Going Native with React Native

Nitin Rajpal

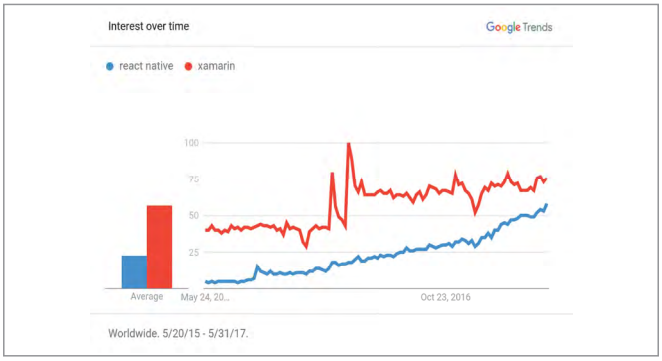
Abstract

The exponential growth in the adoption of mobile apps can be attributed to the increasing popularity of iOS and Android platforms. However, developing and maintaining an efficient app is a complicated process and requires a significant amount of time and money to be invested. The improvements, periodic updates, and fixes are critical for the overall success of the app in the long run. The purpose of this paper is not to compare the two platforms, but rather focus on the ground work done so far with React Native at Coforge. In this paper, we would like to explore the contours of React Native mobile app development framework. The objective will be to share insights and learn from our experience of deploying React Native for our clients.

Mobile App Development: The Available Choices

The last few years have seen a furore of digital engagement about the pros and cons of hybrid apps. Through blogs, analyst reports, forums, and other mediums, users have shared their opinion on this never-ending debate. Though there were different platforms and tools that showed flexibility in shifting to hybrid app development, the limitations and flaws in each platform seemed to get in the way. These challenges were related to user experience (UX), user interface (UI), and performance issues. We were always confronted with the dilemma about how to choose the best platform (specially) when targeting multiplatform or cross (x) platform apps.

Below is the graph showing the popularity of two major cross-platform apps, which are similar to the native look, feel, and performance. The trend is shown for the last two years.



React Native: Seeding the Future of App Development

React Native is an open source cross-platform mobile application development framework, which is emerging as the future of mobile app development. It is built on top of a React framework that is used in web development. The business logic followed is that API is live in JavaScript while the app UI is completely rendered as native views. JavaScript components are built using a set of (standards-compliant ECMA Script code ES7, ES6, and ES5 standards) primitives that are backed by actual native iOS or Android components.

As the UI is rendered in native, React Native has an upper hand to get a high degree of performance and to adhere to UI guidelines of the platform and best practices. React Native also helps you to:

- Write platform-specific code, logic, and styling
- Use platform-specific libraries in your React Native code (using bridge)

React Native: Benefits

Fast-paced Development

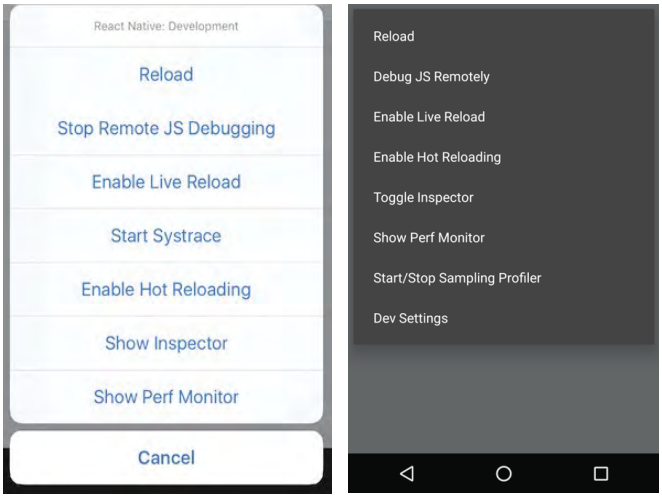
Testing your code on a simulator is nothing short of a nightmare for any developer. Every slight change, even a shift in few pixels, requires the code to be recompiled. This results in slow development, specially in a bulky codebase where compilation is a burden.

However, React Native's goal is to provide the best possible developer experience. A large part of achieving this goal is to reduce is the response time it takes between saving a file and being able to see the immediate changes in the code. Let us look at the benefits:

Instant Code Updates: Instead of recompiling your app every time you make a change, you can reload your app's JavaScript code instantly. In iOS simulator, press R and on Android emulators tap R twice. This helps save a lot of time since the other platform code needs to be recompiled, which can take up to five minutes or more (in slow machines). In React Native, it only takes a few seconds.

Hot Reloading: A common development scenario is to work on a feature that is multiple screens away from the launch screen. Every time you reload, you must click on the same path repeatedly to get back to your feature, making the cycle multiple-seconds long.

The idea behind hot reloading is to keep the app running and to inject new versions of the files that you can edit in runtime. The main aim of hot reloading feature is to make a new code available in less than a second, even as the app matures and grows. You can enable this feature from the Developer Menu (screenshot below):



If you closely check the screenshot, you will find another option named “Enable Live Reload.” It is used to reload or refresh the entire app after a file changes. Live reloading would restart the app and load the app back automatically. Relatively, hot reloading only refreshes the files that were changed without losing the state of the app.

Using Diff Algo for Optimum Performance

Since React Native is an extension to React.JS, they follow the same concept for updating UI (virtual DOM). As a developer, you do not have to care about updating your UI. You declaratively render your UI based on state, and React uses a diffing algorithm to send the smallest amount of changes necessary over the bridge. All this happens in the background thread, keeping the UI thread free.

This is an exclusive feature that is not provided even by native platforms’ software development kit (SDK) (leaving aside the table or list view in iOS and Android platform) but is supported out of the box in React Native. This helps React achieve optimum performance level.

Flexbox for Layout

Compared to iOS Auto Layout, React Native uses Flexbox style code (similar to CSS for HTML), which is far easier to understand and to maintain. You need not manually position and size the views in your application the code ends up being more concise. We tested our app in a number of different device sizes and they looked fine.

App Development: Identifying Bottlenecks and Following Best Practices

Developing an app is extremely complex as compared to a desktop app. Specially, building one that involves frequent releases, different kinds of platforms, and support for multiple devices. Adopting a few best practices, right choice of components, and frequent reviews can help identify the issues early in the development cycle. This makes them easy to trace and fix.

Thumb Rules

- Run the profilers (to catch performance and leaks issues) early on in your code. This process helps you save time by finding issues early in the development cycle and giving you ample time to refactor your code in the early stages (which is less complicated as compared to doing it later).
- Incorporating instruments into your workflow during the beginning stages of the app development process can save you time by helping you find issues early in the development cycle.
- Always run profiler in ‘Device.’ Using it on a Chrome simulator can give you some hints, but this may change on the actual device.
- Run the profiler in release mode instead of debug mode. Running on debug mode can give you hints but it adds a lot of overheads to your code (making it slow).
- Try running your performance on the oldest device you want to support. This could be iPhone 5 or any Android phone with 1GB RAM or less.

Reducing Render Cycles

In real world, your screen (or page) might not have a single view, but a collection of many different views, creating a tree like structure for your DOM. When a component state changes, React Native decides whether an actual DOM update is necessary by comparing the newly returned element with the previously rendered one. When they are not equal, React Native will update the DOM.

Overriding the lifecycle function ‘should Component Update[ ]’ is key here. You can speed up render cycles when you are sure that your component does not need an update. If the default implementation returns true, returning false would skip the rendering process of that component. This can be achieved by checking the nextProps, nextState parameters.

Animation API

Basic stock animations like screen transitions are available and perform well out of the box. However, for

making custom animations such as chart animations, your code should not depend on the third party libraries. Based on our experience, frame rate drastically slows down and in some cases drains the battery faster than expected.

Navigation

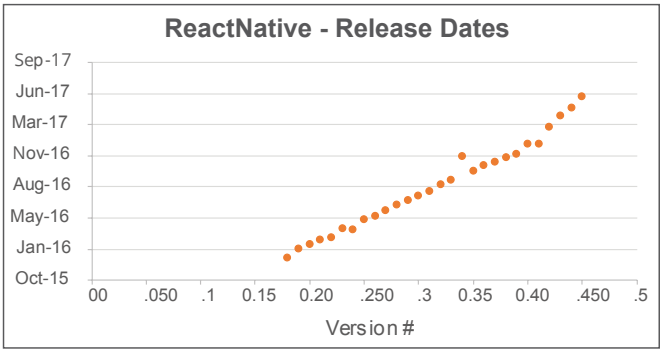
You will find a large number of components (both third party and Facebook recommended) for implementing 'Navigation' in your app. Navigation is the key control of your app skeleton, which provides critical components (nav/tab bars, menus). Any latency in this component can bring the UX of the app down. We have used the following components, which provide the best experience in terms of dev and good UX:

- react-native-navigation (Native implementation)
- react-navigation (JavaScript implementation)
- react-native-router-flux (JavaScript implementation)

Frequent Release cycles

As React Native is evolving at a very fast rate, the libraries being used are also constantly being updated. As a developer, you need to make sure to update your app with the latest version of React Native and supported libraries. Facebook releases new React Native version in a time frame of two to three weeks. With every release, there is a high risk of a codebreak, which essentially requires fixes. Development teams generally tend to spend a decent amount of time in fixing these bugs in order to upgrade to the latest version.

Alternatively, the new releases come out extremely fast. Below are some dates of fixes in the past:

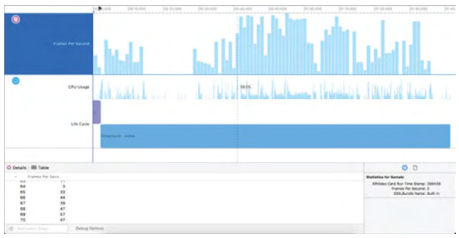
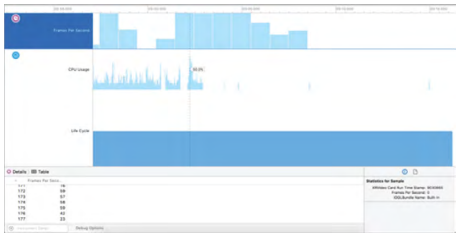


App Performance Monitoring: A Case in Point

In one of the apps that we are developing for our retail customer, we tried running a few performance monitor tests on the current app to see if there were any performance/bottleneck issues in the app. In order to ascertain the full potential of the app, we used the app continuously and traversed to various screens checking the hierarchy (5-8 views) with a very long list of views.

The code we built passed test on various parameters, as it implements the best architectural guidelines and standards. While developing apps in React Native, you have to consider the basic building blocks that are used along with the aforementioned thumb rules to apply.

Given below is the snapshot of the app performance monitoring:

Device	iOS - Core Animation Report
iPhone 7+	
iPhone 5	
Findings	There is a high CPU usage at some points, this is because we were scrolling through a long list (around 1000+ records from server) which takes long render cycles to process.

Device	Android Monitor Report
Moto Z	
Nexus 5	
Findings	As expected, Nexus 5 is an older device, therefore CPU usage is rather high compared to Moto Z. The expected report results were well within range.

React and React Native make for a good framework of choice for both mobile and web. If you are targeting only mobile app development, then it helps you save time during code sharing (sharing business logic), reducing the overall effort and cost for development and maintenance.

The Coforge Thought Board: Get ahead of the curve with React Native

What is the Tri-fold Focus of React Native?

DOM abstraction

Simplified programming

Better app performance

How React Native Allows for a Seamless Performance?

Quicker load time

More efficient in terms of modification

Short development cycles

Live updates and CodePush

Why Is Reach Native the Future of Hybrid App Development?

- Cross-platform capability
- Allows hybrid apps to render natively
- Technologically superior
- Offers third party plug-in compatibility

Bridging the Gap

Over the last few years, a steady growth of interest in the React Native framework has made it a leader in hybrid mobile development. It is not just the ease of development that factors its progress, but also the richness of the ecosystem and the quality of apps.

A hybrid app is more often than not at the crossroads of user experience and ease of development. Facebook's React Native bridges that gap between the ease of development of an app and the performance of a native app.

Getting started with React Native is easy, specially for JavaScript users. Using identical principles as React, React Native maintains a virtual DOM that greatly increases the render performance of mobile apps.

Using an efficient diffing algorithm, the hot reloading capability lets you modify the application code without recompiling the application itself. This saves a lot of cost and time, leading to less memory usage and a smoother experience. You can use the same generic code for both mobile versions.

React Native focuses on a highly responsive user interface while giving the look, speed, and functionality of a native application. It is also easier to write components that are similar in function but adapt to different OS. Using partial implementation, elements can be laid out and basic features can be easily implemented. In addition, it is faster to develop applications if you make use of the vast library of React components that are available under open source.

There has been an incredible progress in the past few years, with React Native becoming the top favorite among JavaScript users. Without a doubt, React Native will continue to grow in the years to come.

About the Author

Nitin Rajpal, Enterprise Architect, leads the Mobile CoC, Digital services practice in Coforge. He brings more than 15 years of experience in giving enabler solutions on various mobile technologies such as cross platform technologies, platform SDKs, MEADP, MBaS and MADP platforms. He remains instrumental in designing solutions in Mobility domain for various verticals such as Travel, Insurance, Manufacturing, etc.

For more information, contact information@coforgetech.com

Coforge is a leading global IT solutions and services organization which believes that real transformation cannot be driven by thinking in technology terms alone. With a mission to “Transform at the Intersect” it aims to bring both deep domain and deep emerging technologies expertise to achieve real-world business impact. A focus on very select industries, a detailed understanding of the underlying processes of those industries and partnerships with leading platforms provides us a distinct vantage. We leverage AI, Cloud and Insight driven technologies, allied with our industry expertise, to transform client businesses into intelligent, high growth enterprises.

Learn more about Coforge at: www.coforgetech.com

Stay connected:

Coforge