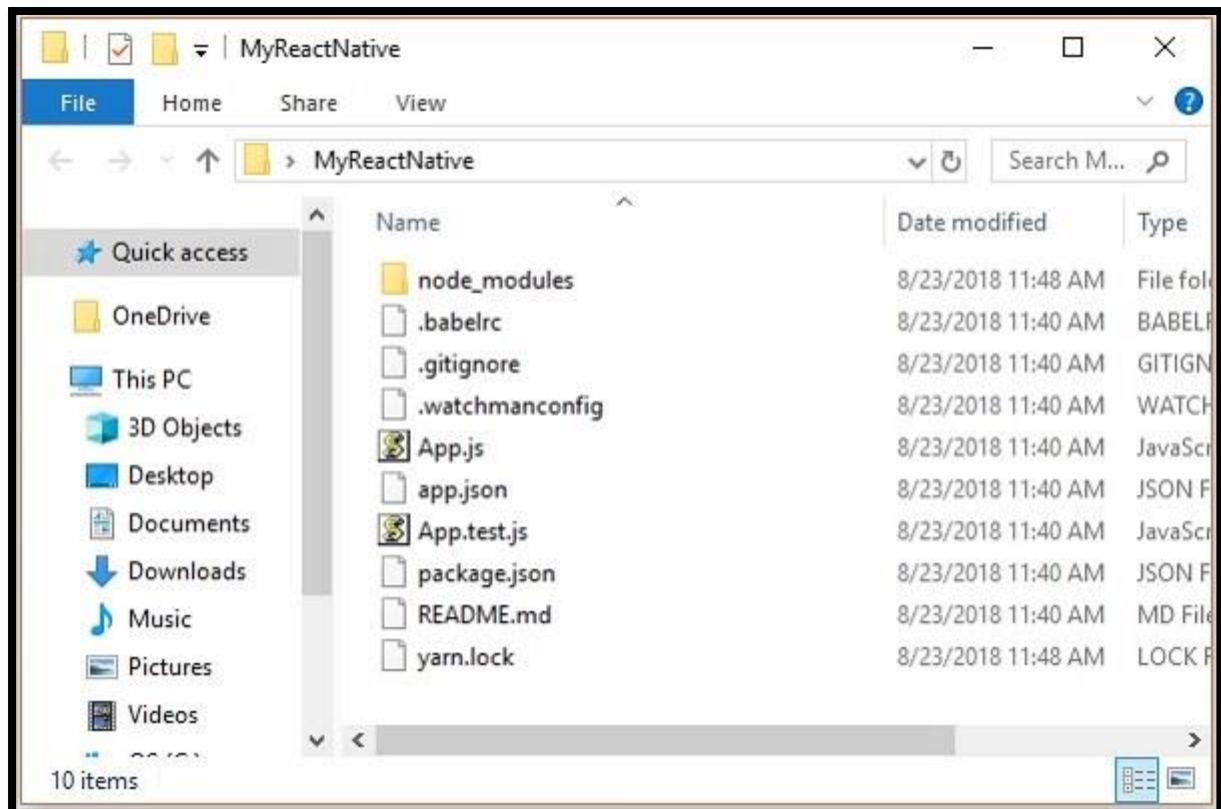
```
C:\Users > npm install -g create-react-native-app
```

Step 2: Create project

Browse through required folder and create a new react native project as shown below.

```
C:\Users > cd Desktop  
C:\Users\Desktop> create-react-native-app MyReactNative
```

After executing the above command, a folder with specifies name is created with the following contents.



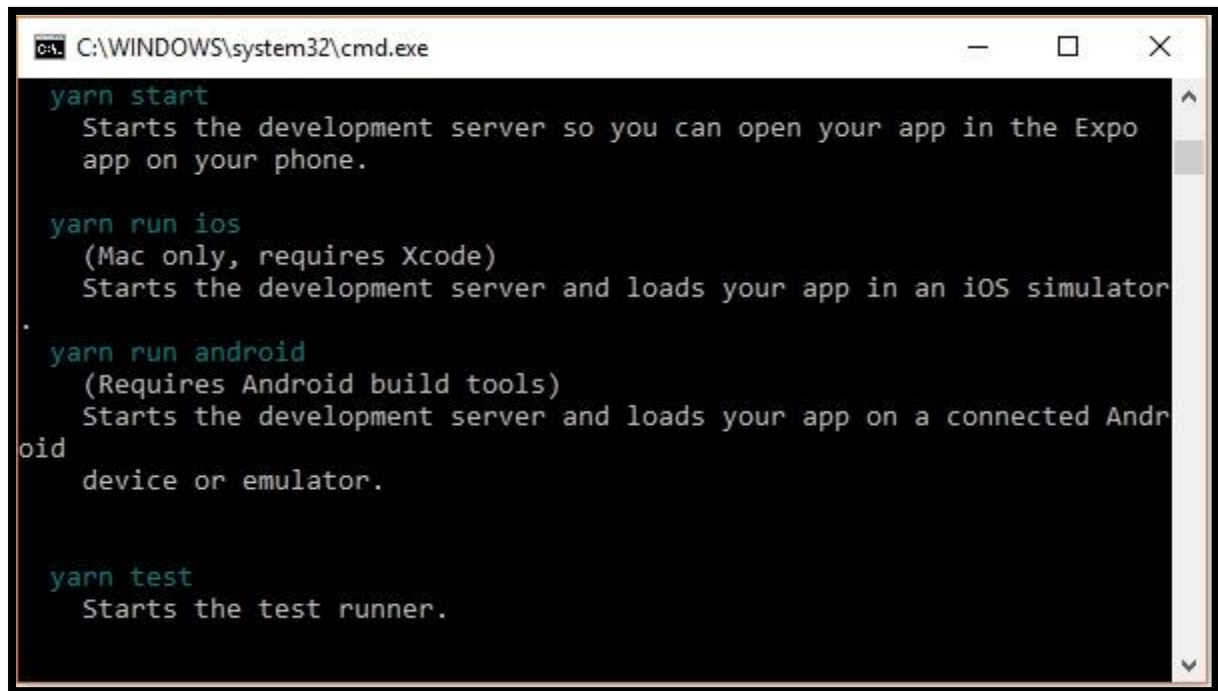
Step 3: NodeJS Python Jdk8

Make sure you have Python NodeJS and jdk8 installed in your system if not, install them. In addition to these it is recommended to install latest version of yarn to avoid certain issues.

Step 4: Install React Native CLI

You can install react native command line interface on npm, using the install -g react-native-cli command as shown below.

npm install -g react-native-cli



```
C:\WINDOWS\system32\cmd.exe

yarn start
  Starts the development server so you can open your app in the Expo
  app on your phone.

yarn run ios
  (Mac only, requires Xcode)
  Starts the development server and loads your app in an iOS simulator

yarn run android
  (Requires Android build tools)
  Starts the development server and loads your app on a connected And
oid
  device or emulator.

yarn test
  Starts the test runner.
```

Step 5: Start react native

To verify the installation browse through the project folder and try starting the project using the start command.

```
C:\Users\Desktop>cd MyReactNative
```

```
C:\Users\Desktop\MyReactNative>npm start
```

If everything went well you will get a QR code as shown below.



As instructed, one way to run react native apps on your android device is to using expo. Install expo client in your android device and scan the above obtained QR code.

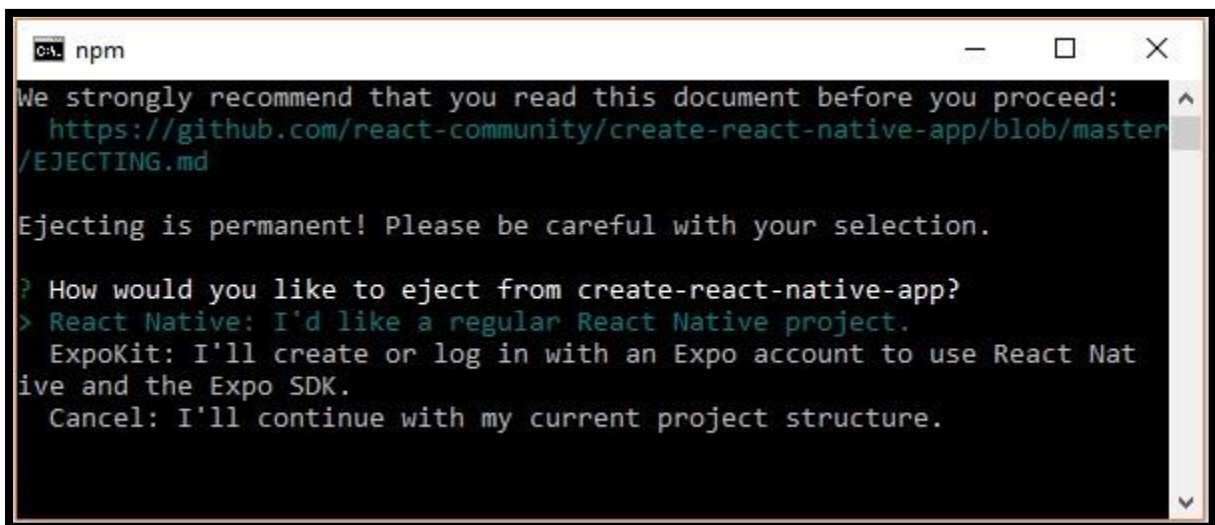
Step 6: Eject the project

If you want to run android emulator using android studio, come out of the current command line by pressing **ctrl+c**.

Then, execute run **eject command** as

```
npm run eject
```

This prompts you options to eject, select the first one using arrows and press enter.

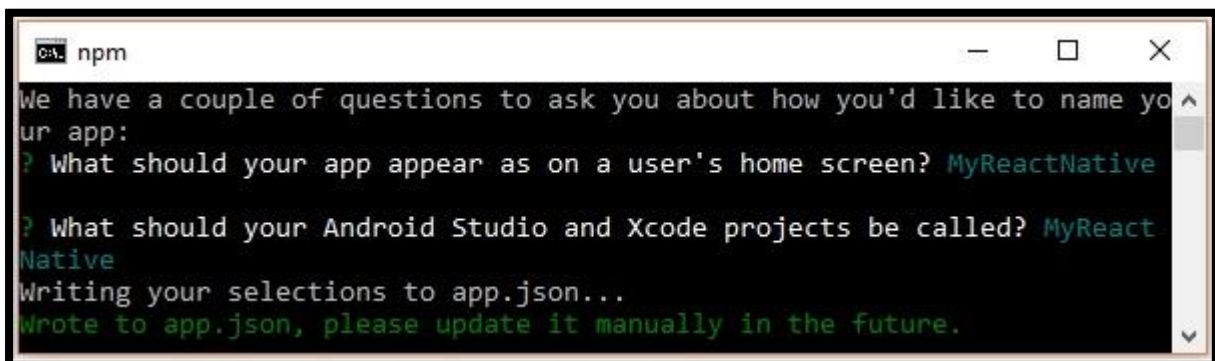
A terminal window titled 'npm' with standard Windows window controls. It displays the output of the 'npm run eject' command. The text includes a recommendation to read a document, a warning that ejecting is permanent, and a prompt asking how to eject. The first option, 'React Native: I'd like a regular React Native project.', is selected and highlighted in green.

```
CA: npm
We strongly recommend that you read this document before you proceed:
  https://github.com/react-community/create-react-native-app/blob/master/EJECTING.md

Ejecting is permanent! Please be careful with your selection.

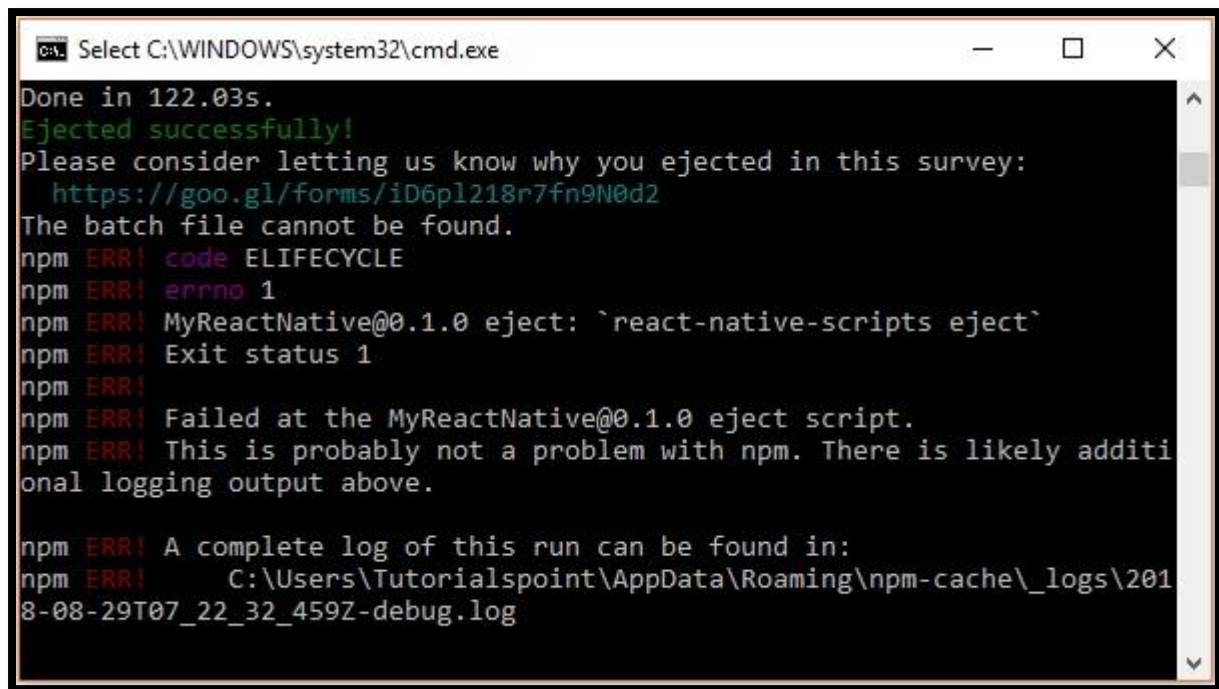
? How would you like to eject from create-react-native-app?
> React Native: I'd like a regular React Native project.
  ExpoKit: I'll create or log in with an Expo account to use React Native and the Expo SDK.
  Cancel: I'll continue with my current project structure.
```

Then, you should suggest the name of the app on home screen and project name of the Android studio and Xcode projects.

A terminal window titled 'npm' with standard Windows window controls. It displays the prompts for naming the app. The user has entered 'MyReactNative' for both the home screen name and the Android Studio/Xcode project name. The selections are highlighted in green.

```
CA: npm
We have a couple of questions to ask you about how you'd like to name your app:
? What should your app appear as on a user's home screen? MyReactNative
? What should your Android Studio and Xcode projects be called? MyReactNative
Writing your selections to app.json...
Wrote to app.json, please update it manually in the future.
```

Though your project ejected successfully, you may get an error as –



```
C:\> Select C:\WINDOWS\system32\cmd.exe

Done in 122.03s.
Ejected successfully!
Please consider letting us know why you ejected in this survey:
https://goo.gl/forms/iD6pl218r7fn9N0d2
The batch file cannot be found.
npm ERR! code ELIFECYCLE
npm ERR! errno 1
npm ERR! MyReactNative@0.1.0 eject: `react-native-scripts eject`
npm ERR! Exit status 1
npm ERR!
npm ERR! Failed at the MyReactNative@0.1.0 eject script.
npm ERR! This is probably not a problem with npm. There is likely additional logging output above.

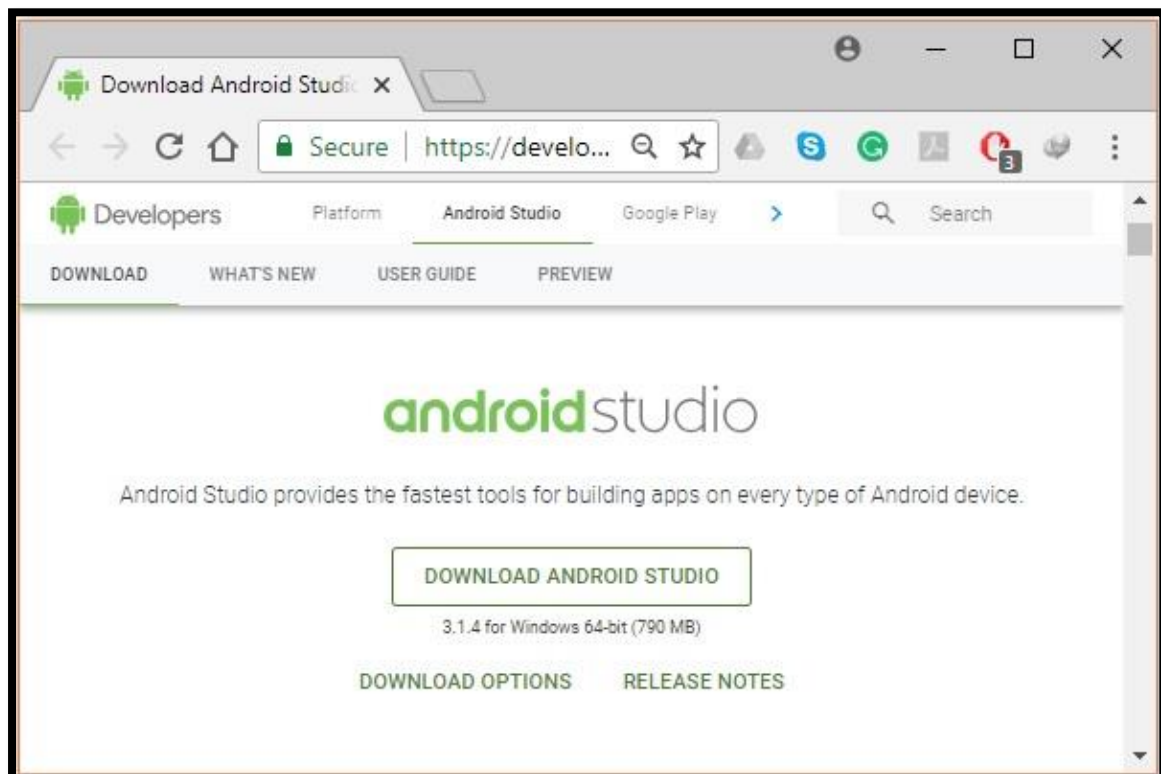
npm ERR! A complete log of this run can be found in:
npm ERR! C:\Users\Tutorialspoint\AppData\Roaming\npm-cache\_logs\2018-08-29T07_22_32_459Z-debug.log
```

Ignore this error and run react native for android using the following command –
react-native run-android

But, before that you need to install android studio.

Step 7: Installing Android Studio

Visit the web page <https://developer.android.com/studio/> and download android studio.

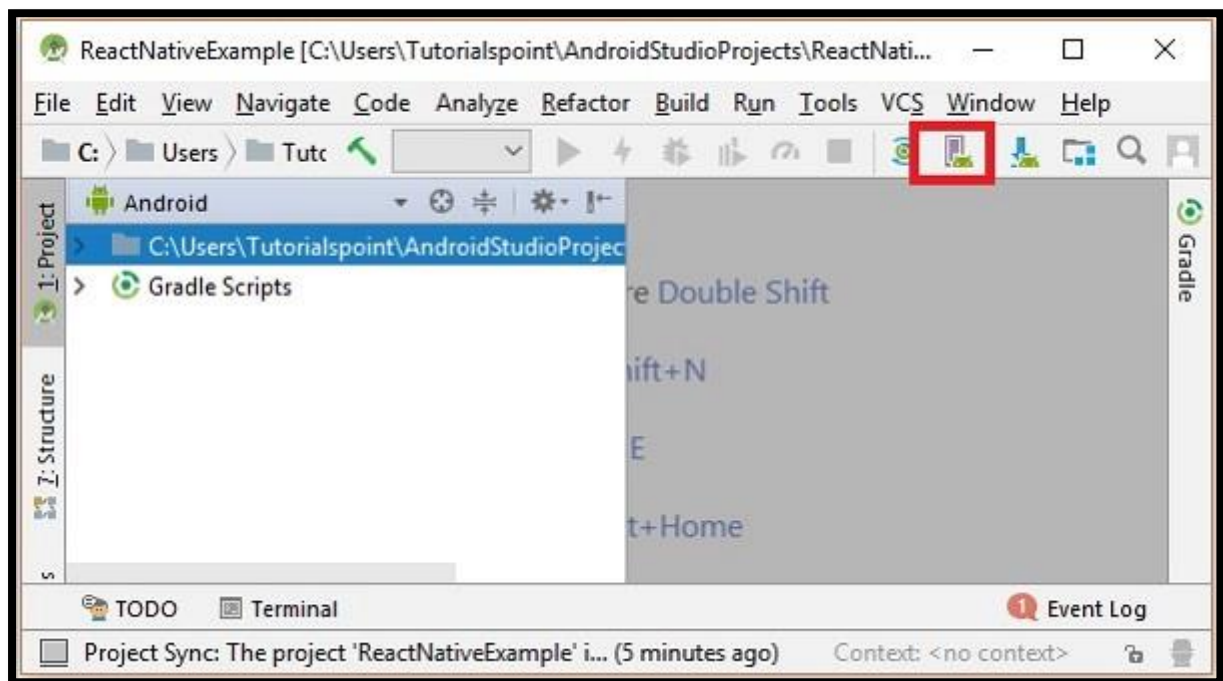


After downloading the installation file of it, double click on it and proceed with the installation.



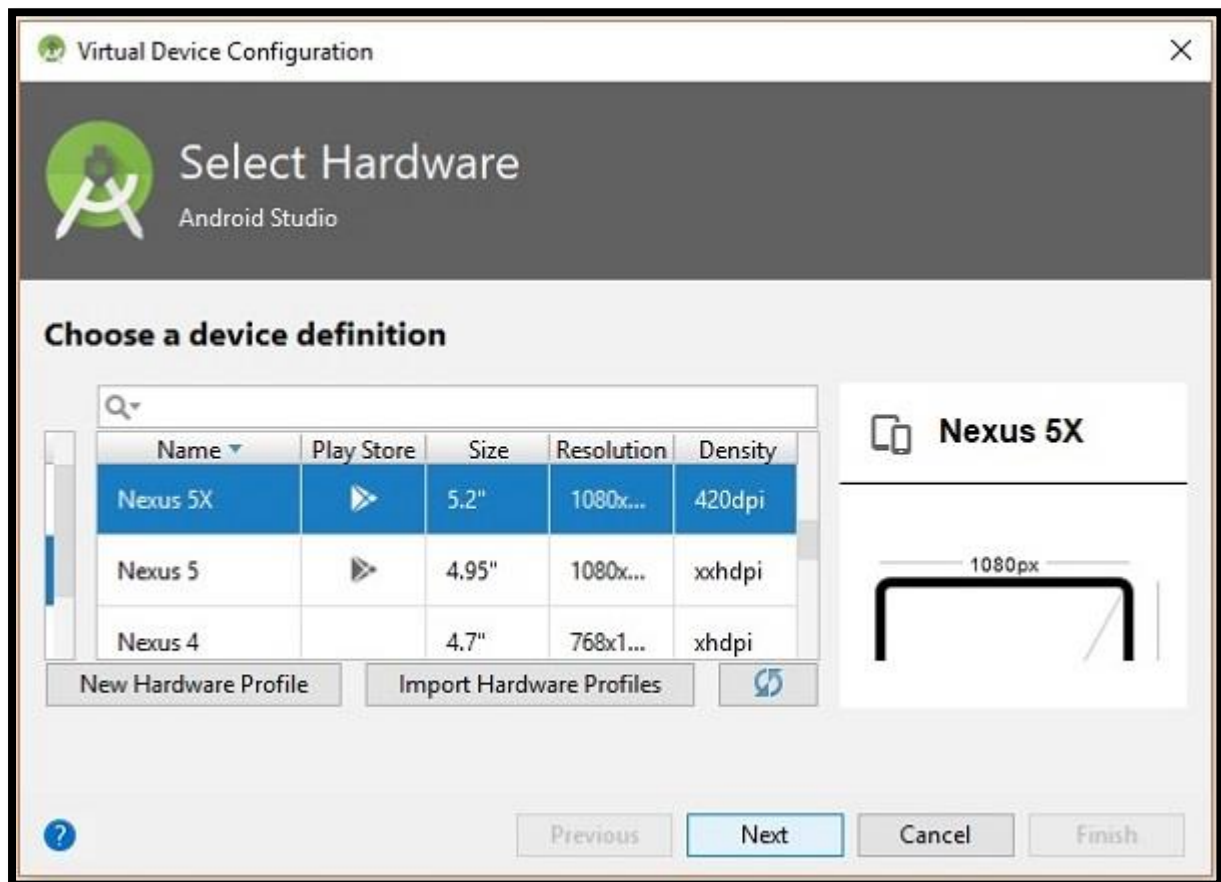
Step 8: Configuring AVD Manager

To configure the AVD Manager click on the respective icon in the menu bar.

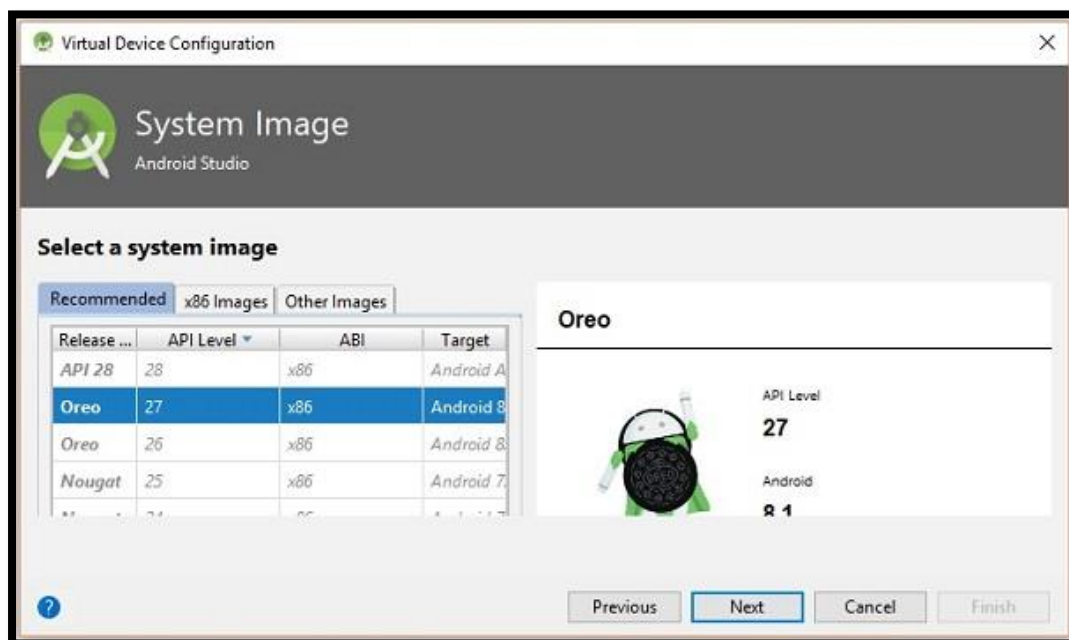


Step 9: Configuring AVD Manager

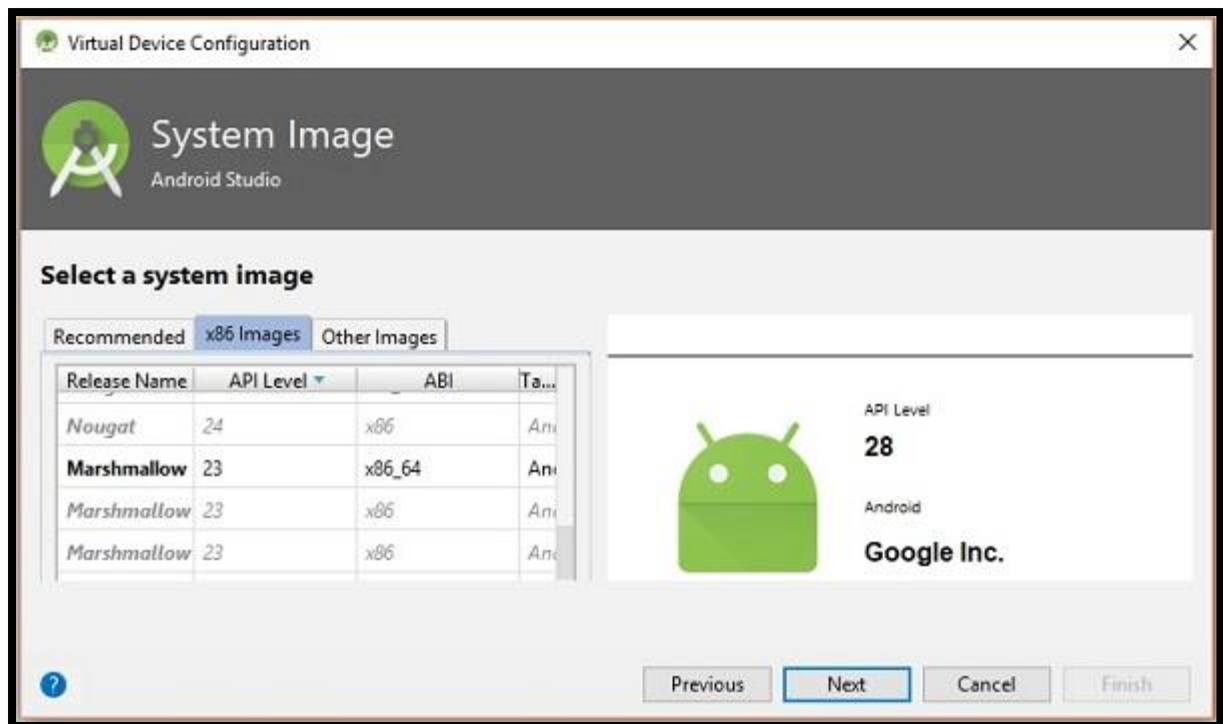
Choose a device definition, Nexus 5X is suggestable.



Click on the Next button you will see a System Image window. Select the **x86 Images** tab.



Then, select Marshmallow and click on next.



Finally, click on the Finish button to finish the AVD configuration.



After configuring your virtual device click on the play button under the Actions column to start your android emulator.



Step 10: Running android

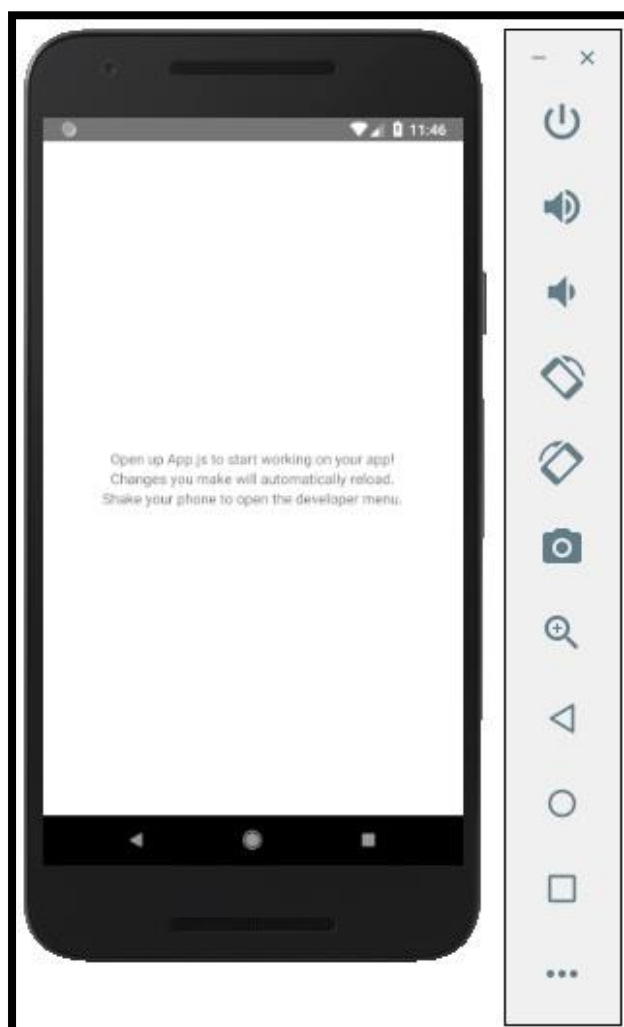
Open command prompt, browse through your project folder and, execute the **react-native run-android** command.

```
C:\WINDOWS\system32\cmd.exe - react-native run-android
C:\Users\Tutorialspoint\Desktop\MyReactNative>react-native run-android
Scanning folders for symlinks in C:\Users\Tutorialspoint\Desktop\MyReactNative\node_modules (534ms)
Starting JS server...
Building and installing the app on the device (cd android && gradlew.bat installDebug)...
Incremental java compilation is an incubating feature.
:app:preBuild UP-TO-DATE
:app:preDebugBuild UP-TO-DATE
:app:checkDebugManifest
:app:preReleaseBuild UP-TO-DATE
:app:prepareComAndroidSupportAppcompatV72301Library
:app:prepareComAndroidSupportSupportV42301Library
:app:prepareComFacebookFbuiTextlayoutbuilderTextlayoutbuilder100Library
:app:prepareComFacebookFrescoDrawee130Library
:app:prepareComFacebookFrescoFbcore130Library
:app:prepareComFacebookFrescoFresco130Library
:app:prepareComFacebookFrescoImagepipeline130Library
```

Then, your app execution begins in another prompt you can see its status.

```
node "C:\Users\Tutorialspoint\Desktop\MyReactNative\node_modules\react-...  
|  
| https://github.com/facebook/react-native  
|  
|  
|  
|  
| Looking for JS files in  
| C:\Users\Tutorialspoint\Desktop\MyReactNative  
|  
| Metro Bundler ready.  
|  
| Loading dependency graph, done.  
| BUNDLE [android, dev] ./index.js 100.0% (481/481), done.  
| ne.
```

In your android emulator you can see the execution of the default app as –



Prob Definition:

We are implementing Music Stream Application, this application help to listen music from external storage. User can listen music without internet access as well as can downloaded music can also added in this application.

User can Paused at any time, Listen Next as well as previous song.

Conclusion:

Hence, Form this experiment we studied about five planes of UXD as well as learn Flutter and React native framework and installation for the same.