Types of Mobile Apps – Part 2

In this lesson, we will continue the discussion, from the previous lesson on different types of mobile apps.

We'll cover the following

Okay, up to this point,you have learned about the two different types of mobile apps and the popular technologies that are leveraged to build them. Now, when we talk about Hybrid apps, the first question that pops up in our mind is: *Why do we need this type of app when we already have native apps? They are performant and have a consistent UI, so why would any business want to compromise user experience by offering its service via a hybrid app?*

In the mobile app development universe, there are a few pain points that come along with the native app development, and businesses have to turn towards Hybrid apps to overcome those pain points. Let’s find out what they are.

Cross-platform app development pain points - The need for hybrid apps#

Earlier we discussed that when writing Native apps, we have to develop dedicated apps for every platform be it Android, iOS, Windows, Blackberry, or any other OS. Developing and maintaining a dedicated mobile app for every OS is the biggest pain point of cross-platform app development. Every OS supports a specific set of technologies to build apps for them. There is no common ground and no common technology that is supported by all the platforms. Due to the need of having a presence on multiple platforms, developers have to first educate themselves on various technologies before they get down to the implementation of any sort.

Businesses have to set up dedicated teams for every platform. A team building an Android app has to be proficient in *Java, Kotlin* or *C++*, and a team building an app for iOS has to be proficient in *Swift*.

Even if the reluctant developers go through the steep learning curve and build and launch their apps on these platforms. What's the guarantee that, in future, a different OS won't pop-up that supports a different set of technologies to build apps for its platform?

Naturally, when starting up, we do not have enough resources (*developers + money*) to set up dedicated teams and codebases for every platform. We need a common codebase, something portable, something that we could build once and run everywhere.

This led to the emergence of Hybrid apps. Since, these apps are developed using open web-based technologies like *HTML5 and JavaScript*.

Developers working in the modern web development space have this skill set already, and they do not have to go through a steep learning curve to start building these apps. Any developer with the modern web development skill set can start writing code without going through a daunting learning process. With Hybrid apps, businesses do not need dedicated teams for different platforms. The same codebase can be deployed on multiple platforms with minor modifications. These apps are easy to build due to the familiarity with the tech. This saves time and money.

So, building Hybrid apps is the way to go right? I am just starting up, my team is small, and I have limited resources. Why would I want to write a dedicated app for every platform? I should pick the Hybrid app approach, right?

Well, I wish the answer was that straightforward, and I could always say yes!! As I've said over and over throughout the course, there is no silver bullet or no one size fits all. Every tech has a use case, and it comes along with its pros and cons. Hybrid apps are no different.

Issues with Hybrid apps#

Hybrid apps are not as performant and smooth as Native apps as they run inside a Native container and talk to the underlying OS via a middle layer. This slows down their performance a bit and introduces lag.

Although a few frameworks and ecosystems claim to be as performant as Native apps, sometimes even better, marketing is one thing and running an app in production achieving the same performance as Native apps is another.

In the past, a few of the businesses have tried to adopt the Hybrid app single codebase strategy to deploy their apps across platforms but have eventually reverted to the Native app approach to achieve the desired user experience.

Here are a few examples:

Real-life examples#

Airbnb ditched react-native for native tech#

In a series of blog posts (https://medium.com/airbnb-engineering/react-native-at-airbnb-f95aa460be1c), Airbnb engineering shared their experience of developing their mobile app with *React-Native*.

They built their desktop website using *React JS*. Hence, they considered React-Native as an opportunity to speed up the app development process by having a single codebase as opposed to having multiple codebases for different platforms.

They spent a couple of years working on it and eventually abandoned React-Native for the native technology. They faced performance issues specifically during the app initialization and initial render time, with app launch screen, when navigating between different screens. They also experienced dropped frames.

They had to write several patches for React-Native to get the desired Native functionality. They found some of the trivial stuff, that could be easily done with the Native tech, quite difficult to pull off with React-Native.

The lack of type safety in JavaScript made it difficult to scale, and the development process turned out to be difficult for engineers who were used to writing code in languages with default type-safety checks. The lack of type safety made code refactoring extremely difficult.

For a full account of their experience read React Native at Airbnb (https://medium.com/airbnb-engineering/react-native-at-airbnbf95aa460be1c)

Udacity abandoned React Native#

Here is another instance, where the Udacity mobile engineering team abandoned React-Native due to the increased number of Android-specific features requested by their users. Their Android team was reluctant to go ahead with the Hybrid app approach, and the long-term maintenance costs of the React-Native codebase were high. They also faced UX consistency issues across the platforms. For a full account of their experience, here you go (https://engineering.udacity.com/react-native-a-retrospective-from-the-mobile-engineering-team-at-udacity-89975d6a8102).

Facebook admitted counting big on HTML5 for their mobile app was a mistake#

This is back in 2012. I know it's been a while, and technologies have matured a lot. Still, I felt I should add this instance.

Facebook admitted that they made a big mistake investing too much time and resources writing their mobile app with HTML5 instead of using the Native tech. Their mobile strategy relied too much on open web technologies. Here is a full account on VentureBeat (https://venturebeat.com/2012/09/11/facebooks-zuckerberg-the-biggest-mistake-weve-made-as-a-company-is-betting-on-html5-over-native/).

With this, we have reached the end of the lesson. Here are a few interesting reads:

Who Will Steal Android From Google? (https://medium.com/@steve.yegge/who-will-steal-android-from-googleaf3622b6252e)

The Story Of Firefox OS (https://medium.com/@bfrancis/the-story-of-firefox-os-cb5bf796e8fb)

In the next lesson, I'll talk about how to choose the right mobile app type for our use case. Hybrid or Native?

← Back
Types of Mobile Apps – Part 1

Next →
Choosing Between a Native and a Hy...

✓ Mark as Completed

Report an Issue