CPSC 490 Final Report

Exploring The Potential of React Native in Modern App Development

William Sun

Advisor: Ruzica Piskac

May 4, 2017

Yale University

Class of 2017

williamjinsun@gmail.com
Table of Contents

1. Abstract
2. Background
3. Design
4. Development
5. React Discussion
6. Further Implementation
1. Abstract

Mobile app development has resulted in an abundance of software innovation, in terms of asynchronous I/O, client-side logic, and reusable code. In this project, I build an iOS movie app similar to IMDB using React-Native, Facebook's library for mobile app development, in order to explore mobile development strategies and technologies, while putting to use the user interface theory, software development and graphic design skills I've learned during my undergraduate time. This project makes use of Sketch for design, Facebook’s React-Native library for front end development, and NodeJS API calls to IMDB for data.

2. Background

Mobile technology is innovative and more important than ever. In 2014, the number of mobile devices surpassed desktop devices, and in 2015, mobile usage eclipsed desktop usage. The growth of mobile is even more pronounced in developing nations, which will only increase the growth curve of mobile as these nations mature.

In the last few years, an explosive growth of front end tools has taken off based on the rapid increase in mobile usage and the opportunities for software development to fill the gap in supply and demand. New tools, preprocessors, frameworks, libraries, and heuristics come out every day. However, throughout the changing chaos of modern front end development, several important paradigms are here to stay.

Firstly, the increasing power of personal devices has resulted in logic shifted to the client side. This has led to JavaScript’s rise, as more and more logic is shifted from server-side PHP to client side Javascript. Javascript, used for client logic, was adapted to more and more powerful usage cases, and has long since become the dominant front end technology.

However, as more developers became reliant on Javascript, it was adopted to be used on the server side as well in the form of NodeJS, which is quickly becoming a major player. NodeJS has proven superior to previous backend technologies in many cases, and Uber, Groupon, Netflix, PayPal, LinkedIn and other flexible companies are pivoting to use it.

Finally, the most trend is the consolidation of front-end technologies into a clear, usable framework organized by the largest tech companies. Both Facebook and Google released their
own libraries\(^1\), AngularJS and ReactJS. Both are extraordinary in terms of their scope of vision in redesigning front end development, but currently ReactJS seems to be winning out. Consequently, I have chosen to discuss React in this paper, and how it has massively improved the rendering speed and development costs, as well as some new complications.

\(^1\) I use ‘library’ and ‘framework’ interchangeably here – although React is technically a library, the ecosystem has a clear structure on how you develop with it, so I consider it a framework)
A few graphs of JavaScript’s current dominance

GitHub Statistics Report, 2016

StackOverflow Developer Survey 2016 - Front End
## StackOverflow Developer Survey 2016 - Back End

<table>
<thead>
<tr>
<th>Language</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>JavaScript</td>
<td>55.4%</td>
</tr>
<tr>
<td>SQL (or SQL Server)</td>
<td>48.1%</td>
</tr>
<tr>
<td>Java</td>
<td>36.3%</td>
</tr>
<tr>
<td>C#</td>
<td>30.9%</td>
</tr>
<tr>
<td>PHP</td>
<td>25.9%</td>
</tr>
<tr>
<td>Python</td>
<td>24.9%</td>
</tr>
<tr>
<td>C++</td>
<td>19.4%</td>
</tr>
<tr>
<td>C</td>
<td>15.5%</td>
</tr>
<tr>
<td>Node.js</td>
<td>17.2%</td>
</tr>
<tr>
<td>AngularJS</td>
<td>17.9%</td>
</tr>
<tr>
<td>Ruby</td>
<td>8.9%</td>
</tr>
<tr>
<td>Objective-C</td>
<td>6.5%</td>
</tr>
</tbody>
</table>

### GitHub’s Most Popular Repositories, Page 1

1,920,176 repository results

<table>
<thead>
<tr>
<th>Repository Name</th>
<th>Language</th>
<th>Stars</th>
</tr>
</thead>
<tbody>
<tr>
<td>freeCodeCamp/freeCodeCamp</td>
<td>JavaScript</td>
<td>266k</td>
</tr>
<tr>
<td>twbs/bootstrap</td>
<td>JavaScript</td>
<td>110k</td>
</tr>
<tr>
<td>vhf/free-programming-books</td>
<td></td>
<td>64.6k</td>
</tr>
<tr>
<td>facebook/react</td>
<td>JavaScript</td>
<td>63.9k</td>
</tr>
<tr>
<td>d3/d3</td>
<td>JavaScript</td>
<td>63.9k</td>
</tr>
<tr>
<td>getify/You-Dont-Know-JS</td>
<td>JavaScript</td>
<td>57.9k</td>
</tr>
</tbody>
</table>

It continues on like this.
3. Design

I used Sketch 3, a professional app for digital design, to plan the basic layout of my app so that I could better envision my final product and increase my development speed. Sketch was very useful to quickly getting a sense of what the final vision would look like. I learned to use it through YouTube videos, which were very easy to follow and understand.

I began with a basic layout for my application, and then planned how the general component hierarchy would look. Then, I began developing a rough prototype so that I could get a feel for
how React development actually felt like before going back and creating a more detailed mockup. This plan would help me in organizing my project structure and files.

4. Development

After planning out the application, I delved into Facebook’s React Docs. These were immensely helpful, but I still needed to put it together. I began working on a minimum viable product to see if I needed to go back and change my idea.

In order to get setup, I followed Facebook’s documentation, which involved installing React-Native’s command line interface and Node Package Manager. I also used ESLint to debug my code. The GitHub page has a handy tutorial.

All in all, I was surprised at how simple the set up process was. React CLI handles most of the packages you need, including Babel (JSX compiler), jest, yarn, CSS preprocessors, etc. These are all packages that I previously had to install manually with other frameworks.

I began with a simple ListView. (An important thing to note about React-Native is that components translate into both iOS and Android native components, meaning they are just as fast as native code, but also buggy sometimes.) The ListView was my basic body, and I began to work with a ripoff of IMDB called ‘TMDB’ and their API.

Unfortunately, TMDB does not provide the current most popular box office movies. I wrote a script (scrapeIMDB.js) to scrape IMDB’s main page for it’s top selling box office movies, and then save the data in an JSON file. I then made a call to TMDB for each movie’s data.

Unfortunately, I could not integrate the script with my react-native code at this time, since some of the modules used are unsupported. The script is included in my directory though.
This rough application seemed to work decently. It was time to polish up my application and go back to the design board.

I came up with this much better looking mock, and began implementing it. I soon realized how much more difficult it was to make a more polished product, especially in regards to actions. It’s not just a matter of adding a few more components, you have to ensure that they all work together and move smoothly. It was much more complicated than I thought moving to this stage. In order to make sure the scrolling worked smoothly and snapped into place, I went
through several options, but finally settled on a library ‘react-native-snap-carousel’. It did not work as I wanted so I had to adapt it a bit.

You can run the final app yourself from the GitHub page with a few simple lines. After cloning the repo, just install React and the modules. For Macs with Xcode, it’s just a few lines of code:

```bash
brew install node
brew install watchman
npm install -g react-native-cli
cd moviesApp/
react-native run-ios
```
5. Technology Discussion

What is React?
React is a Javascript Library that helps you build User Interfaces. It’s almost always tied with JSX, a way of including HTML code inside a Javascript file (instead of the other way around!)

So, how does React work?
First, you build UI components with Javascript. Then you pass them into React’s ‘app.render’ function, and voila, React renders it! Behind the scenes, React translates your code into actual HTML/JS/CSS.

Why is React important?
React is not just the newest tool or library. It represents (and introduces) fundamental changes in front end development theory – truly reusable, modular components and Shadow DOM rerendering.

Reusable Components
Traditionally, UI pieces are separated according to MVC (Model, View, Controller). React does away with that system. While it seemed to make sense that software would be more organized if you separated these, in reality, you would still have to go back and forth between all these different files when you ran into a bug or wanted to add a new component on a webpage.

The core idea behind react is that separating HTML, CSS, and JS is not actually abstracting away your concerns – it’s only a separation of technologies. Instead, React separates code based on modular components.

Each React component’s structure, presentation and logic live in a single JS file. This vastly simplifies file structures and interdependencies. Components can then contain other components, and can be used in multiple places.

For example, in React I can define a Header component like so:
const Header = () => {
    getSOME_NAME()
    ..... 
    return (
        <View style = {somestyle} >
            <Text>Hello {getSomeName() } </Text>
        </View>
    
    )
</Header>

somestyle = {
    height: 100,
    ...
}

Then I can use this component I’ve defined anywhere else! I can pass properties, functions, and even other components into this Header component.

The Shadow DOM
To put it simply, React only re-renders components that have changed, resulting in massively improved loading times. React does this by keeping a virtual representation of HTML in memory. Every time state changes, and we get a new HTML tree that needs to be taken to the browser’s DOM, instead of writing the whole new tree React will only write the difference between the new tree and the previous tree (since React has both trees in memory).

In the case of React-Native, React’s internal tree representation will generate iOS UIKit code instead. The same process goes on behinds the scenes. Your React code is translated into the Virtual DOM tree, and only re-renders when necessary.

6. Conclusion
React-Native takes everything great about React and lets you use it to write mobile applications. All of React’s cutting edge software developments, including Shadow Dom rerendering and modular components, have been translated to iOS and Android phenomenally.

At the UI level, React’s translation of components into native components makes the app work seamlessly. From a development perspective, the extra modularization of structure, styles and logic is invaluable to organizing and developing a complicated app. Additionally, the setup and development process is painless and seamless. This is a framework that truly makes your development better, instead of just complicating it.

1 https://octoverse.github.com/
2 http://stackoverflow.com/insights/survey/2016#technology
3 http://stackoverflow.com/insights/survey/2016#technology
4 https://github.com/search?q=stars:%3E1&s=stars&type=Repositories
5 https://facebook.github.io/react-native/docs/getting-started.html
6 https://github.com/roadhump/SublimeLinter-eslint
7 https://github.com/WillTheSun/MovieApp