**Q1. What platforms are available for developing mobile apps?**

**Ans.**

**Android Studio** - Android Studio is the official Integrated Development Environment (IDE) for Android app development, based on IntelliJ IDEA. On top of IntelliJ's powerful code editor and developer tools, Android Studio offers even more features that enhance our productivity.

To develop mobile apps in the android studio we can use - Java, Kotlin, Flutter framework

**Xcode** - Xcode is Apple's integrated development environment (IDE) for macOS, used to develop software for macOS, iOS, iPadOS, watchOS, and tvOS. Xcode consists of a suite of tools that developers use to build apps for Apple platforms. Xcode can be used to manage our entire development workflow—from creating apps to testing, optimizing, and submitting it to the App Store.

For mobile app development, Xcode supports languages like Objective-C, Swift, Java and with third party support, we can also use C#.

**VSCode -** Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages.

**Q2. Native(Java/Kotlin/Swift/Objective-c) vs Cross Platforms(React-native, Flutter, Cordova, ionic)**

**Ans.**

**Native -** native app development refers to building a mobile app with programming languages and tools that are specific to a single platform. For example, you can develop a native Android app with Java or Kotlin and choose Swift and Objective-C for iOS apps. Native apps are known to deliver exceptional user experience as they are generally high performance. User experience is also enhanced as the visuals are tailored to the platform UX.

* **Pros** -
  + More functionality: we’ll have access to every API and tool provided by the platform that you’re working on.
  + Performance and Great UX: ‍The direct interaction between the code and the underlying resources results in high performance. Also, native apps generally have a better UX.
  + Scalability: the application is more flexible and scalable, due to the “native” tools usage and arrays of tools available. Also fewer restrictions in the architecture and functions
* **Cons-** 
  + Costly: ‍Building native apps can be costly when you need to launch for both iOS and Android. It means you’ll need to deploy two teams working on different platforms.
  + Time-consuming: Native app development is time-consuming as the work done for a platform cannot be duplicated for another. Instead, a separate team is needed to work on the other version.

**Cross-platform -** Cross-platform development points to the process of creating an app that works on several platforms. This is done by using tools like React Native, Xamarin, and Flutter, where the apps created can be deployed on both Android and iOS. While cross-platform development saves time and cost, you risk sacrificing quality in the process. It is difficult to tailor an app that runs optimally on various platforms.

* **Pros -**
  + ‍Less Costly: Instead of having two teams of developers, you’ll need only one to create a cross-platform app. Therefore, you’ll save on the development cost.
  + Faster Development‍: Only a single cycle of development is needed to create an app that runs on multiple platforms.
  + ‍Single Code Base: As the app is created with a single cross-platform development tool, only one code base is created.
* **Cons -** 
  + Slower App: ‍The need for an additional abstraction layer and rendering process makes the cross-platform app slower than its native counterpart.
  + ‍Limited Functionality: Developers may have difficulty accessing smartphone functionalities like the microphone, camera, and geolocation in ways possible for a native app.
  + ‍Limited UX: ‍Cross-platform apps are unable to take advantage of native UX components. Therefore, it can’t deliver the same UX experience that is accustomed to the platform.

**Q3. Core differences among - React-native, Flutter, Cordova, ionic, etc**

**Ans.**

**React-native -** React Native is a JavaScript framework for writing real, natively rendering mobile applications for iOS and Android. It’s based on React, Facebook’s JavaScript library for building user interfaces, but instead of targeting the browser, it targets mobile platforms. React Native makes it easy to simultaneously develop for both Android and iOS. Under the hood, the React Native “bridge” invokes the native rendering APIs in Objective-C (for iOS) or Java (for Android). Thus, your application will render using real mobile UI components.

**Flutter -** Flutter is Google’s portable UI toolkit for crafting high-quality native experiences for mobile. An app built with Flutter does not use native UI components but consists of building blocks called Widgets. Also, there is no bridge between the view and the codebase as in React Native. It provides its own rendering engine, ready-to-use widgets, command-line tools, APIs, and more that make the entire development process easier.

**Ionic -** Ionic is a complete open-source SDK for hybrid mobile app development. It provides tools and services for developing hybrid mobile apps using Web technologies like CSS, HTML5. Apps can be built with these Web technologies and then distributed through native app stores to be installed on devices by leveraging Cordova. With Cordova (and Ionic) you can write a single piece of code for your app that can run on both iOS and Android

Core Differences -

* Code Reusability -
  + **Ionic**: The “wrapped web app” concept ensures that you can easily re-use your codebase for various applications.
  + **React-native**: Also compiles to native defaults but only provides a basic set of components to start with. You have to style most of them on your own, hence more work is required to achieve appropriate styles on both platforms.
  + **Flutter**: it provides a comprehensive toolkit that helps developers create modern apps with ready-to-use widgets, APIs.
* Performance -
  + **Ionic**: Its performance isn’t as similar to React Native or flutter offers since it utilizes web technologies to render an application. This approach significantly decreases the speed. Also, it doesn’t utilize native components and tries to create a native look and feel by using web technologies. Hence the performance is Moderate.
  + **React-native**: The performance it provides is very similar to native applications as it renders code elements specifically to the native APIs. React additionally allows developers to use native modules to write code for complicated operations. The performance is close to native.
  + **Flutter**: The performance provided by flutter is almost native and is the best among the three. Because it has advantages of Dart and there is no JavaScript bridge for starting interactions with the device native components, the speed it offers is amazing.
* Accessing Native Device Features -
  + **Ionic**: Ionic uses Cordova or its own solution, Capacitor, to give you access to native device features. It provides a very decent set of packages to access common native device functionalities like the camera but is very limited.
  + **React-native**: A rich set of third-party packages as well as some built-in APIs for accessing native platform functionalities. However, problems might arise if the developer of the third party library quits.
  + **Flutter**: provides official packages for some of the most common native device features you need access to. And Third-party packages for pretty much any native feature. we can also write and connect real native code if you need to.
* Third-party libraries -
  + **Ionic**: Ionic uses JavaScript for its logic and has great support for Angular. Therefore, benefits from both these ecosystems.
  + **React-native**: it makes use of React and JavaScript and so you will have both their ecosystems available. Apart from that, React Native itself has managed to have a strong community. This means we will find a large number of third-party packages. The drawback is that some third-party plugins are not able to update itself with the changes in React Native.
  + **Flutter**: It has tremendous growth and we can find a lot of discussions and third-party libraries and packages. Despite being relatively new, it’s already quite mature and has a decent ecosystem.

**Q4. Why react native and why not? (Pros and Cons)**

**Ans.**

**Pros -**

* Cross-platform development-

With React Native, one codebase runs on iOS and Android. Only a small portion of the app needs to be customized for each operating system. It provides faster, more reliable performance than a hybrid. It saves time and resources.

* Development Speed

React-native being cross-platform, in reality, the translation between operating systems isn’t perfect, it still requires some platform-specific tailoring but there’s still a huge chunk of the codebase shared between platforms.

That cuts development time for an app by as much as half while still supporting multiple platforms.

* Native-like performance

React Native apps render their UI using native APIs. As a result, their performance is in most cases on par with native iOS/Android apps. Moreover, the JavaScript logic is executed on a separate thread without blocking the UI rendering.

* Cost of development

There are clear savings in both time and money when one app can be repurposed to cover all devices. There are fewer overall development costs and lower up-front investment. Maintaining one code base lowers long-term maintenance expenses.

* Hot reloading

React Native supports Hot reload and Live reload. A system where we can see our changes live without having to re-run or reload the app. Hot reloading is a feature that reflects changes while maintaining the app’s current state.

* Large developer community

React Native has huge community support as it uses Javascript. So if you are working with React Native and you find yourself surrounded with bugs then we can take help from the active community.

**Cons -**

* Performance

If your app has a lot of complex interactions, this can noticeably degrade its performance. The more tabs, navigations, controls, animations, third-party libraries your app has, the slower React Native becomes.

App size increases heavily on using third-party libraries.

React native offloads all complex functions to the Javascript thread which causes a delay in updating the app’s user interface. And when that happens, the app stops responding to user inputs and performance lags are extremely apparent.

* “Reusable” codebase

The “write once, use anywhere” motto isn’t entirely accurate. Developers have to tailor certain parts in the app for each platform. In practice anywhere from 60-90% of the codebase can be fully shared and rest needs to be configured according to the platform.

* Feature lag

Google and Apple constantly introduce new features with updates to their OS.

React Native team does its best to bring new functionality to the platform. But that takes time.

* Native Development knowledge Still Needed

To solve some of the issues we need the help of native modules. But implementing them requires Java/Objective-C/Swift expertise. So you’ll still occasionally need some help from native developers.

The implementation of some more advanced features might still require help from native developers.

* Difficult animations

Animations are difficult and complex when you are working with React Native.

* Startup time

Whatever code you write in React Native is compiled and bundled into a JS bundle which the app loads at the startup. This JS bundle holds your app’s logic. So the more third-party libraries you use the more is the size of the JS bundle and the more time the app takes to startup.

**Q5. Execution process of react native code.**

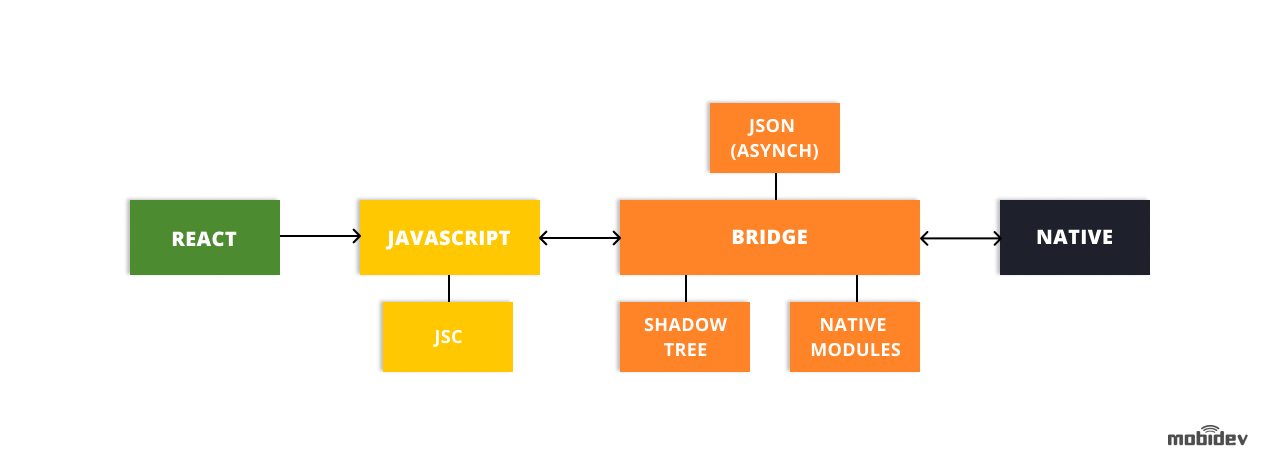
**Ans.**

If we consider the big picture, there are three parts to the RN platform:

* **Native Code/Modules**: Most of the native code in case of iOS is written in Objective C or Swift, while in the case of Android it is written in Java or Kotlin. But for writing our React Native app, we would hardly ever need to write native code for iOS or Android.
* **Javascript VM**: The JS Virtual Machine that runs all our JavaScript code. In the case of iOS, React Native uses the JavaScriptCore provided by the iOS platform. In the case of Android, React Native bundles the JavaScriptCore along with the application. This increases the app size.
* **React Native Bridge:** React Native bridge is a C++/Java bridge which is responsible for communication between the native and Javascript thread. A custom protocol is used for message passing.

Bridge work as Asynchronous, Batched, Serializable.

Now, whenever the React Native app is launched, the first item to be loaded is the native entry point. The Native thread spawns the JS thread which runs the bundled JS code. The JS code has all the business logic of the application. The Native thread now sends messages via the RN Bridge to start the JS application. Now, the spawned JS thread starts issuing instructions to the native thread via the RN Bridge. The instructions include what views to load, what information is to be retrieved from the hardware, etc.



When a React Native application is launched, it spawns up the following threading queues.

**Main thread (Native Queue)** - This is the main thread which gets spawned as soon as the application launches. It loads the app and starts the JS thread to execute the Javascript code. The native thread also listens to the UI events like 'press', 'touch', etc. These events are then passed to the JS thread via the RN Bridge.

Once the Javascript loads, the JS thread sends the information on what needs to be rendered onto the screen. This information is used by a shadow node thread to compute the layouts. The shadow thread is basically like a mathematical engine which finally decides on how to compute the view positions. These instructions are then passed back to the main thread to render the view.

**Javascript thread (JS Queue)** - The Javascript Queue is the thread queue where the main bundled JS thread runs. The JS thread runs all the business logic, i.e., the code we write in React Native.

**Custom Native Modules** - Apart from the threads spawned by React Native, we can also spawn threads on the custom native modules we build to speed up the performance of the application. For example - Animations are handled in React Native by a separate native thread to offload the work from the JS thread.