MSc in Computer Science

Team Project

Speed Testing Report

TrailSeek

**React Native Application Architecture**

React Native is JavaScript based where, via a bridge, the React Native component communicates with the Native thread. We primarily have three key threads running the framework. Shadow Tree Thread, UI Thread, and JavaScript Thread (also known as native thread).

Therefore, the architecture of React Native has two essential parts:

* Built on Java, Swift, or Objective-C, the Native Part
* JavaScript-built React Native section.

In order to enhance the efficiency of your React Native application, here are some standard practices that we have adopted while building our application to ensure that the application is optimized and performs quickly. Also some practices are mentioned that are that may be better in the future.

**Issue of Memory**

For native apps with a lot of unnecessary processes running in the background, this is one of the most common problems.

We have discovered leaks by the use of Xcode  
XCode -> Product -> Profile (⌘ + I keys)   
Afterwards, when it shows all templates, choose leaks

Find leaks with Android Studio

Execute react native and Android Studio

From the menu,

Go to Tools -> Android -> Enable ADB Integration menu  
Go to Tools → Android → Android Device Monitor menu

Click Monitor -> Preferences when the Android System Monitor appears. Perf Monitor is a good option to use for the monitoring of Android leaks.

We also used scrolling lists like FlatList, SectionList, or VirtualList instead of Listview to avoid memory leaks in applications. The scrolling list also helps to smooth the infinite scroll pagination, so it is recommended to consider it if you are creating an application with a pull-to-refresh function and large data points.

**Image Size Reduction**

Images are a major problem with heavy memory consumption. Optimization of images in the React Native framework is very significant.

Using pictures that are smaller in scale.  
Using PNG instead of t JPG.  
Converting the format for pictures to WebP.

Why WebP format?

WebP images will speed up the loading time of your image to 28 percent. Reduced package size for CodePush with .webp format by 66 percent Helps to minimize binary sizes for iOS and Android by 25 percent with the format .webp React Native JS Thread feels so much quicker than Navigator improvements are so much easier.

locally Caching the images

For loading images more quickly, image caching is critical. As of now, React Native only offers image caching support on iOS.

There are several npm libraries available for Android out there that help address the problem of image caching.

**Avoiding Needless Renders**

Among the most common issues is not to first enforce this diff state, lifecycle and props yourself-you should first make sure that whether your components should change or not and make sure that not too much excessive work is transferred to Reconciler, it will probably drop the FPS of your JS thread. Outspread instead of Component from PureComponent, however.

**JSON Data Optimizing**

Mobile applications often need a service or remote URL to load services, and programmers fetch requests to pull data from that server to perform those acts. In a JSON type with some kind of compound nested objects, the fetched data from private as well as public APIs returns. Usually, for local offline access, most programmers store the same JSON data and the output suffers as JS applications slowly make JSON data. So before rendering, what I would like to advise you is to transform raw JSON data into simpler objects.