A comparative analysis of developing Android mobile applications using the native Android language Java, compared to using the cross-platform development framework React-Native.

Robert Rigler Student ID:4939377

Computer Science

School of Computing, Engineering and Mathematics, Coventry University

Supervisor 1: Dr. Faiyaz Doctor

Supervisor 2: Dr. Dianbasi Nkantah

Date of Submission: 02/05/2017 (2nd May 2017)

# 

# Table of Contents

[Table of Contents 1](#_Toc481523121)

[300COM / 303COM Declaration of Originality 4](#_Toc481523122)

[Statement of Copyright 4](#_Toc481523123)

[Statement of Ethical Engagement 4](#_Toc481523124)

[Abstract 5](#_Toc481523125)

[Keywords 5](#_Toc481523126)

[Introduction 6](#_Toc481523127)

[Literature Review 8](#_Toc481523128)

[Performance 8](#_Toc481523129)

[Development-Time 9](#_Toc481523130)

[Cost 10](#_Toc481523131)

[Stability 11](#_Toc481523132)

[Method 13](#_Toc481523133)

[Method Description 13](#_Toc481523134)

[Performance 13](#_Toc481523135)

[Development-Time 13](#_Toc481523136)

[Variables and Specifications 14](#_Toc481523137)

[The application 14](#_Toc481523138)

[Feature List 14](#_Toc481523139)

[Measurable Features & Metrics 16](#_Toc481523140)

[Evaluation Results 18](#_Toc481523141)

[Development Time 18](#_Toc481523142)

[Cost 21](#_Toc481523143)

[Performance 23](#_Toc481523144)

[Splash Page Animation 23](#_Toc481523145)

[Results 23](#_Toc481523146)

[Raw-Data 23](#_Toc481523147)

[Graphical-Representation 25](#_Toc481523148)

[Drawer Open Animation 27](#_Toc481523149)

[Results 27](#_Toc481523150)

[Raw-Data 27](#_Toc481523151)

[Graphical Representation 30](#_Toc481523152)

[Drawer Close Animation 31](#_Toc481523153)

[Results 31](#_Toc481523154)

[Raw-Data 31](#_Toc481523155)

[Graphical Representation 33](#_Toc481523156)

[Network Request 35](#_Toc481523157)

[Results 35](#_Toc481523158)

[Raw-Data 35](#_Toc481523159)

[Graphical Representation 37](#_Toc481523160)

[List View - Infinite Scrolling FPS over 20 seconds 39](#_Toc481523161)

[Results 39](#_Toc481523162)

[Raw-Data 39](#_Toc481523163)

[Graphical Representation 41](#_Toc481523164)

[List View: Infinite Scrolling - Time to render new items 42](#_Toc481523165)

[Results 42](#_Toc481523166)

[Raw-Data 42](#_Toc481523167)

[Graphical Representation 45](#_Toc481523168)

[Details View: Forward Scene Change FPS 46](#_Toc481523169)

[Results 46](#_Toc481523170)

[Raw-Data 46](#_Toc481523171)

[Graphical Representation 48](#_Toc481523172)

[Details View: Backward Scene Change FPS 50](#_Toc481523173)

[Results 50](#_Toc481523174)

[Raw-Data 50](#_Toc481523175)

[Graphical Representation 53](#_Toc481523176)

[List view - Delete Item from top of list: Time taken to redraw list 54](#_Toc481523177)

[Results 54](#_Toc481523178)

[Raw-Data 54](#_Toc481523179)

[Graphical Representation 57](#_Toc481523180)

[Listview - Delete Item from bottom of list: Time taken to redraw list 58](#_Toc481523181)

[Results 58](#_Toc481523182)

[Raw-Data 58](#_Toc481523183)

[Graphical Representation 61](#_Toc481523184)

[Listview - Delete Item from middle of list: Time taken to redraw list 62](#_Toc481523185)

[Results 62](#_Toc481523186)

[Raw-Data 62](#_Toc481523187)

[Graphical Representation 65](#_Toc481523188)

[Discussion & Conclusions 66](#_Toc481523189)

[Where did React-Native perform better? 66](#_Toc481523190)

[Time 66](#_Toc481523191)

[Cost 66](#_Toc481523192)

[Performance 66](#_Toc481523193)

[Reflection 68](#_Toc481523194)

[Presentation Feedback 69](#_Toc481523195)

[References 71](#_Toc481523196)

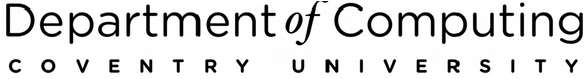
[Detailed Project Proposal 73](#_Toc481523197)

# 

# 300COM / 303COM Declaration of Originality

I Declare that This project is all my own work and has not been copied in part or in whole from any other source except where duly acknowledged. As such, all use of previously published work (from books, journals, magazines, internet etc.) has been acknowledged by citation within the main report to an item in the References or Bibliography lists. I also agree that an electronic copy of this project may be stored and used for the purposes of plagiarism prevention and detection.

## Statement of Copyright

I acknowledge that the copyright of this project report, and any product developed as part of the project, belong to Coventry University. Support, including funding, is available to commercialise products and services developed by staff and students. Any revenue that is generated is split with the inventor/s of the product or service. For further information please see www.coventry.ac.uk/ipr or contact ipr@coventry.ac.uk.

Please complete all fields.

First Name:

Robert

Last Name:

Rigler

Student ID number

4939377

Ethics Application

Number

**P51417**

1

st

Supervisor Name

Dr. Faiyaz Doctor

2

nd

Supervisor Name

Dr. Dianabasi Nkantah

## Statement of Ethical Engagement

I declare that a proposal for this project has been submitted to the

Coventry University ethics monitoring website

(https://ethics.coventry.ac.uk/) and that the application number is listed below (Note: Projects without an ethical application number will be rejected for marking)

Signed:  Date: 02/05/2017

This form must be completed, scanned and included with your project submission to Turnitin. Failure to append these declarations may result in your project being rejected for marking.

# Abstract

React-Native is a cross-platform mobile application development framework created by Facebook in 2015. It allows a developer to simultaneously create mobile applications for both the Android and iOS platforms in JavaScript instead of the platform native language (Java or Objective-C respectively). React-Native’s popularity stems from its intelligent combination of native and non-native code using “Bridges”. Bridges are a way of exposing native APIs to the JavaScript thread, allowing the utilization of the platforms’ high-performance native modules. This allows React-Native to achieve comparable performance to purely native applications whilst using the arguably simpler application design structure of React and JavaScript as well as having the benefit of creating applications for both platforms at once. React-Native operates very differently to other cross-platform technologies such as Cordova and Xamarin, which use an embedded web browser to render the application in HTML and CSS as if it were a normal web application. Whilst this method is considered cross-platform it sacrifices performance and freedom of design. Those who use React-Native claim to have seen reduced development time and cost, whilst sacrificing very little in terms of performance. These claims, however, are backed by very little evidence and almost no statistical data has been produced. Most of the available data comes from blogs that have a little objective evaluation of the development methods, or do not consider any special business requisites for the application, such as high-performance requirements. Businesses and individuals who are deciding which method to use when creating a new mobile application will find it difficult to form an accurate, unbiased decision using the data available, and may waste a large amount of time and money using a development method that is entirely unsuitable for their needs. The primary objective of this research is to fill the substantial gap in the data used to compare the two development methods. Using this data, we can accurately assess if using React-Native to develop mobile applications produces any substantial improvement in performance, development time, cost, and stability compared to using purely platform native development. These are arguably the most critical areas of concern for any business looking to develop a mobile application. The research involves creating two identical Android mobile applications and individually assessing each aspect of the application. The measurements will be taken for each feature of the application, so to produce a fine-grained analysis as well as considering the two methods on a broader scale.

This study found that, if Development-Time and Cost are the two principal factors in deciding to create a mobile-application, then React-Native is the superior choice. React-Native can potentially half the cost of developing mobile applications, whilst maintaining a similar development time. However, if the performance of the application is the first priority, then React-Native should be avoided, as React-Native sacrifices performance, a traditional native solution is far superior.

## Keywords

react-native; android; performance; development-time; mobile-application; comparison;

# Introduction

The aim of this project was to provide a detailed comparison and analysis into the development of mobile applications using the cross-platform development framework React-Native compared to only using traditional native development. This study focuses specifically in the areas most important for business use-cases: Performance, development time, cost, and stability. The results and data produced by this project can be used by both individuals and businesses alike when deciding how to develop their applications, whether the possible increased productivity of cross-platform development outweighs the possible loss in performance.

To fully substantiate the need for this study, I will first explore why businesses create mobile applications, why some applications are perceived to fail, how cross-platform development may help or hinder applications, and how React-Native aims to solve the problems application development.

The advantages of having a mobile application to extend a platform are apparent. In 2015 the number of people with smartphones was 1.86 Billion (Statista.com, 2016). This is an incredibly large potential user base that businesses and individuals can take advantage of to increase recognition and profits.

In march of 2016, Domino’s Pizza reported that 48.6% of its online sales were done via its mobile application. (BBC,

However, whilst mobile-applications can be of great benefit to a business there is also great risk. In 2014, Swrve, a leading marketing automation platform, released their App-Metrics-Report. In this report they found that thirty-four percent (34%) of mobile app engagements last less than one minute, and the average session length was five minutes and seven seconds. In the study “Why Mobile Apps Fail” (PerfectoMobile, 2014), it’s stated that the top four app issues that users found were: UI, performance, monitoring, and device compatibility issues.

Device compatibility issues refers to the differences between the two most popular platforms: Apple’s iOS and Google’s Android. Both of these platforms offer different hardware, User-interface guidelines. Ignoring platform specific features such as Android's’ hardware Back button can lead to poor app retention. Performance based problems refers to: Long loading times, low FPS and generally slow performance. However the leading cause of poor app retention is because of instability, mostly in the form of software bugs that hinder the user's ability to use the application.

Traditionally mobile-applications are developed separately for each platform, often by separate teams who are experienced in the technology supported by the platform. Having to hire two separate teams, could potentially mean that the cost of developing an application can double.

# Literature Review

My research focuses on the statistical benefits of creating mobile applications in React-Native, as opposed to using purely native development. The aim of this research is to be able to compare both types of development and assess each of them in key areas of mobile development. These areas include: Performance, Development Time, Cost, and Stability. React-Native was created by Facebook in 2015, and the framework is currently still in an ‘Unstable’ state as an official release version is yet to be produced (The current version is 0.42). Despite its relative youth compared to native mobile application development, React-Native has exceeded google search trends for both “android development”, and “iOS development”.

### Performance

Generally, most authors agree that there is a noticeable performance deficit when using React-Native. However, there is some ambiguity over where this deficit occurs. According to Vytenis Narusis, a Junior mobile developer at Devbridge, the greatest performance impact comes from the JavaScript Language itself: “It’s natural that JavaScript code is not as efficient for calculation-intensive tasks, and there is an overhead when JavaScript is controlling native elements.” (Narusis), He also states that “In performance, React Native stays behind an optimised native application.”. No statistical data has been given to support this argument, one could surmise that a low-level compiled language such as Java, will perform better than a JIT (Just in time) compiled language such as JavaScript, however this is purely conjecture as it is impossible to know the optimisations made in either language, and often performance will have a greater dependence on how a problem is implemented, rather than in what language it is implemented.

Others argue that the performance deficit stems from the Single Threaded nature of JavaScript. React-Native runs all JavaScript code on a single separate thread, as JavaScript cannot be declaratively parallelised. React-Native utilises Native API’s and components by passing data from the JavaScript thread to the Native thread, this is known as a bridge. Many authors indicate that these bridges are the source of React-Natives performance bottle-necks. In the blog article Performance Limitations Of React Native And How To Overcome Them, by Tal Kol, a Start-up founder and Senior Developer at Wix.com, he states that: “The performance bottleneck often occurs when we move from one realm to the other. In order to architect performant React Native apps, we must keep passes over the bridge to a minimum.” (Kol, 2016). He refers to the Native threads and JavaScript thread as separate Realms, he advocates to minimise the use of Bridges. This is a better indicator of where the performance problem lies, the use of bridges in React-Native is unavoidable, and thus careful optimisation is imperative to both the framework itself, and the developer using React-Native. The bridging performance bottleneck seems to be especially common when attempting to create animations solely using JavaScript. Sriraman Paneer (a mobile application developer a Witworks), states in his article “What we learned after using React Native for a year” that “Animation APIs run all the processes on the JS Thread. Consequently, we had faced some performance issues while using these APIs.” (Panneer). A similar claim is made by Robin Chen, a senior developer at Discord, who chose to migrate their VOIP application from Native to React-Native. “The Animated library also cannot deliver the animations as smooth as the native while doing heavy duty works on the JS thread.” (Chen).

After reading many articles, and using the sources that I have given here, I am left with more questions about React-Native performance that when I began. It seems clear that there is a definitive decrease in performance when using React-Native, however, it is not clear about how large or small that sacrifice is. E.g. How much of a difference In FPS is there between Animations in Native and React-Native? How much longer would an intensive calculation take using JavaScript than Java? Does the problem lie in the developers implementation rather than the framework? My research aims to fill these gaps in the knowledge, as the ambiguous information available is simply not enough to make an informed decision about Whether to use React-Native of not.

### Development-Time

A crucial factor of creating any application is development-time, especially for prototyping new features, or creating an MVP (Minimum viable product). How quickly an application can be developed may be quintessential to a business model, or may be entirely irrelevant. In cases, development-time has a direct correlation to the cost of developing that application.

Being a Hybrid application framework (i.e. Can develop applications for both iOS and Android simultaneously, using the same code base), one would expect that the development time for a React-Native to be half that of developing a native application for both platforms. However, my research will only be focusing on the Android platform, so whilst the cross-platform development is of great importance, I must primarily focus on single platform development.

Greg Gomez, a Project Manager at 3-Sided-Cube said that: “The initial benefit that strikes me is the ability to prototype with even more rapidity than previously. React-Native will offer developers unparalleled speed in developing high fidelity prototypes, so ideas can be validated, assumptions debunked and implementations battle-tested in record time.” (Gomez, 2017). This is certainly high praise for React-Native, however he has not stated why React-Native had a faster development-time. He later advocates the re-usability aspect of React-Native, “Developing a component once allows us to reuse it easily in other projects, meaning that over time you’ll amass a nice library of drop-in components to build prototypes from. It means that time spent re-inventing the wheel, can be spent solving your client’s problems and delivering value.”. This is a far more important than it seems, React, and therefore React-Native, has a component based architecture, which advocates creating components of the application independently. This allows for a minimal amount of dependency between different parts of the application, meaning that a developer could develop generic components that can be used in anywhere in any application, or perhaps use open-source components that others have developed. Whilst this is still possible in Native development, it is not possible to the extent that React-Native allows composition of components.

In Native development, every time that new code is implemented, changed or updated, the entire application needs to be re-compiled, if the application is large this could take quite some time, and quickly becomes aggravating, especially when the code changes are minimal. React-Native being able to take advantage of both Native and non-native JavaScript, has an impressive feature called Hot-Reloading. Hot-Reloading, allows any changes to the JavaScript code to be almost instantaneously reflected in the application; “React Native’s feedback loop is bewitchingly low. It takes less than one or two seconds between you saving a file and seeing the change in your app. That’s easily ten times less than the typical Build and Run cycle we’re used to in (Native development).” (Elkin, 2016). This does not apply to Native code, as this will need to be recompiled. This feature alone could make an enormous impact on the speed of the application development.

Naturally, not all developers will be able to immediately become productive with React-Native, traditional Native developers with little experience with JavaScript or React may struggle. “As native developers we found the learning curve of React Native pretty high, as it may take up to a month until one could start feeling comfortable developing with this framework.” ("React Native: Is It The End Of Native Development?", 2016). This may entirely negate the benefits of rapid prototyping and Hot-Reloading, having to potentially delay development for a month or more has the potential to incur serious costs. This however, is not true for all situations, Developers that are familiar with JavaScript, and more importantly familiar with the React framework will take to React-Native far quicker that purely Native developers.

Essentially, my research will focus on single platform development, specifically the Android platform. I aim to provide an analysis into the time taken to develop different parts of a mobile-application. Depending on the results I will offer reasons as to why there was or was not a difference in development time. “Assuming equal skill level on the part of the developer, it shouldn’t take longer to build an app on one platform or the other,” (Chernov, 2013), said Joe Chernov, VP of marketing on behalf of Kinvey’s engineering team. Perhaps React-Native has some particularly obtuse API’s that hinder development time for particular areas of development. Perhaps the only reason React-Native has faster development time is because of Hot-Reloading. These are the types of questions that are missing from the current information available.

### Cost

The cost of developing mobile applications is very high, many businesses will be looking to minimise the cost of creating an application whilst still sacrificing as little as possible in other areas. Whilst I have separated cost into a separate category, it is strongly tied to the amount of time it takes to develop an application, and so for the sake of clarity, some points from the previous section may be repeated.

Bernard Kohan of Comentum created an estimated ball-park cost of development for creating an MVP application (Minimum Viable Product) for both native and hybrid applications. He estimated that the cost of creating a native MVP mobile application for just one platform to be approximately one-hundred and fifty-two thousand dollars ($152,000), and creating the same native application for both iOS and Android platforms would cost two-hundred and fifty-one thousand dollars ($251,000). He also estimated the development cost for a hybrid application, i.e. An application that can be simultaneously developed on both the iOS platform and Android platform, to be one-hundred and sixty-two thousand dollars ($162,500). Either way, this is a substantial investment, Kohan goes on to state that: “If the developer needs to build the app for all platforms, and the app needs to be built using the native programming language of each platform, the cost of development will be doubled or tripled because of the amount of time that is needed to build the app using different programming languages of each platform.” (Kohan). The premise that hybrid development is almost half the price of native development is backed up by Keerti, a developer at Walmart Labs, who created the Walmart application in React-Native, he states that; “The defining feature of React Native, and arguably its best selling point, is that it’s cross platform — allowing for simultaneous development on iOS and Android by the same team, which can cut labor costs roughly in half.” (Keerti). With the price of Native application development being so high, it is natural that Hybrid development would seem the more attractive choice. Others, such as [NAME HERE] would dispute this claim, stating that: “As native developers we found the learning curve of React Native pretty high, as it may take up to a month until one could start feeling comfortable developing with this framework.” ("React Native: Is It The End Of Native Development?", 2016). Whilst this is not a direct opposition to the previous argument, I do feel that this is a relevant piece of information to consider. If it takes up to a month and possibly more for a native developer to familiarize themselves with the framework then this may effectively negate any improvements in development time, and thus an implicit reduction in cost. However this is a fairly subjective claim, even though the React-Native framework is relatively new, there are those who have been using the Web framework React for quite some time, and they would take far quicker to acclimatize themselves to using React-Native.

Ultimately, cost is a category that I will only be able to estimate by comparing the amount of time It will take for me to complete the development of both applications and average developer salaries, so it is more than likely that my research will not yield any results that are different from the sources above. However, I may be able to ascertain which parts of the application development would cost the most, and produce a relationship between cost, development-time and performance, which I believe would be a valuable new piece of information.

### Stability

The stability of an application has the ability to significantly impact the cost and development-time of an application, so it must be closely monitored. In the article “Why I’m not a React-Native developer” by Ariel Elkin, she states that: “JavaScript lacks these safeguards against programmer error, making preventable runtime crashes and preventable programmer errors part of your routine.” (Elkin, 2016). Implying that the use JavaScript is a disadvantage and will adversely affect the stability of an application. Mostly this is a reference to the absence of type checking in JavaScript, and the ability to send data over a bridge from JavaScript to Native code that has not been type-checked, which will in most cases cause a fatal runtime error. These “Preventable programmer errors” that Elkin suggests may however be negated by the decreased development time in React-Native. Earlier in the article Elkin also stated that: “React Native’s feedback loop is bewitchingly low. It takes less than one or two seconds between you saving a file and seeing the change in your app. That’s easily ten times less than the typical Build and Run cycle we’re used to in Xcode.” (Elkin, 2016). Do these two points contradict each other? Is an error produced when compiling an application different to when an error is found when testing an application? Whilst the compiled application may catch the error quicker (Only if the time it takes to compile the application is low), it may only be relevant if those bugs are deployed into the production application. it seems that React-Native may be more vulnerable to developer error than native applications, and thus React-Native has a greater dependence on Unit-Testing and User-testing than Native development.

React-Native is a relatively new Framework, made public in 2015, but it still does not have an official release as the current version is 0.42 and it may not get a stable 1.0 release for some time. However, being an open source framework, when bugs are found, they are rapidly fixed and integrated. “While it is good that they bring more features and push the framework towards maturity, often they bring breaking changes as well” ("React Native: Is It The End Of Native Development?", 2016). Whilst new features and bugs are rapidly integrated into the framework, there’s also the possibility of producing breaking changes, i.e. Major changes in the API that will cause previous code to break. This may be viewed as a positive light, when assuming that breaking changes are only introduced when they are a vast improvement, however, depending on an applications reliance on these features, an equally large amount of time may have to spent on refactoring and redesigning code to be able to take advantage of these latest features, which in-turn would increase the costs of supporting the application. This quote from the same article validates this assumption: “These changes sometimes where a burden to overcome, as we had people spending a lot of time fixing things when upgrading the framework.” ("React Native: Is It The End Of Native Development?", 2016).

You may have noticed that all my sources have come from either, independent blogs, or from a company blog. This is because there are no existing scientific reviews, or articles covering React-Native, there were however a plethora of blogs and web-based articles. Many of these I had to exclude as sources entirely because of their sensationalist or inaccurate nature. I also chose to exclude any sources that were over one year old, this is because of the quick release schedule of react-native, the version of React-Native that existed a year ago, is entirely different from the one available today, and thus most of the information in the sources is similarly out-dated.

In summary, there is a large amount of informal data surrounding the advantages and disadvantages of using React-Native, apart from cost however, there is no statistical data available to create an accurate depiction of the differences between Native and non-Native development. My research aims to produce this statistical data.

# 

# Method

## Method Description

I will develop two mobile-applications, one using only Native Java using Android Studio as an Integrated Development Environment. The other application will be created using Facebook’s React-Native, a cross-platform development framework, using the VSCode editor.  
  
The application will be separated into small features, and for each feature I will record the amount of time taken to implement each feature. Once both applications are completed I will produce performance metrics for each feature, thus producing a set of data for each application which can be directly compared to each other.

The following areas will be compared between the two applications:

1. Performance
2. Development-time
3. Cost (Derived)

### Performance

For most features, a performance metric will be directly measurable. E.g. For measuring the performance of animations, the metric would be average FPS (Frames per Second) over the duration of the animation.

For large list manipulation I.e. (Removing an item from the middle of a large list), the metric would be TTD (Time to Draw) which is the time it took to remove the element from the list and redraw the screen with the new data.

### Development-Time

For the development-time metric, I will record the time it took to complete a feature. A feature is "Completed" when it passes the set of unit tests that have been created for that feature (Both applications will use the same unit tests, however, they will be implemented using the platform respective unit-testing framework).

Being very familiar with both Java and React-Native, I will be able to efficiently implement both applications.

## Variables and Specifications

All the tests were conducted on a OnePlus 3 device, with a 2.0 GHz Processor and 8 GBs or RAM. The device was restored to factory condition and is running and version 7.1 Nougat.

The applications were running in an optimised released mode.

## The application

The application represents a typical mobile-application. Specifically, this application acts as a front-end to a movie API, allowing the user to search for movie, and see details about specific movies, as well as being able to get recommendations for similar movies.

This application covers a range of the most used mobile-application features, and I will explore each feature in the section below.

All code can be found under the following public repository: <https://github.com/Riglerr/330COM>

## Feature List

|  |  |  |
| --- | --- | --- |
| **Domain** | **Feature** | **Description** |
| Splash Page |  | The splash page is shown whilst the app is initializing. Generally, a splash is used to load any data that is needed before allowing the user to use the application. |
| Splash Page | Page Layout: centered Text, white background |  |
| Splash Page | Title and subTitle text |  |
| Splash Page | Loading Animation |  |
| Drawer View |  | A Drawer is used to navigate several predetermined pages in an application. It is hidden until the user presses the drawer Icon on the toolbar, or swipes in from the right of the screen. Upon pressing the name of the target view, the drawer closes, toolbar title changes, and the desired page is shown in the content view. |
| Drawer View | Toolbar Layout |  |
| Drawer View | Toolbar Icon |  |
| Drawer View | Toolbar Icon open/close action |  |
| Drawer View | Toolbar Dynamic Title |  |
| Drawer View | Navigation View Layout |  |
| Drawer View | Navigation View Header |  |
| Drawer View | Navigation View List |  |
| Drawer View | Navigation View Active Item |  |
| Drawer View | Navigation View List item onPress Action |  |
| Search View | Page layout |  |
| Search View | Text Input |  |
| Search View | Text input Search Icon |  |
| Search View | List View |  |
| Search View | List View Row Layout |  |
| Search View | List View Data binding |  |
| Search View | List View Infinite Scrolling |  |
| Search View | List View Row Item Press |  |
| Details View | Layout |  |
| Details View | Android Back Button Interaction |  |
| Recommend View | Text Input |  |
| Recommend View | Search Icon |  |
| Recommend View | Row Layout |  |
| Recommend VIew | List View Data Binding |  |
| Recommend View | List View Row Layout |  |
| Recommend View | List VIew Infinite Scrolling |  |
| Recommend View | Item Swipe Interaction |  |
| Recommend View | Row Item Press |  |
| Recommend View | Individual List item removal |  |
| API | Search API function |  |
| API | Recommend API function |  |

## Measurable Features & Metrics

|  |  |  |
| --- | --- | --- |
| **Domain** | **Feature** | **Metric** |
| Splash Page | Animation | Average FPS over 100 seconds |
| Drawer View | Animation Open | Average FPS over 100 events |
| Drawer View | Animation Close | Average FPS over 100 events |
| Search View | Movie Search (Using API) | Average Time to Render Data over 100 events |
| Search View | List Scrolling | Fps over 2 second scroll |
| Search View | Infinite Scrolling FPS | FPS over 10 seconds of scrolling |
| Search View | Infinite Scrolling: Time to draw new Items | Average time over 100 events |
| Details View | Forward Scene Change FPS | Average FPS over 50 events |
| Details View | Backward Scene Change FPS | Average FPS over 50 events |
| Recommend View | Delete Item From Top: Time to redraw | Time taken in ms to redraw the list |
| Recommend View | Delete item from bottom: time to redraw | Time taken in ms to redraw the list |
| Recommend View | Delete item from the Middle: time to redraw | Time taken in ms to redraw the list |

# Evaluation Results

All code can be found under the following public repository: <https://github.com/Riglerr/330COM>

## Development Time

Development-time is a critical metric when deciding to create a mobile application.

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature Domain** | **Feature Name** | **Native Development Time (mins)** | **React-Native Development Time (mins)** |
| Splash View | Page Layout | 20 | 10 |
| Splash View | Title & Subtitle: Position, color, size | 10 | 10 |
| Splash View | Loading Animation | 180 | 130 |
| Drawer View | Toolbar: Dynamic title | 30 | 30 |
| Drawer View | Toolbar: Drawer Icon | 5 | 20 |
| Drawer View | Toolbar: Layout & child positioning | 10 | 15 |
| Drawer View | Toolbar: Icon open/close action | 5 | 30 |
| Drawer View | Navigation View Layout | 60 | 5 |
| Drawer View | Navigation View: Header | 20 | 10 |
| Drawer View | Navigation View List | 15 | 10 |
| Drawer View | Navigation View: Active item state & styles | 5 | 15 |
| Drawer View | Navigation View: List Item | 25 | 15 |
| Drawer View | Navigation View: List Item on Press Action | 20 | 20 |
| Drawer View | Open Drawer Action | 5 | 5 |
| Drawer View | Close Drawer Action | 5 | 10 |
| Search View | Page layout | 10 | 5 |
| Search View | Text Input | 5 | 10 |
| Search View | Text input Search Icon | 5 | 15 |
| Search View | List View | 20 | 15 |
| Search View | List View Row Layout | 15 | 10 |
| Search View | List View Data binding | 30 | 10 |
| Search View | List View Infinite Scrolling | 50 | 60 |
| Search View | List View Row Item Press | 15 | 25 |
| Details View | Layout | 20 | 10 |
| Details View | Android Back Button Interaction | 10 | 25 |
| Recommend View | Text Input | 5 | 5 |
| Recommend View | Search Icon | 5 | 5 |
| Recommend VIew | List View Data Binding | 25 | 20 |
| Recommend View | List View Row Layout | 15 | 5 |
| Recommend View | List View Infinite Scrolling | 30 | 60 |
| Recommend View | Item Swipe Interaction | 80 | 50 |
| Recommend View | Row Item Press | 35 | 15 |
| Recommend View | Individual List item removal | 60 | 30 |
| API | Search API function | 50 | 45 |
| API | Recommend API function | 45 | 30 |

|  |  |  |
| --- | --- | --- |
|  | Total in mins | Total in Hours |
| Native | 945 | 15.75 |
| React-Native | 785 | 13.08 |

This data shows that the React-Native solution was 16.9% faster, taking 13.08 Hours to develop, as opposed to the Native solutions 15.75 Hours.

Longer, more complex features like the Splash Screen animation took approximately 3 hours on the native solution, whilst it took only two for the React-Native solution. This was because of the relative simplicity of React-Natives’ animation APIs.

Generally, layout features where 20% faster on the React-Native. In Native android development using Android Studio, the UI is designed via a Drag-and-Drop interface, whereas React-Native layout is done via text which highly resembles HTML and CSS. React-Native’s method, was far more direct and less error prone than the native GUI, which often produced unexpected results.

Images were a problem for react-native, as often it would take a large amount of time to resolve images that were in the android resources folder. This was trivialized in the Native GUI as one could simply “Browse Images” and select the desired image, rather than having to construct a complex relative destination string.  
Similarly, it took longer to scale images in React-Native, as it had to be done via trial and error by changing the value and waiting for the view to refresh. In the Native GUI, I could simply change the size of the image in the preview window.  
  
React-Natives *Hot Reloading* proved to be very useful throughout the development process. *Hot-Reloading* is a feature of React-Native’s development, which allows code changes to be instantly reflected on the targeted device or simulator. Whereas in Native development, any changes to code or design require the whole application to be re-compiled.   
Hot-Reloading was especially helpful when implementing the features which involved more complex code. Such as the two API methods, which were around 10% faster than their native counterparts, as well as binding data to the search and recommend list views.

In the Native solution, data binding requires me to create separate data adapter classes, which control how the list behaves with the data source. React-Native was as simple as passing an Array as a parameter to the List View component.

Because React-Native is a cross-platform framework, the application I have developed also works on iOS devices, such as iPhones and iPads, the Native solution only works on Android devices. So, React-Native was 16.9% quicker developed than the Native solution, and technically twice the work was achieved.

# Cost

It’s important to note that I developed both applications myself and did not receive payment or sponsorship from any individual or organisation. The statics provided here are theoretical, that are derived from development time and Mean average salary.

According to ITJobsWatch, a company specialised in tracking the UK IT Job Market and Salary Benchmarking

Android Developer 2017 Median Daily Rate: £435 (Sample Size of 193 Job quotes)

React-Native Development 2017 Median Daily Rate: £475 (Sample size of 49 Job quotes)

iOS Software Developer 2017 Median Daily Rate: £400 (Sample Size of 200 Jobs)

Here we can see that the React-Native developers are on average, paid 9.43% more per day than Native Android Developers, and 15.79% more than iOS developers. Although the React-Native statistic is gathered from a far lower sample size, which may influence how accurate this figure is.

In the UK, the average daily hours worked is 8 hours. From this we can theorize that the Mean Hourly rate for each type of developer is:

|  |  |
| --- | --- |
|  | Median Hourly Rate |
| Ios | £50 |
| Android | £54.37 |
| React-Native | £59.37 |

Using this data, and the time taken to create both of my applications, I can surmise how much It would have cost to create each application.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Hourly Rate | Time to develop (in Hours) | Total Predicted cost |
| Android | £54.37 | 15.75 | £856.32 |
| React-Native | £59.37 | 13.08 | £776.56 |

From this data we can see that, even though a React-Native developer has a higher hourly rate, it would still cost less to develop this application using React-Native.

Even when we round up the time taken to develop to the nearest hour (16 and 14 respectively) the React-Native solution is still cheaper.

React-Native is a cross-platform framework, so in the 13.08 hours it took to develop the application I have created two applications, one for Android and on for iOS. This means that potentially, by using React-Native, one could develop two mobile applications for a comparable price of developing one application.

If we consider the hourly Rate for iOS developers, and assume that it takes a similar time to develop an iOS application compared to an Android application, we can theorize that the cost of developing two separate mobile applications (one for each platform) would cost £1567.42. Which is more than double the cost of the cross-platform solution.

## Performance

### Splash Page Animation

**Description:** This test measures the Average FPS of the splash page animation over a sixty second period (60 seconds). The FPS was polled every second over the duration of each test, therefore each result is the average of sixty measurements.

**Key Metric**: Average FPS over 60 seconds

**Test Iterations**: 50

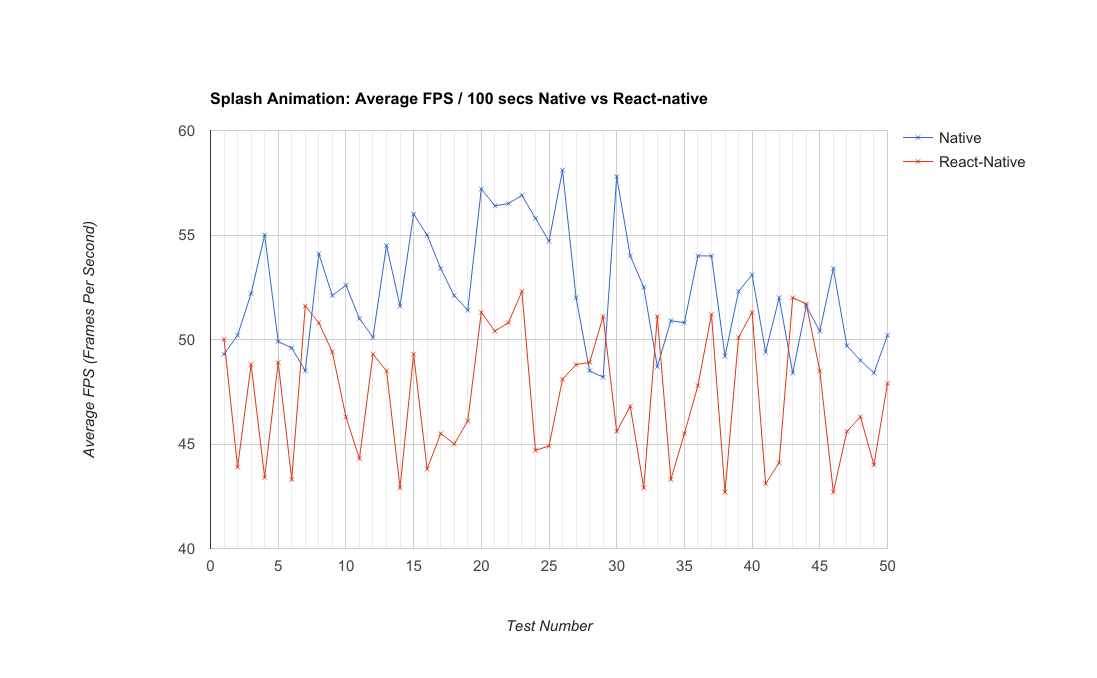
#### Results

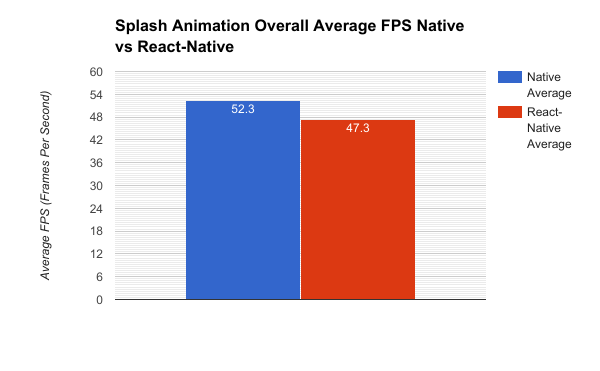
##### Raw-Data

|  |  |  |
| --- | --- | --- |
| **Test No.** | **Average FPS / 60 secs Native** | **Average FPS / 60 secs React-Native** |
| 1 | 49.3 | 50 |
| 2 | 50.2 | 43.9 |
| 3 | 52.2 | 48.8 |
| 4 | 55 | 43.4 |
| 5 | 49.9 | 48.9 |
| 6 | 49.6 | 43.3 |
| 7 | 48.5 | 51.6 |
| 8 | 54.1 | 50.8 |
| 9 | 52.1 | 49.4 |
| 10 | 52.6 | 46.3 |
| 11 | 51 | 44.3 |
| 12 | 50.1 | 49.3 |
| 13 | 54.5 | 48.5 |
| 14 | 51.6 | 42.9 |
| 15 | 56 | 49.3 |
| 16 | 55 | 43.8 |
| 17 | 53.4 | 45.5 |
| 18 | 52.1 | 45 |
| 19 | 51.4 | 46.1 |
| 20 | 57.2 | 51.3 |
| 21 | 56.4 | 50.4 |
| 22 | 56.5 | 50.8 |
| 23 | 56.9 | 52.3 |
| 24 | 55.8 | 44.7 |
| 25 | 54.7 | 44.9 |
| 26 | 58.1 | 48.1 |
| 27 | 52 | 48.8 |
| 28 | 48.5 | 48.9 |
| 29 | 48.2 | 51.1 |
| 30 | 57.8 | 45.6 |
| 31 | 54 | 46.8 |
| 32 | 52.5 | 42.9 |
| 33 | 48.7 | 51.1 |
| 34 | 50.9 | 43.3 |
| 35 | 50.8 | 45.5 |
| 36 | 54 | 47.8 |
| 37 | 54 | 51.2 |
| 38 | 49.2 | 42.7 |
| 39 | 52.3 | 50.1 |
| 40 | 53.1 | 51.3 |
| 41 | 49.4 | 43.1 |
| 42 | 52 | 44.1 |
| 43 | 48.4 | 52 |
| 44 | 51.6 | 51.7 |
| 45 | 50.4 | 48.5 |
| 46 | 53.4 | 42.7 |
| 47 | 49.7 | 45.6 |
| 48 | 49 | 46.3 |
| 49 | 48.4 | 44 |
| 50 | 50.2 | 47.9 |

|  |  |  |
| --- | --- | --- |
|  | Native | React-Native |
| Average | 52.3 | 47.3 |
| Min | 48.2 | 42.7 |
| Max | 58.1 | 52.3 |
| Range | 9.9 | 9.6 |

###### Graphical-Representation





Here we can see that React-Native performed on average, 10% worse than the purely native solution. React-Native’s highest recorded average was identical to the Native solutions average score. Only four of the recorded results for React-Native achieved a higher FPS than its Native counterpart.   
This data is a clear indication that React-Native’s animation functionality is not as efficient as purely native animation. The most probable cause of this would be React-Native’s bridges. Al large amount of position related data is transferred between React-Native and the Native modules, over a Bridge. This bottle-neck does not exist in a purely native solution, which is evident by its much higher recorded FPS values.

### Drawer Open Animation

**Description**: This test measures the average FPS for during the Drawer opening animation. This animation consists of a list sliding in from the left-hand side to cover 70% screen, whilst simultaneously dimming the background view. This animation lasts approximately half a second (0.5 seconds).

**Key Metrics**: Average FPS for event duration.

**Iterations**: 50

#### Results

##### Raw-Data

|  |  |  |
| --- | --- | --- |
| Test No. | Average Native FPS | Average React-Native FPS |
| 1 | 51.3 | 51.7 |
| 2 | 53.9 | 53.5 |
| 3 | 58.2 | 55.9 |
| 4 | 56.8 | 56.6 |
| 5 | 54 | 55.8 |
| 6 | 57.4 | 51.9 |
| 7 | 52 | 55 |
| 8 | 51.5 | 53.1 |
| 9 | 54.6 | 57 |
| 10 | 51.6 | 48.6 |
| 11 | 58.2 | 53.1 |
| 12 | 57.3 | 48.7 |
| 13 | 57.3 | 57.2 |
| 14 | 56.6 | 52.5 |
| 15 | 52.4 | 51.9 |
| 16 | 54 | 52.5 |
| 17 | 55.3 | 50.5 |
| 18 | 52.1 | 53.2 |
| 19 | 52.5 | 56.7 |
| 20 | 52.3 | 50.7 |
| 21 | 57.8 | 56.7 |
| 22 | 52.1 | 54.5 |
| 23 | 53.9 | 56 |
| 24 | 52.9 | 51.9 |
| 25 | 58.9 | 54.6 |
| 26 | 53.5 | 57.2 |
| 27 | 59.8 | 51 |
| 28 | 57.9 | 51.9 |
| 29 | 57 | 51.4 |
| 30 | 55.2 | 53.2 |
| 31 | 50.2 | 55 |
| 32 | 57.7 | 53.2 |
| 33 | 53.7 | 48.7 |
| 34 | 52.1 | 49 |
| 35 | 51.3 | 48.5 |
| 36 | 54.6 | 54.3 |
| 37 | 53.7 | 49.1 |
| 38 | 53.7 | 51 |
| 39 | 58.1 | 56.3 |
| 40 | 56.5 | 55.8 |
| 41 | 58.2 | 53.4 |
| 42 | 59.6 | 54.2 |
| 43 | 57.8 | 51.8 |
| 44 | 57 | 54.9 |
| 45 | 53.3 | 49.3 |
| 46 | 58.8 | 50.6 |
| 47 | 54.1 | 53.2 |
| 48 | 56.5 | 50.6 |
| 49 | 57.7 | 56.6 |
| 50 | 58.4 | 56.5 |

**Overall Average FPS (Frames Per Second)**

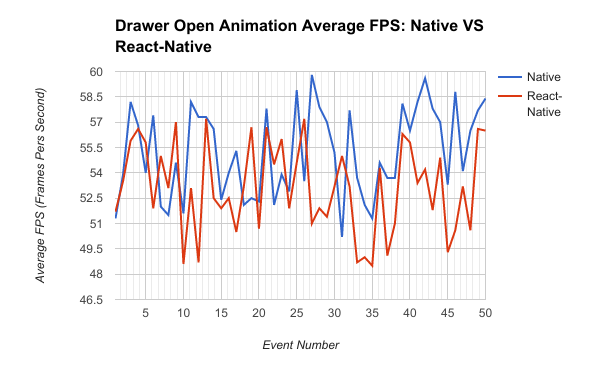
Native: 55.2 FPS

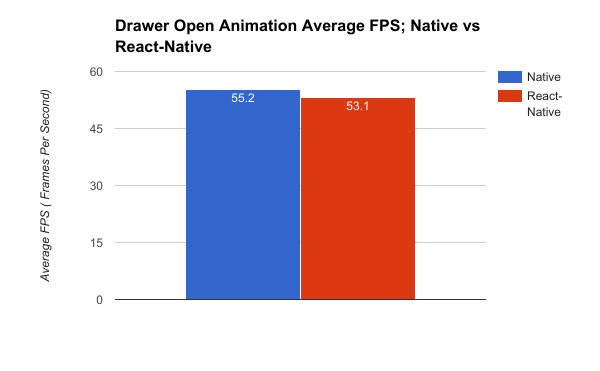
React-Native: 53.1

In this test, there is a much smaller difference between the React-Native and the Native solution. This may be because this animation is much less intensive than the splash animation, thus React-Native is more able to provide higher FPS values.

##### 

##### Graphical Representation





### Drawer Close Animation

**Description**: This test measures the average FPS for during the Drawer closing animation. This animation consists of the previously opened drawer sliding off the screen to the left, whilst simultaneously restoring the background view to its original brightness. This animation lasts approximately half a second (0.5 seconds).

**Key Metrics**: Average FPS for event duration.

**Iterations**: 50

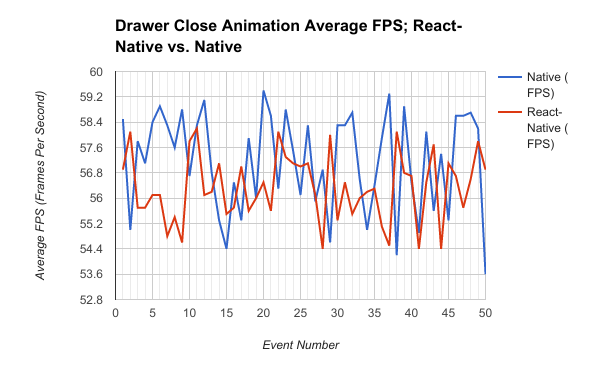
#### Results

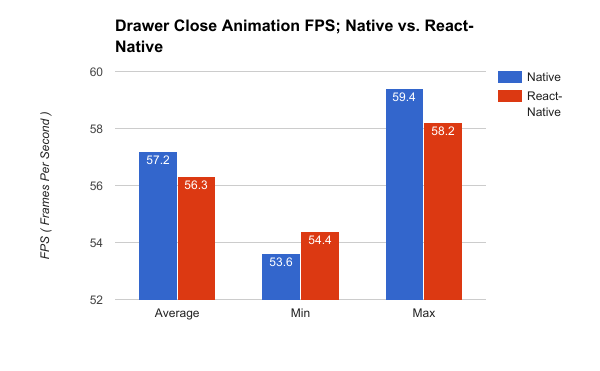
##### Raw-Data

|  |  |  |
| --- | --- | --- |
| Test No. | Native (FPS) | React-Native (FPS) |
| 1 | 58.5 | 56.9 |
| 2 | 55 | 58.1 |
| 3 | 57.8 | 55.7 |
| 4 | 57.1 | 55.7 |
| 5 | 58.4 | 56.1 |
| 6 | 58.9 | 56.1 |
| 7 | 58.3 | 54.8 |
| 8 | 57.6 | 55.4 |
| 9 | 58.8 | 54.6 |
| 10 | 56.7 | 57.8 |
| 11 | 58.3 | 58.2 |
| 12 | 59.1 | 56.1 |
| 13 | 56.8 | 56.2 |
| 14 | 55.3 | 57.1 |
| 15 | 54.4 | 55.5 |
| 16 | 56.5 | 55.7 |
| 17 | 55.3 | 57 |
| 18 | 57.9 | 55.6 |
| 19 | 56 | 56 |
| 20 | 59.4 | 56.5 |
| 21 | 58.6 | 55.6 |
| 22 | 56.3 | 58.1 |
| 23 | 58.8 | 57.3 |
| 24 | 57.5 | 57.1 |
| 25 | 56.1 | 57 |
| 26 | 58.3 | 57.1 |
| 27 | 55.9 | 56.1 |
| 28 | 56.9 | 54.4 |
| 29 | 54.6 | 58 |
| 30 | 58.3 | 55.3 |
| 31 | 58.3 | 56.5 |
| 32 | 58.7 | 55.5 |
| 33 | 56.6 | 56 |
| 34 | 55 | 56.2 |
| 35 | 56.4 | 56.3 |
| 36 | 57.9 | 55.1 |
| 37 | 59.3 | 54.5 |
| 38 | 54.2 | 58.1 |
| 39 | 58.9 | 56.8 |
| 40 | 56.5 | 56.7 |
| 41 | 54.9 | 54.4 |
| 42 | 58.1 | 56.5 |
| 43 | 55.6 | 57.7 |
| 44 | 57.4 | 54.4 |
| 45 | 55.3 | 57.1 |
| 46 | 58.6 | 56.7 |
| 47 | 58.6 | 55.7 |
| 48 | 58.7 | 56.6 |
| 49 | 58.2 | 57.8 |
| 50 | 53.6 | 56.9 |

|  |  |  |
| --- | --- | --- |
|  | Native | React-Native |
| Average | 57.2 | 56.3 |
| Min | 53.6 | 54.4 |
| Max | 59.4 | 58.2 |

##### Graphical Representation





This test yielded almost identical results to its predecessor, further supporting the claim that React-Native animations are of similar performance to Native animations when the device or application is not under stress.

### Network Request

**Description**: This measurement is the time taken to complete An API Request.

To make sure that this test was as balanced as possible, I sent the request to the same endpoint: <https://api.themoviedb.org/3/search/movie?query=star> . This endpoint always returns the same response, which is 15.5 KB in size.

**Key Metric:** MS (Milliseconds). Time taken to receive a response from the API server. Lower Is Better

**Iterations:** 50

#### Results

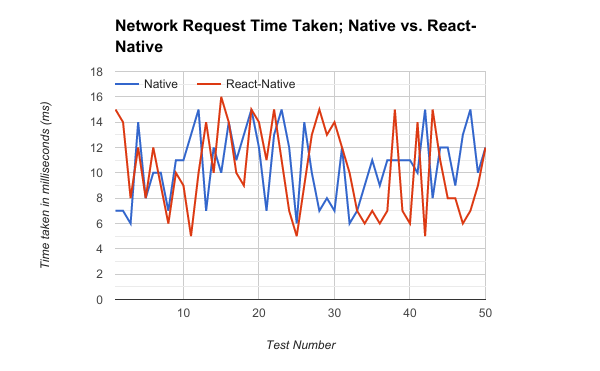
##### Raw-Data

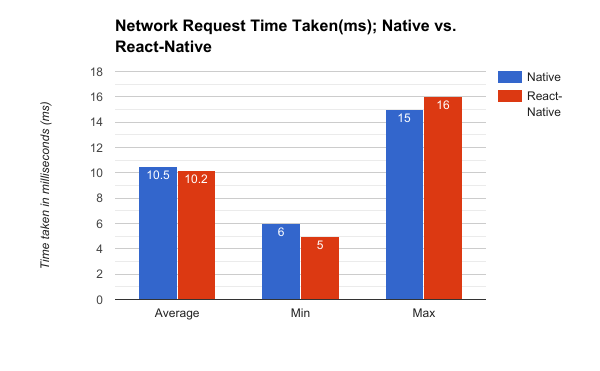
|  |  |  |
| --- | --- | --- |
| Test No. | Native (ms) | React-Native (ms) |
| 1 | 7 | 15 |
| 2 | 7 | 14 |
| 3 | 6 | 8 |
| 4 | 14 | 12 |
| 5 | 8 | 8 |
| 6 | 10 | 12 |
| 7 | 10 | 9 |
| 8 | 7 | 6 |
| 9 | 11 | 10 |
| 10 | 11 | 9 |
| 11 | 13 | 5 |
| 12 | 15 | 10 |
| 13 | 7 | 14 |
| 14 | 12 | 10 |
| 15 | 10 | 16 |
| 16 | 14 | 14 |
| 17 | 11 | 10 |
| 18 | 13 | 9 |
| 19 | 15 | 15 |
| 20 | 12 | 14 |
| 21 | 7 | 11 |
| 22 | 13 | 15 |
| 23 | 15 | 11 |
| 24 | 12 | 7 |
| 25 | 6 | 5 |
| 26 | 14 | 9 |
| 27 | 10 | 13 |
| 28 | 7 | 15 |
| 29 | 8 | 13 |
| 30 | 7 | 14 |
| 31 | 12 | 12 |
| 32 | 6 | 10 |
| 33 | 7 | 7 |
| 34 | 9 | 6 |
| 35 | 11 | 7 |
| 36 | 9 | 6 |
| 37 | 11 | 7 |
| 38 | 11 | 15 |
| 39 | 11 | 7 |
| 40 | 11 | 6 |
| 41 | 10 | 14 |
| 42 | 15 | 5 |
| 43 | 8 | 15 |
| 44 | 12 | 11 |
| 45 | 12 | 8 |
| 46 | 9 | 8 |
| 47 | 13 | 6 |
| 48 | 15 | 7 |
| 49 | 10 | 9 |
| 50 | 12 | 12 |

**Statistics**

|  |  |  |
| --- | --- | --- |
|  | Native | React-Native |
| Average | 10.5 | 10.2 |
| Min | 6 | 5 |
| Max | 15 | 16 |

##### Graphical Representation





The statistics produced by this are to be taken with a grain of salt, as the results are heavily influenced by external factors. Due to the nature of a network request, the time it takes to complete a request depends on several factors:   
Broadband Speed (Bandwidth),

Distance to ISP (Internet Service Provider),

Distance to Destination Server,

Code efficiency on the destination server,

Size of the response payload,

Etc.…

However, with these restrictions in mind, the results show a minimal difference between the two development methods. Both had averages of 10 milliseconds, and minimum and maximum values within one milliseconds of one another.

Due to the sporadic nature of Network requests, this feature requires further in-depth testing, which is beyond the scope of this study. I would have liked to compare the two platforms with increasing response payload sizes then measure how long it would take to parse the JSON response, which might bottle-neck the React-Native application. This would show us if there is a response size threshold.

## List View - Infinite Scrolling FPS over 20 seconds

**Description**: This measurement represents the average FPS over 15 seconds of scrolling an infinite list. This infinite list will be pre-populated with a small number of items, then once reaching the end of the list (with a 100 pixel threshold), new Items will be requested from the external API, and added onto the end of the list, allowing the list to continue scrolling. The FPS will be polled and a one second (1 second) interval for a duration of twenty seconds (20 seconds). The result will be the average of the fifteen (15) poll values.  
  
I was unable to reliably automate a scrolling using software. I manually scrolled the list by hand every one second, attempting to keep the same velocity, direction and distance travelled each time.

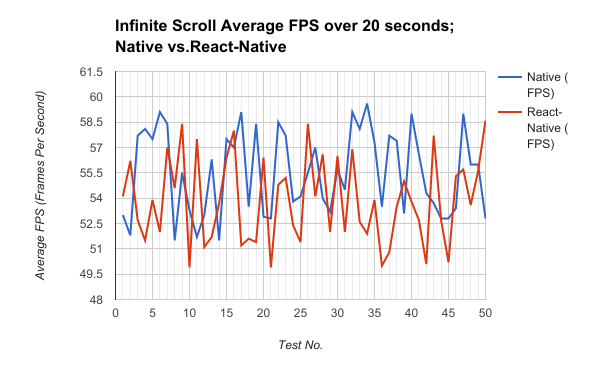
**Key Metric:** FPS (Frames Per Second), The Average scrolling fps over 15 seconds.

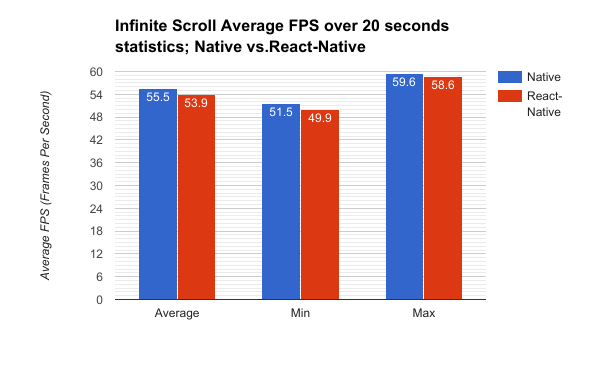
### Results

#### Raw-Data

|  |  |  |
| --- | --- | --- |
| Test No. | Native (FPS) | React-Native (FPS) |
| 1 | 53 | 54.1 |
| 2 | 51.8 | 56.2 |
| 3 | 57.7 | 52.7 |
| 4 | 58.1 | 51.5 |
| 5 | 57.5 | 53.9 |
| 6 | 59.1 | 52 |
| 7 | 58.4 | 57 |
| 8 | 51.5 | 54.6 |
| 9 | 55.5 | 58.4 |
| 10 | 53.3 | 49.9 |
| 11 | 51.7 | 57.5 |
| 12 | 53 | 51.1 |
| 13 | 56.3 | 51.7 |
| 14 | 51.5 | 53.7 |
| 15 | 57.5 | 56.4 |
| 16 | 57 | 58 |
| 17 | 59.1 | 51.2 |
| 18 | 53.5 | 51.6 |
| 19 | 58.4 | 51.4 |
| 20 | 52.9 | 56.4 |
| 21 | 52.8 | 49.9 |
| 22 | 58.5 | 54.8 |
| 23 | 57.7 | 55.2 |
| 24 | 53.8 | 52.4 |
| 25 | 54.1 | 51.4 |
| 26 | 55.5 | 58.4 |
| 27 | 57 | 54.1 |
| 28 | 54 | 56.6 |
| 29 | 53.2 | 52 |
| 30 | 55.7 | 56.5 |
| 31 | 54.5 | 52 |
| 32 | 59.1 | 56.9 |
| 33 | 58.1 | 52.6 |
| 34 | 59.6 | 51.9 |
| 35 | 57.3 | 53.9 |
| 36 | 53.5 | 50 |
| 37 | 57.7 | 50.8 |
| 38 | 57.4 | 53.5 |
| 39 | 53.1 | 55 |
| 40 | 59 | 53.8 |
| 41 | 56.6 | 52.7 |
| 42 | 54.3 | 50.1 |
| 43 | 53.7 | 57.7 |
| 44 | 52.8 | 52.7 |
| 45 | 52.8 | 50.2 |
| 46 | 53.4 | 55.3 |
| 47 | 59 | 55.7 |
| 48 | 56 | 53.6 |
| 49 | 56 | 55.6 |
| 50 | 52.8 | 58.6 |

#### Graphical Representation





## List View: Infinite Scrolling - Time to render new items

**Description**: This measurement specifically deals with the pre-emptive fetching of new data in an infinitely scrolling list, specifically, the time (ms) it takes to render new list items. This process involves making a network request to the external API to retrieve more data, parsing the response, adding the new data to the data source, and re-rendering the list.

**Key Metric:** Time taken in milliseconds to render new list items.

**Iterations:** 50

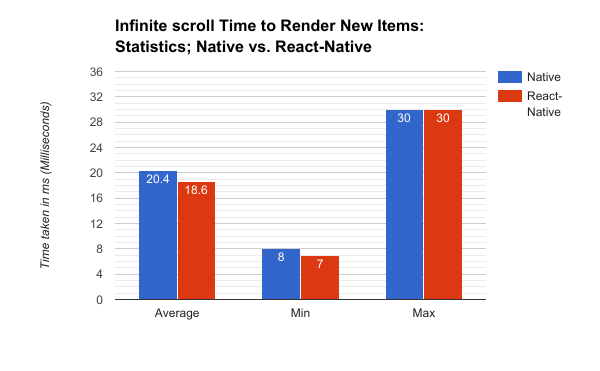
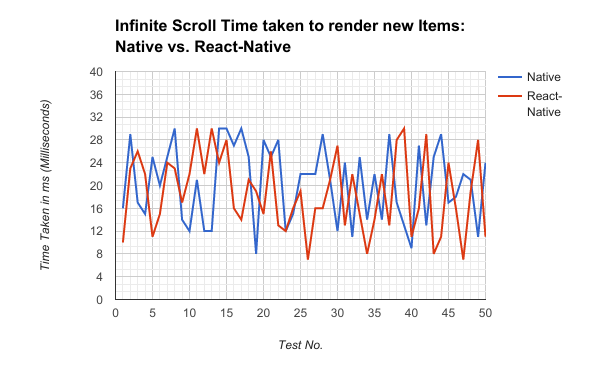
### Results

#### Raw-Data

|  |  |  |
| --- | --- | --- |
| Test No. | Native (ms) | React-Native (ms) |
| 1 | 16 | 10 |
| 2 | 29 | 23 |
| 3 | 17 | 26 |
| 4 | 15 | 22 |
| 5 | 25 | 11 |
| 6 | 20 | 15 |
| 7 | 25 | 24 |
| 8 | 30 | 23 |
| 9 | 14 | 17 |
| 10 | 12 | 22 |
| 11 | 21 | 30 |
| 12 | 12 | 22 |
| 13 | 12 | 30 |
| 14 | 30 | 24 |
| 15 | 30 | 28 |
| 16 | 27 | 16 |
| 17 | 30 | 14 |
| 18 | 25 | 21 |
| 19 | 8 | 19 |
| 20 | 28 | 15 |
| 21 | 25 | 26 |
| 22 | 28 | 13 |
| 23 | 12 | 12 |
| 24 | 15 | 16 |
| 25 | 22 | 19 |
| 26 | 22 | 7 |
| 27 | 22 | 16 |
| 28 | 29 | 16 |
| 29 | 21 | 21 |
| 30 | 12 | 27 |
| 31 | 24 | 13 |
| 32 | 11 | 22 |
| 33 | 25 | 15 |
| 34 | 14 | 8 |
| 35 | 22 | 14 |
| 36 | 14 | 22 |
| 37 | 29 | 13 |
| 38 | 17 | 28 |
| 39 | 13 | 30 |
| 40 | 9 | 11 |
| 41 | 27 | 16 |
| 42 | 13 | 29 |
| 43 | 25 | 8 |
| 44 | 29 | 11 |
| 45 | 17 | 24 |
| 46 | 18 | 16 |
| 47 | 22 | 7 |
| 48 | 21 | 19 |
| 49 | 11 | 28 |
| 50 | 24 | 11 |

|  |  |  |
| --- | --- | --- |
|  | Native | React-Native |
| Average | 20.4 | 18.6 |
| Min | 8 | 7 |
| Max | 30 | 30 |

#### Graphical Representation



## 

## Details View: Forward Scene Change FPS

**Description**: This measurement considers the FPS (Frames per second) during a forward scene change animation. A Forward Scene Change is when a new view is added to the navigator stack, this slides the new view to cover the old view. The state of the old view is persisted, so that the application can return to it via ‘popping’ the new view off of the navigator stack.

**Key Metric:** FPS (Frames per Second) during the Forward scene change event animation.

**Iterations**: 50

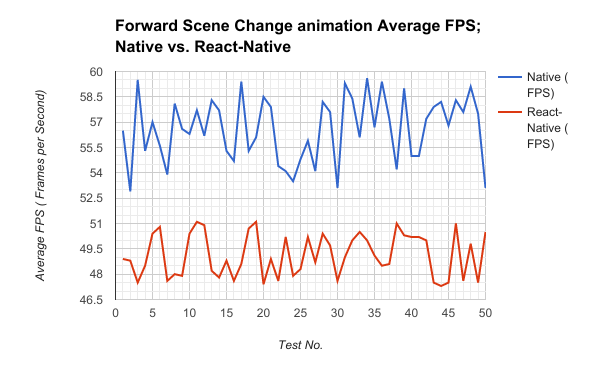
### Results

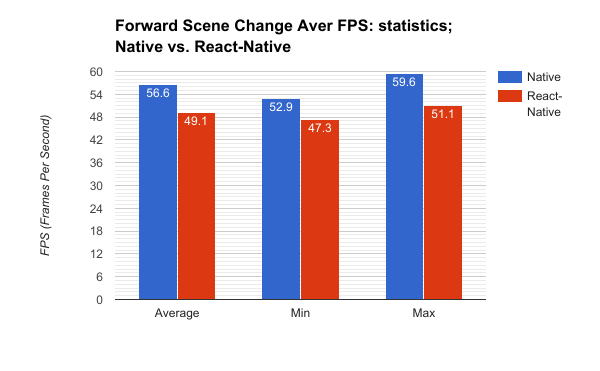
#### Raw-Data

|  |  |  |
| --- | --- | --- |
| Test No. | Native (FPS) | React-Native (FPS) |
| 1 | 56.5 | 48.9 |
| 2 | 52.9 | 48.8 |
| 3 | 59.5 | 47.5 |
| 4 | 55.3 | 48.5 |
| 5 | 57 | 50.4 |
| 6 | 55.6 | 50.8 |
| 7 | 53.9 | 47.6 |
| 8 | 58.1 | 48 |
| 9 | 56.6 | 47.9 |
| 10 | 56.3 | 50.4 |
| 11 | 57.7 | 51.1 |
| 12 | 56.2 | 50.9 |
| 13 | 58.3 | 48.2 |
| 14 | 57.7 | 47.8 |
| 15 | 55.3 | 48.8 |
| 16 | 54.7 | 47.6 |
| 17 | 59.4 | 48.6 |
| 18 | 55.3 | 50.7 |
| 19 | 56.1 | 51.1 |
| 20 | 58.5 | 47.4 |
| 21 | 57.9 | 48.9 |
| 22 | 54.4 | 47.6 |
| 23 | 54.1 | 50.2 |
| 24 | 53.5 | 47.9 |
| 25 | 54.8 | 48.3 |
| 26 | 55.9 | 50.2 |
| 27 | 54.1 | 48.7 |
| 28 | 58.2 | 50.4 |
| 29 | 57.6 | 49.7 |
| 30 | 53.1 | 47.6 |
| 31 | 59.3 | 49 |
| 32 | 58.4 | 50 |
| 33 | 56.1 | 50.5 |
| 34 | 59.6 | 50 |
| 35 | 56.7 | 49.1 |
| 36 | 59.4 | 48.5 |
| 37 | 57.2 | 48.6 |
| 38 | 54.2 | 51 |
| 39 | 59 | 50.3 |
| 40 | 55 | 50.2 |
| 41 | 55 | 50.2 |
| 42 | 57.2 | 50 |
| 43 | 57.9 | 47.5 |
| 44 | 58.2 | 47.3 |
| 45 | 56.8 | 47.5 |
| 46 | 58.3 | 51 |
| 47 | 57.6 | 47.6 |
| 48 | 59.1 | 49.8 |
| 49 | 57.5 | 47.5 |
| 50 | 53.1 | 50.5 |

|  |  |  |
| --- | --- | --- |
|  | Native | React-Native |
| Average | 56.6 | 49.1 |
| Min | 52.9 | 47.3 |
| Max | 59.6 | 51.1 |

#### Graphical Representation





Scene Change

## 

## 

## Details View: Backward Scene Change FPS

**Description**: This measurement considers the FPS (Frames per second) during a backward scene change animation. A Backward Scene Change is when the current view is removed from the navigator stack, and the previous view is shown to the user. The animation consist of the view that is being removed, sliding off of the screen to the right, whilst the next view slides in from the left.

**Key Metric:** FPS (Frames per Second) during the Backward scene change event animation.

**Iterations**: 50

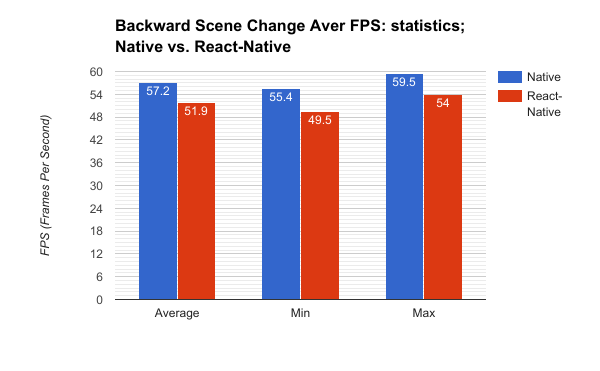
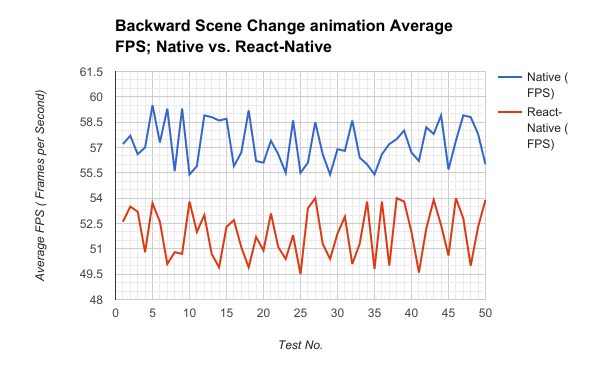
### Results

#### Raw-Data

|  |  |  |
| --- | --- | --- |
| Test No. | Native (FPS) | React-Native (FPS) |
| 1 | 57.2 | 52.6 |
| 2 | 57.7 | 53.5 |
| 3 | 56.6 | 53.2 |
| 4 | 57 | 50.8 |
| 5 | 59.5 | 53.7 |
| 6 | 57.3 | 52.6 |
| 7 | 59.3 | 50.1 |
| 8 | 55.6 | 50.8 |
| 9 | 59.3 | 50.7 |
| 10 | 55.4 | 53.8 |
| 11 | 55.9 | 52 |
| 12 | 58.9 | 53 |
| 13 | 58.8 | 50.7 |
| 14 | 58.6 | 49.9 |
| 15 | 58.7 | 52.3 |
| 16 | 55.9 | 52.7 |
| 17 | 56.7 | 51.1 |
| 18 | 59.2 | 49.9 |
| 19 | 56.2 | 51.7 |
| 20 | 56.1 | 50.9 |
| 21 | 57.4 | 53.1 |
| 22 | 56.6 | 51.1 |
| 23 | 55.5 | 50.4 |
| 24 | 58.6 | 51.8 |
| 25 | 55.5 | 49.5 |
| 26 | 56.1 | 53.4 |
| 27 | 58.5 | 54 |
| 28 | 56.6 | 51.3 |
| 29 | 55.4 | 50.4 |
| 30 | 56.9 | 51.9 |
| 31 | 56.8 | 52.9 |
| 32 | 58.6 | 50.1 |
| 33 | 56.4 | 51.3 |
| 34 | 56 | 53.8 |
| 35 | 55.4 | 49.8 |
| 36 | 56.6 | 53.8 |
| 37 | 57.2 | 50 |
| 38 | 57.5 | 54 |
| 39 | 58 | 53.8 |
| 40 | 56.7 | 52 |
| 41 | 56.2 | 49.6 |
| 42 | 58.2 | 52.2 |
| 43 | 57.8 | 53.9 |
| 44 | 58.9 | 52.4 |
| 45 | 55.7 | 50.6 |
| 46 | 57.4 | 54 |
| 47 | 58.9 | 52.8 |
| 48 | 58.8 | 50 |
| 49 | 57.8 | 52.3 |
| 50 | 56 | 53.9 |

|  |  |  |
| --- | --- | --- |
|  | Native | React-Native |
| Average | 57.2 | 51.9 |
| Min | 55.4 | 49.5 |
| Max | 59.5 | 54 |

#### Graphical Representation



## List view - Delete Item from top of list: Time taken to redraw list

**Description:** This measurement is taken from the amount of time taken (ms) to redraw a large list of items, after removing a single item from the top of this list. This done to demonstrate React-Native’s emphasis on only updating the view elements that have changed rather than re-rendering the entire view, this measurement will determine if this makes any effective difference.

**Key Metrics:** Time taken (in milliseconds), to re-draw the list, after an item has been removed.

**Iterations:** 50

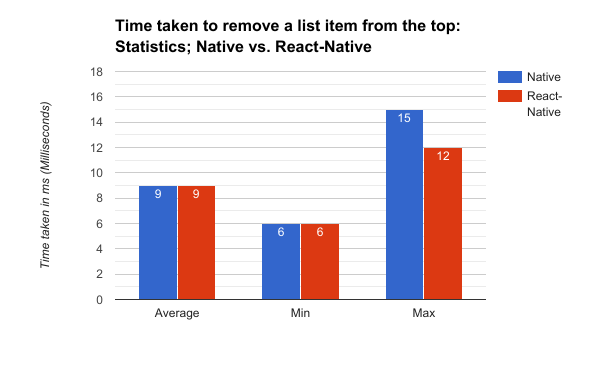
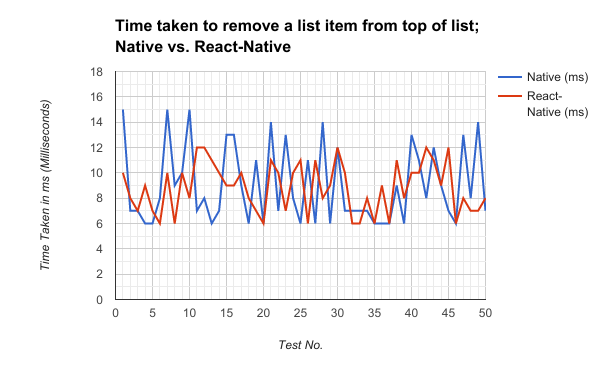
### Results

#### Raw-Data

|  |  |  |
| --- | --- | --- |
| Test No. | Native (ms) | React-Native (ms) |
| 1 | 15 | 10 |
| 2 | 7 | 8 |
| 3 | 7 | 7 |
| 4 | 6 | 9 |
| 5 | 6 | 7 |
| 6 | 8 | 6 |
| 7 | 15 | 10 |
| 8 | 9 | 6 |
| 9 | 10 | 10 |
| 10 | 15 | 8 |
| 11 | 7 | 12 |
| 12 | 8 | 12 |
| 13 | 6 | 11 |
| 14 | 7 | 10 |
| 15 | 13 | 9 |
| 16 | 13 | 9 |
| 17 | 9 | 10 |
| 18 | 6 | 8 |
| 19 | 11 | 7 |
| 20 | 6 | 6 |
| 21 | 14 | 11 |
| 22 | 7 | 10 |
| 23 | 13 | 7 |
| 24 | 8 | 10 |
| 25 | 6 | 11 |
| 26 | 11 | 6 |
| 27 | 6 | 11 |
| 28 | 14 | 8 |
| 29 | 6 | 9 |
| 30 | 12 | 12 |
| 31 | 7 | 10 |
| 32 | 7 | 6 |
| 33 | 7 | 6 |
| 34 | 7 | 8 |
| 35 | 6 | 6 |
| 36 | 6 | 9 |
| 37 | 6 | 6 |
| 38 | 9 | 11 |
| 39 | 6 | 8 |
| 40 | 13 | 10 |
| 41 | 11 | 10 |
| 42 | 8 | 12 |
| 43 | 12 | 11 |
| 44 | 9 | 9 |
| 45 | 7 | 12 |
| 46 | 6 | 6 |
| 47 | 13 | 8 |
| 48 | 8 | 7 |
| 49 | 14 | 7 |
| 50 | 7 | 8 |

|  |  |  |
| --- | --- | --- |
|  | Native | React-Native |
| Average | 9 | 8.8 |
| Min | 6 | 6 |
| Max | 15 | 12 |

#### Graphical Representation



## 

## Listview - Delete Item from bottom of list: Time taken to redraw list

**Description:** This measurement is taken from the amount of time taken (ms) to redraw a large list of items, after removing a single item from the bottom of this list. This done to demonstrate React-Native’s emphasis on only updating the view elements that have changed rather than re-rendering the entire view, this measurement will determine if this makes any effective difference.

**Key Metrics:** Time taken (in milliseconds), to re-draw the list, after an item has been removed.

**Iterations:** 50

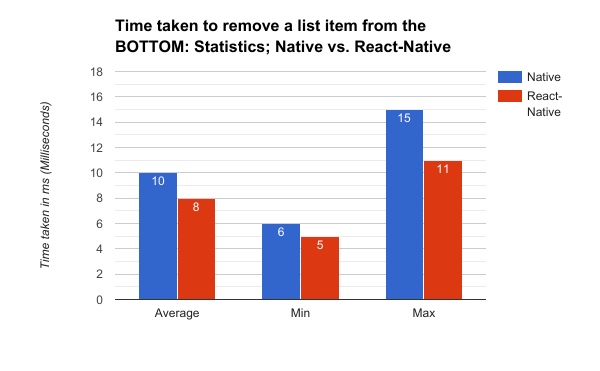
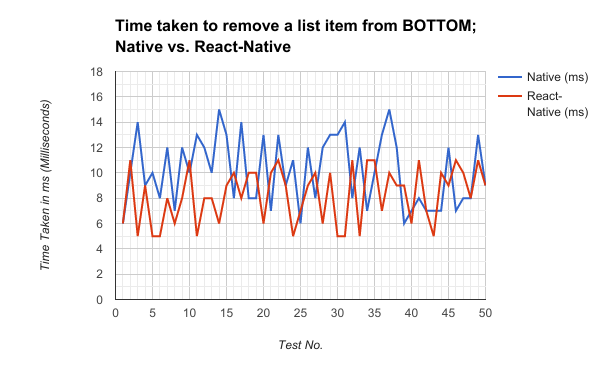
### Results

#### Raw-Data

|  |  |  |
| --- | --- | --- |
| Test No. | Native (ms) | React-Native (ms) |
| 1 | 6 | 6 |
| 2 | 10 | 11 |
| 3 | 14 | 5 |
| 4 | 9 | 9 |
| 5 | 10 | 5 |
| 6 | 8 | 5 |
| 7 | 12 | 8 |
| 8 | 7 | 6 |
| 9 | 12 | 8 |
| 10 | 10 | 11 |
| 11 | 13 | 5 |
| 12 | 12 | 8 |
| 13 | 10 | 8 |
| 14 | 15 | 6 |
| 15 | 13 | 9 |
| 16 | 8 | 10 |
| 17 | 14 | 8 |
| 18 | 8 | 10 |
| 19 | 8 | 10 |
| 20 | 13 | 6 |
| 21 | 7 | 10 |
| 22 | 13 | 11 |
| 23 | 9 | 9 |
| 24 | 11 | 5 |
| 25 | 6 | 7 |
| 26 | 12 | 9 |
| 27 | 8 | 10 |
| 28 | 12 | 6 |
| 29 | 13 | 10 |
| 30 | 13 | 5 |
| 31 | 14 | 5 |
| 32 | 8 | 11 |
| 33 | 12 | 5 |
| 34 | 7 | 11 |
| 35 | 10 | 11 |
| 36 | 13 | 7 |
| 37 | 15 | 10 |
| 38 | 12 | 9 |
| 39 | 6 | 9 |
| 40 | 7 | 6 |
| 41 | 8 | 11 |
| 42 | 7 | 7 |
| 43 | 7 | 5 |
| 44 | 7 | 10 |
| 45 | 12 | 9 |
| 46 | 7 | 11 |
| 47 | 8 | 10 |
| 48 | 8 | 8 |
| 49 | 13 | 11 |
| 50 | 9 | 9 |

|  |  |  |
| --- | --- | --- |
|  | Native | React-Native |
| Average | 10.1 | 8.2 |
| Min | 6 | 5 |
| Max | 15 | 11 |

#### Graphical Representation



## 

### Listview - Delete Item from middle of list: Time taken to redraw list

**Description:** This measurement is taken from the amount of time taken (ms) to redraw a large list of items, after removing a single item from the middle of this list. This done to demonstrate React-Native’s emphasis on only updating the view elements that have changed rather than re-rendering the entire view, this measurement will determine if this makes any effective difference.

**Key Metrics:** Time taken (in milliseconds), to re-draw the list, after an item has been removed.

**Iterations:** 50

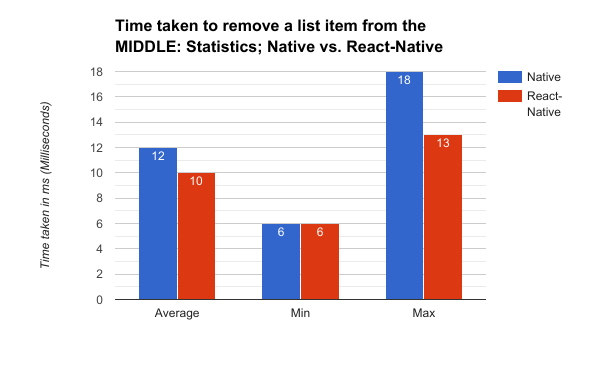
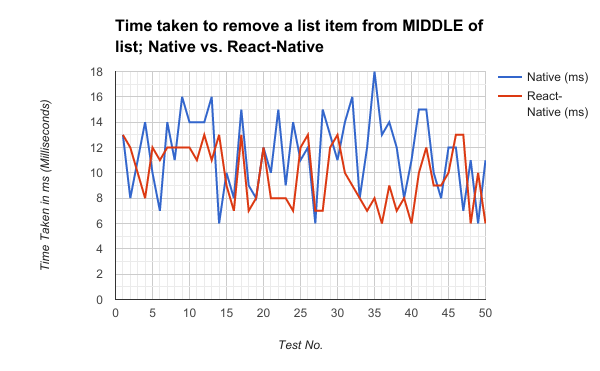
#### Results

##### Raw-Data

|  |  |  |
| --- | --- | --- |
| Test No. | Native (ms) | React-Native (ms) |
| 1 | 13 | 13 |
| 2 | 8 | 12 |
| 3 | 11 | 10 |
| 4 | 14 | 8 |
| 5 | 10 | 12 |
| 6 | 7 | 11 |
| 7 | 14 | 12 |
| 8 | 11 | 12 |
| 9 | 8 | 12 |
| 10 | 14 | 12 |
| 11 | 14 | 11 |
| 12 | 14 | 13 |
| 13 | 6 | 11 |
| 14 | 6 | 13 |
| 15 | 10 | 9 |
| 16 | 8 | 7 |
| 17 | 15 | 13 |
| 18 | 9 | 7 |
| 19 | 8 | 8 |
| 20 | 12 | 12 |
| 21 | 10 | 8 |
| 22 | 15 | 8 |
| 23 | 7 | 8 |
| 24 | 14 | 7 |
| 25 | 11 | 12 |
| 26 | 8 | 13 |
| 27 | 6 | 7 |
| 28 | 7 | 7 |
| 29 | 10 | 12 |
| 30 | 6 | 13 |
| 31 | 8 | 10 |
| 32 | 6 | 9 |
| 33 | 8 | 8 |
| 34 | 7 | 7 |
| 35 | 10 | 8 |
| 36 | 13 | 6 |
| 37 | 14 | 9 |
| 38 | 12 | 7 |
| 39 | 8 | 8 |
| 40 | 11 | 6 |
| 41 | 15 | 10 |
| 42 | 15 | 12 |
| 43 | 10 | 9 |
| 44 | 8 | 9 |
| 45 | 12 | 10 |
| 46 | 12 | 13 |
| 47 | 7 | 13 |
| 48 | 11 | 6 |
| 49 | 6 | 10 |
| 50 | 11 | 6 |

|  |  |  |
| --- | --- | --- |
|  | Native | React-Native |
| Average | 10.2 | 9.8 |
| Min | 6 | 6 |
| Max | 15 | 13 |

##### Graphical Representation



# Discussion

## Where did React-Native perform better?

### Time

The React-Native application was approximately 16% quicker to develop than the Native application. This figure is subjective to developer ability, and familiarity with the platform and framework.

### Cost

The median Hourly Rate of a React-Native developer is approximately 10% higher than an android developer. This may be because the technology is still quite new, and so React-Native practitioners are considered specialists. This could also be because of the rising popularity of the framework, as React-Native becomes more popular, there is a higher demand for React-Native specialists.

However, we need to consider that React-Native is a cross-platform framework, and that the application that I developed also works on iOS. This means that potentially that using React-Native can produce two applications for the price of one. Which is appealing to any business or individual who have a low budget.

### Performance

React-Native performed consistently worse throughout all tests related to Animations and FPS (Frames Per Second). However, the difference between the two seem to depend on how complex the animation is, or whether the phone is at a heavy load, this warrants further investigation.

Scene change animations (Forward scene change and Backward scene changes) are far worse on the React-Native app. The Navigator component is one of the oldest APIs in the React-Native framework, thus it does not leverage new optimisations in React-Native’s animations, and the developer does not have any control over how the animations are done using this component. This could be solved by either developing a custom Navigator components, with bespoke scene change animations, or use one of the numerous third-party Navigator components. Unfortunately, testing third-party components is beyond the scope of this project, even though using community developed components is widely accepted and encouraged by Facebook and the React-Native community.

The reason for the low performance on animations is React-Native’s bridge bottleneck, as a large amount of data needs to be passed from the Native modules, to the JavaScript thread, and having to “translate” the data from a Java Object into JSON takes a fair amount of time. Thus, leading to dropped frames (as new data was not render within the 1 millisecond window) and subsequently lower FPS.

This would indicate that React-Natives animations would perform substantially slower when the phone is under heavy load / or the application is doing an intensive task whilst simultaneously displaying an animation.

# Conclusion

React-Native is a powerful framework, and offers many advantages of traditional Native Android development. In this study, the performance of the React-Native application had an overall performance loss of 12%, compared to the native application. The performance deficit was most noticeable when considering intensive animations, and least noticeable during static-retendering. However, the React-Native application was 16.9% faster to develop. When both development times were combined with the median hourly salary of their respective developers, the React-Native application was estimated to be 10 % cheaper to develop than the Native application. We also need to acknowledge that that React-Native is a cross platform framework which can produce both Android and iOS applications simultaneously, which this study estimates reduces both the cost and development-time of the application by over 50%.

To answer the question, Does React-Native produce any statistical benefit over Native application development? The answer is yes, as we can see from the above statistics that React-Native has better development time and cost predictions than the Native solution.  
This however, is better placed in a business context, and the aim of the application being developed.  
A business or an individual whose application requires the most performance possible from the device, for example a Game, which uses many animations and intensive calculations (physics, etc..), should prefer to use the traditional Native development method, as the performance loss of React-Native will greatly hinder the application. However, if performance is not as important as minimising the resources used, then React-Native proves to be far more advantageous.

All code can be found under the following public repository: <https://github.com/Riglerr/330COM>

# Reflection

Due to lack of resources I was unable to test using iOS. The apps may behave differently on different platforms, and would potentially change the results of both development time, cost and performance. As the largest attractions of React-Native is cross-platform functionality, the study would gain tremendous worth from including the iOS platform.

I would like to further explore the effect of React-Native when the phone is under Heavy load, and attempt to compare it to Native applications at different CPU utilisation percentages. This would mean I would be able to establish a threshold at which the React-Native application is no longer “acceptable”.

The network request test, proved to be entirely inconclusive, and I would have liked to do more testing on the subject by increasing the response payload size, and see if there is a threshold at which React-Native becomes slower (if at all). As the reponse data needs to be sent over a bridge to the JavaScript Thread, which could potentially be a considerable bottle neck.

I Missed the obvious metric of CPU & Memory profiling for these applications, is one type of development better for older phones with less resources? Which uses more RAM? Which uses highest CPU %?

To summarise the reflections, I feel that this is a good initial start to the study, however the scope of the project was too broad. I would like to have produced more technical data around each feature, and go *deeper*  into React-Native and identify the critical bottle-necks and areas lacking in performance.

# Presentation Feedback

The primary feedback from my feedback was the need to contextualize the need for this study.

As my project mostly consisted of a performance comparison of Native and React-Native development.

Whilst a comparison of React-Native to Native development is not an area that has been thoroughly researched before in the way that my study does. However, initially it was not clear who would benefit from this study, as the intended results would only be technical data surrounding the performance of the applications. For this reason, I decided to apply a business context to the project. I chose to focus on why businesses make mobile-applications, how one could measure the success or failure of a mobile-application, and what are the most contributing factors to the success of a mobile-application in term of its technical development. After some research (detailed in the introduction and literature review), I extended the scope of the project to also consider the development-time, cost and stability of both application development methods. With these additional metrics, I could create substantial links between them, and provide meaningful insight into the situations where each development method either excels or falls behind. This added more depth to the study as well as a clearly defined target audience.

To support this new contextualization, I extended the literature review and report introduction to include details about the relationship between businesses and mobile-application and how their development methodologies can influence the success or failure of the application, and how cross-platform frameworks such as React-Native can help.

The secondary feedback was to research if there are any similar experiments or studies that have been done in the past, so that I may be able to contrast my methodologies with theirs.

The tertiary feedback was to ensure that the report and my methods clearly indicate the derivative nature of some of the metrics I produced, specifically the cost metric, as it is derived from the development-time. Also, as I noticed quite late on in the process some flaws in the methodology of the project, I need to have a section in the report that details these flaws and offer some potential solutions to them.

# References

1. PerfectoMobile, 2014. Why Mobile Apps Fail. [PDF] Available at: <http://www.retail-systems.com/rs/perfectomobile/why-mobile-apps-fail-report.pdf>
2. Statista.com, 2016. Number of smartphone users worldwide from 2014 to 2020 (in billions) [Website] Available at: <https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/>
3. Elkin, Ariel. (2016). Why I'm Not A React Native Developer - Arielelkin.github.io [online] . Available at: <https://arielelkin.github.io/articles/why-im-not-a-react-native-developer.html> /[Accessed 1 Mar. 2017].
4. Novoda . 2017 React Native: Is It The End Of Native Development? – Novoda.com. [online] . Available at: [https://www.novoda.com/blog/is-it-the-end-of-native-development/](https://www.novoda.com/blog/is-it-the-end-of-native-development%20/) [Accessed 2 Mar. 2017].
5. Kol, Tal. 2016. Performance Limitations Of React Native And How To Overcome Them. Medium.com. [online] Available at:   
   <https://medium.com/@talkol/performance-limitations-of-react-native-and-how-to-overcome-them-947630d7f440#.ayr73y94x/> [Accessed 3 Mar. 2017.]
6. Panneer, Sriraman. 2016 What We Learned After Using React Native For A Year - Hashnode. Hashnode.com. [online] Available at: <https://hashnode.com/post/what-we-learned-after-using-react-native-for-a-year-civdr8zv6058l3853wqud7hqp/> [Accessed: 3 Mar. 2017]
7. Narusis, Vytenis. 2016. Pros And Cons Of React Native - A Course Of Building Native Apps With Javascript. - Devbridge.com. [online] Available at: <https://www.devbridge.com/articles/pros-cons-of-react-native-crash-course/>   
    [Accessed at: 2 Mar. 2017.]
8. Keerti. 2016 React Native At Walmartlabs – Walmartlabs. Medium.com [online] Available at: <https://medium.com/walmartlabs/react-native-at-walmartlabs-cdd140589560#.e8esee7za> [Accessed 4 Mar. 2017.]
9. Kohan, Bernard.2015. Mobile App, Iphone App Development Cost, Pricing. Comentum.com. [online] Available at: <http://www.comentum.com/mobile-app-development-cost.html>   
   [Accessed 4 Mar. 2017.]
10. Chen, Robin. 2016 Using React-Native: One year later, Discord.com. [online] Available at:<https://blog.discordapp.com/using-react-native-one-year-later-91fd5e949933#.sqsegx7nm> [Accessed 1 Mar. 2017]
11. Gomez, G. (2017). React-Native: The Good, The Bad & The Ugly   
    -3 SIDED CUBE.[online] 3 SIDED CUBE. Available at:  
     <https://3sidedcube.com/blog/2016/04/react-native-the-good-the-bad-the-ugly/>   
    [Accessed 8 Mar. 2017].
12. BBC, (2016). Domino's Pizza profits boosted by mobile app. [Online] BBC News. Available at: <http://www.bbc.co.uk/news/business-35714501>
13. Swrve, (2016. One Third of Mobile App Engagements last less than one minute. [Online] Available at: <http://www.marketwatch.com/story/one-third-of-mobile-app-engagements-last-less-than-one-minute-2014-07-17>
14. Riktam Tech, (2015). *Why Mobile applications fail .*[online] Available at: <http://www.riktamtech.com/reasons-why-mobile-apps-fail/>
15. ITJobsWatch (2017). Android Developer Salary Statistics. [Online]  
     Available at: <https://www.itjobswatch.co.uk/contracts/uk/android%20developer.do>
16. ITJobsWatch (2017). React-Native Developer Salary Statistics [online] Available at: <https://www.itjobswatch.co.uk/contracts/uk/react%20native.do>

# Appendix

All code can be found under the following public repository: <https://github.com/Riglerr/330COM>

# Detailed Project Proposal

**­­­300/303COM Detailed project proposal**

The objective of the detail project proposal is to help you refine your general research question down to a well-focused and achievable piece of practical research work.

The first section: “Defining your research project” focuses on your research question and the plan for conducting your primary method. The second section: “Abstract and Literature Review” is to help you identify current academic sources of literature that are highly relevant to your project and to help you get a head-start in producing your literature review.

Your detailed project proposal will be graded in the second semester – however, it is highly recommended that you submit it by the end of the first semester (04/01/2016) in order to obtain detailed supervisor feedback on your project.

There is no suggested word length for the detailed proposal – although 2000-2500 words would be in order.

The Detailed Project Proposal is worth 20% of the project mark.

**300/303COM Detailed Project Proposal**

|  |  |
| --- | --- |
| First Name: | Robert |
| Last Name: | Rigler |
| Student Number: | 4939377 |
| Supervisor: | Dr. Faiyaz Doctor |

Section one: Defining your research Project

**1.1 Detailed research question**

**Help:** Your detailed research question is the statement of a problem within the computing domain which you will address in your project. Refining the research question involves narrowing down an initial question until it is answerable using a primary research method(s) that you will conduct during the time of your project. The refined research question must not be so general that it is answerable with a yes or no answer. It must not be so broad that you would be unable to achieve a solution during your project. The key to this is BEING SPECIFIC: Narrow down the method or technology you will use, narrow down the group that the question refers to (localize a general question) If the project is still ‘too big’, can you think of a way to work on a part of the problem? Avoid using words that cannot be measured, by you, without a huge research budget e.g. 'effects on society', 'effects on business'. *Example:* The initial question "Does cloud computing effect business" needs narrowing down *(for a start the answer is yes) W*hat is meant by cloud computing? Or 'effect'? Or 'business', in this question? Refining this first question will involve narrowing it down to something you, personally, can measure. A refined version of this question might be: "Does implementing a cloud based voting system improve the speed of decision making in a small company in Coventry?" This refined question is implementable: You can now identify a small company to work with, document their current decision making processes, implement a cloud based voting system, compare decision making speeds over a limited time period (say 1 month) and evaluate your findings. *A small piece of genuinely new knowledge is produced.*

|  |
| --- |
| Does developing Android mobile applications in the cross-platform development framework React-Native produce any quantitative benefits in terms of performance, development time, cost, and stability compared to developing applications exclusively in the platform native language (Java)? |

**1.2 Keywords**

**Help:** Include up to 6 keywords separated by a semi-colon; what keywords are appropriate to describe your project in an online database like Google Scholar? Keywords should include the general research area and the specific technologies you will be working with. *Example.* A project that proposes a novel way of visualising large amounts of twitter feed data may have the keywords: Data visualisation; twitter; hashtags; database design; graphics libraries. For further help take a look at the ACM keywords list <http://www.computer.org/portal/web/publications/acmtaxonomy>

|  |
| --- |
| react-native; android; performance; development-time; mobile-application; comparison; |

**1.3 Project title**

**Help:** The project title is a statement based on your detailed research question. For example, the research question *'to what extent does a mobile application reduce the number of errors made in class registers at Coventry University in comparison to current paper based registers'* may be stated in the project title*: "A Wi-Fi driven mobile application for large group registers using iBeacons".*

|  |
| --- |
| A comparative analysis of developing Android mobile applications using the native Android language Java, compared to using the cross-platform development framework React-Native. |

**1.4 Client, Audience and Motivation:**

**Help:** Why is this project important? To whom is this project important? A research project must address a research question that generates a small piece of new knowledge. This new knowledge must be important to a named group or to a specific client (such as a company, an academic audience, policy makers, people with disabilities) to make it worthwhile carrying out. This is the ***motivation*** for your project. In this section you should address who will benefit from your findings and how they will benefit. Example: If you intend to demonstrate that a mobile application that automates class registers at Coventry University will be more efficient than paper based registers - the group who would be interested in knowing/applying these findings would be both academic and administrative staff at Coventry University and they would benefit by time saved and a reduction in their administrative workload. If you are making a business case for an organization explain how the organisation will benefit from your findings.

|  |
| --- |
| **Motivation**  React-Native is a relatively new cross-platform mobile application development framework for Android and iOS. Its combination of the popular JavaScript-based web-design framework: React, and Platform Native development has gained a huge amount of interest from hobbyists and professionals alike, many of whom believe this new framework to be a superior method of creating mobile applications. However, despite its popularity, very little formal research has been done and almost no quantitative data has been produced to properly compare the two types of development. There is a large amount of uncertainty concerning the performance of React-Native, and whether the performance trade-off is worthwhile when also considering the reduced development time and cross-platform aspects of React-Native.  The goal of this project is to produce accurate, unbiased analytical data and use it to objectively compare the two types of development. This data should be able to alleviate some of the uncertainty surrounding the subject and act as a reliable resource for individuals, businesses, and development companies when assessing the usefulness of the React-Native framework.  **Audience**  This research and the conclusions therein are intended for use by any individual or company who are deciding whether to using Native application development or using React-Native. The data I produce aims to alleviate the ambiguity between the two development techniques.  Specifically, I am looking into the 4 areas of app development which are most relevant to businesses: Performance, Cost, Stability, and Development-Time.  Depending on the application in question and the resources available (time and budget, etc...), the choice between using traditional app development and using React-Native becomes difficult as there is little data supporting either side of the argument. |

**1.5 Primary Research Plan**

**Help:** This is the plan as to how you will go about answering your detailed research question - It must include a primary research method (an extended literature review is not an acceptable primary method). Think and plan logically. Primary methods may include experiments, applications or software demonstrators, process models, surveys, analysis of generated data …  
  
Example: In the class register example above "to what extent does a mobile application reduce the number of errors made in class registers at Coventry University in comparison to current paper based registers" - the research plan may involve: 1) Collecting and analysing paper based registers in a given class on five occasions. 2) Identifying the error rate average on these occasions 3) Designing and implementing a mobile application that automatically records attendance in class. 4) Deploying the application in the class on five occasions. 5) Identifying the error rate average of the mobile application on these occasions. 6) Comparison of data and summary of findings.

|  |
| --- |
| The primary research plan for this project will be to create two identical mobile applications and then compare them to each other. One of these applications will be developed using the traditional Java method, the other will be using the cross-platform framework React-Native.  Firstly I am going to design the application, then I will reduce this design into a set of *features*, and those features will be broken down into *tasks.* Using this structure I can then proceed to develop the applications, and record my progress and the relevant metrics for each task and feature, thus producing a set of data for each application which can be directly compared to each other.  The following areas will be compared between the two applications:   * Performance * Development-time * Stability * Cost (Derived)   **Performance**  For most features, a performance metric will be directly measurable. E.g. For measuring the performance of animations, the metric would be average FPS (Frames per Second) over the duration of the animation.  For large list manipulation I.e. (Removing an item from the middle of a large list), the metric would be TTD (Time to Draw) which is the time it took to remove the element from the list and redraw the screen with the new data.  **Development-Time**  For the development-time metric, I will record the time it took to complete a feature. A feature is "Completed" when it passes the set of unit tests that have been created for that feature (Both applications will use the same unit tests, however, they will be implemented using the platform respective unit-testing framework).  Being very familiar with both Java and React-Native, I will be able to efficiently implement both applications.  **Stability**  Stability is of great importance when creating any application, I will measure stability by recording each time a feature implementation cause a previously successful unit test to fail. Any bugs not caught by the unit tests will also be recorded in this metric.  I can produce the following metrics:   * Bugs (number of reworks) per feature ( **B/F** ). * Broken lines of code(SLOC) per feature. * Time spent fixing Bugs.   All of the above three metrics will also be produced for the application as a whole, rather than just on a per feature basis. |

This is the end of section one.

Section Two: abstract and Literature review (1500 words suggested)

**2.1 Abstract**

**Help:** An abstract is a short summary of a research project that enables other researchers to know if your report or research paper is relevant to them without reading the whole report. It is usually written retrospectively so that it can include findings and results. It is fully expected that you will rewrite your abstract when you come to write your final paper. For now, you should write an abstract of about 250 words that define the project described in section one. Before writing your abstract you MUST read some abstracts from conference or journal papers on *Google Scholar* or from *portal.acm.org* (to understand their style) and then provide your own abstract that outlines what your question is and what you 'did' to answer it.

|  |
| --- |
| React-Native is a cross-platform mobile application development framework created by Facebook in 2015. It allows a developer to simultaneously create mobile applications for both the Android and iOS platforms in JavaScript instead of the platform native language (Java or Objective-C respectively). React-Native’s popularity stems from its intelligent combination of native and non-native code using “Bridges”. Bridges are a way of exposing native APIs to the JavaScript thread, allowing the utilization of the platforms’ high-performance native modules. This allows React-Native to achieve comparable performance to purely native applications whilst using the arguably simpler application design structure of React and JavaScript as well as having the benefit of creating applications for both platforms at once. React-Native operates very differently to other cross-platform technologies such as Cordova and Xamarin, which use an embedded web browser to render the application in HTML and CSS as if it were a normal web application. Whilst this method is considered cross-platform it sacrifices performance and freedom of design. Those who use React-Native claim to have seen reduced development time and cost, whilst sacrificing very little in terms of performance. These claims, however, are backed by very little evidence and almost no statistical data has been produced. Most of the available data comes from blogs that have a little objective evaluation of the development methods, or do not consider any special business requisites for the application, such as high-performance requirements. Businesses and individuals who are deciding which method to use when creating a new mobile application will find it difficult to form an accurate, unbiased decision using the data available, and may waste a large amount of time and money using a development method that is entirely unsuitable for their needs. The primary objective of this research is to fill the substantial gap in the data used to compare the two development methods. Using this data, we can accurately assess if using React-Native to develop mobile applications produces any substantial improvement in performance, development time, cost, and stability compared to using purely platform native development. These are arguably the most important areas of concern for any business looking to develop a mobile application. The research involves creating two identical Android mobile applications and individually assessing each aspect of the application. The measurements will be taken for each feature of the application, so to produce a fine-grained analysis as well as considering the two methods on a broader scale. |

**2.2 Initial/Mini Literature Review (500 words – 750 words)**

**Help:** A literature review is a select analysis of current existing research which is relevant to your topic, showing how it relates to your investigation. It explains and justifies how your investigation may help answer some of the questions or gaps in this area of research. A literature review is not a straightforward summary of everything you have read on the topic and it is not a chronological description of what was discovered in your field. Use your literature review to:

• compare and contrast different authors' views on an issue  
• criticise aspects of methodology, note areas in which authors are in disagreement  
• highlight exemplary studies  
• highlight gaps in research  
• show how your study relates to previous studies

|  |
| --- |
| My research focuses on the statistical benefits of creating mobile applications in React-Native, as opposed to using purely native development. The aim of this research is to be able to compare both types of development and assess each of them in key areas of mobile development. These areas include: Performance, Development Time, Cost, and Stability. React-Native was created by Facebook in 2015, and the framework is currently still in an ‘Unstable’ state as an official release version is yet to be produced (The current version is 0.42). Despite its relative youth compared to native mobile application development, React-Native has exceeded google search trends for both “android development”, and “iOS development”.  **Performance**  Generally, most authors agree that there is a noticeable performance deficit when using React-Native. However, there is some ambiguity over where this deficit occurs. According to Vytenis Narusis, a Junior mobile developer at Devbridge, the greatest performance impact comes from the JavaScript Language itself: “It’s natural that JavaScript code is not as efficient for calculation-intensive tasks, and there is an overhead when JavaScript is controlling native elements.” (Narusis), He also states that “In performance, React Native stays behind an optimised native application.”. No statistical data has been given to support this argument, one could surmise that a *low-level* compiled language such as Java, will perform better than a JIT (Just in time) compiled language such as JavaScript, however this is purely conjecture as it is impossible to know the optimisations made in either language, and often performance will have a greater dependence on *how* a problem is implemented, rather than in what language it is implemented.  Others argue that the performance deficit stems from the Single Threaded nature of JavaScript. React-Native runs all JavaScript code on a single separate thread, as JavaScript cannot be declaratively parallelised. React-Native utilises Native API’s and components by passing data from the JavaScript thread to the Native thread, this is known as a *bridge.* Many authors indicate that these bridges are the source of React-Natives performance bottle-necks. In the blog article *Performance Limitations Of React Native And How To Overcome Them*, by Tal Kol, a Start-up founder and Senior Developer at Wix.com, he states that: “*The performance bottleneck often occurs when we move from one realm to the other. In order to architect performant React Native apps, we must keep passes over the bridge to a minimum.”* (Kol, 2016). He refers to the Native threads and JavaScript thread as separate *Realms,* he advocates to minimise the use of Bridges. This is a better indicator of where the performance problem lies, the use of *bridges* in React-Native is unavoidable, and thus careful optimisation is imperative to both the framework itself, and the developer using React-Native. The *bridging* performance bottleneck seems to be especially common when attempting to create animations solely using JavaScript. Sriraman Paneer (a mobile application developer a Witworks), states in his article “*What we learned after using React Native for a year*” that “Animation APIs run all the processes on the JS Thread. Consequently, we had faced some performance issues while using these APIs.” (Panneer). A similar claim is made by Robin Chen, a senior developer at Discord, who chose to migrate their VOIP application from Native to React-Native. “The Animated library also cannot deliver the animations as smooth as the native while doing heavy duty works on the JS thread.” (Chen).  After reading many articles, and using the sources that I have given here, I am left with more questions about React-Native performance that when I began. It seems clear that there is a definitive decrease in performance when using React-Native, however, it is not clear about how large or small that sacrifice is. E.g. How much of a difference In FPS is there between Animations in Native and React-Native? How much longer would an intensive calculation take using JavaScript than Java? Does the problem lie in the developers implementation rather than the framework? My research aims to fill these gaps in the knowledge, as the ambiguous information available is simply not enough to make an informed decision about Whether to use React-Native of not.  **Development-Time**  An important factor of creating any application is development-time, especially for prototyping new features, or creating an MVP (Minimum viable product). How quickly an application can be developed may be quintessential to a business model, or may be entirely irrelevant. In cases, development-time has a direct correlation to the cost of developing that application.  Being a Hybrid application framework (i.e. Can develop applications for both iOS and Android simultaneously, using the same code base), one would expect that the development time for a React-Native to be half that of developing a native application for both platforms. However, my research will only be focusing on the Android platform, so whilst the cross-platform development is of great importance, I must primarily focus on single platform development.  Greg Gomez, a Project Manager at 3-Sided-Cube said that: “*The initial benefit that strikes me is the ability to prototype with even more rapidity than previously. React-Native will offer developers unparalleled speed in developing high fidelity prototypes, so ideas can be validated, assumptions debunked and implementations battle-tested in record time.”* (Gomez, 2017).This is certainly high praise for React-Native, however he has not stated why React-Native had a faster development-time. He later advocates the re-usability aspect of React-Native, *“Developing a component once allows us to reuse it easily in other projects, meaning that over time you’ll amass a nice library of drop-in components to build prototypes from. It means that time spent re-inventing the wheel, can be spent solving your client’s problems and delivering value.”*. This is a far more important than it seems, React, and therefore React-Native, has a component based architecture, which advocates creating components of the application independently. This allows for a minimal amount of dependency between different parts of the application, meaning that a developer could develop generic components that can be used in anywhere in any application, or perhaps use open-source components that others have developed. Whilst this is still possible in Native development, it is not possible to the extent that React-Native allows composition of components.  In Native development, every time that new code is implemented, changed or updated, the entire application needs to be re-compiled, if the application is large this could take quite some time, and quickly becomes aggravating, especially when the code changes are minimal. React-Native being able to take advantage of both Native and non-native JavaScript, has an impressive feature called *Hot-Reloading*. *Hot-Reloading,* allows any changes to the JavaScript code to be almost instantaneously reflected in the application; “*React Native’s feedback loop is bewitchingly low. It takes less than one or two seconds between you saving a file and seeing the change in your app. That’s easily ten times less than the typical Build and Run cycle we’re used to in (Native development).”* (Elkin, 2016). This does not apply to Native code, as this will need to be recompiled. This feature alone could make an enormous impact on the speed of the application development.  Naturally, not all developers will be able to immediately become productive with React-Native, traditional Native developers with little experience with JavaScript or React may struggle. “*As native developers we found the learning curve of React Native pretty high, as it may take up to a month until one could start feeling comfortable developing with this framework.”* ("React Native: Is It The End Of Native Development?", 2016). This may entirely negate the benefits of rapid prototyping and Hot-Reloading, having to potentially delay development for a month or more has the potential to incur serious costs. This however, is not true for all situations, Developers that are familiar with JavaScript, and more importantly familiar with the React framework will take to React-Native far quicker that purely Native developers.  Essentially, my research will focus on single platform development, specifically the Android platform. I aim to provide an analysis into the time taken to develop different parts of a mobile-application. Depending on the results I will offer reasons as to why there was or was not a difference in development time. “Assuming equal skill level on the part of the developer, it shouldn’t take longer to build an app on one platform or the other,” (Chernov, 2013), said  Joe Chernov, VP of marketing on behalf of Kinvey’s engineering team. Perhaps React-Native has some particularly obtuse API’s that hinder development time for particular areas of development. Perhaps the only reason React-Native has faster development time is because of Hot-Reloading. These are the types of questions that are missing from the current information available.  **Cost**  The cost of developing mobile applications is very high, many businesses will be looking to minimise the cost of creating an application whilst still sacrificing as little as possible in other areas. Whilst I have separated cost into a separate category, it is strongly tied to the amount of time it takes to develop an application, and so for the sake of clarity, some points from the previous section may be repeated.  Bernard Kohan of Comentum created an estimated ball-park cost of development for creating an MVP application (Minimum Viable Product) for both native and hybrid applications. He estimated that the cost of creating a native MVP mobile application for just one platform to be approximately one-hundred and fifty-two thousand dollars ($152,000), and creating the same native application for both iOS and Android platforms would cost two-hundred and fifty-one thousand dollars ($251,000). He also estimated the development cost for a hybrid application, i.e. An application that can be simultaneously developed on both the iOS platform and Android platform, to be one-hundred and sixty-two thousand dollars ($162,500). Either way, this is a substantial investment, Kohan goes on to state that: “If the developer needs to build the app for all platforms, and the app needs to be built using the native programming language of each platform, the cost of development will be doubled or tripled because of the amount of time that is needed to build the app using different programming languages of each platform.” (Kohan). The premise that hybrid development is almost half the price of native development is backed up by Keerti, a developer at Walmart Labs, who created the Walmart application in React-Native, he states that; “The defining feature of React Native, and arguably its best selling point, is that it’s cross platform — allowing for simultaneous development on iOS and Android by the same team, which can cut labor costs roughly in half.” (Keerti). With the price of Native application development being so high, it is natural that Hybrid development would seem the more attractive choice. Others, such as [NAME HERE] would dispute this claim, stating that: “*As native developers we found the learning curve of React Native pretty high, as it may take up to a month until one could start feeling comfortable developing with this framework.”* ("React Native: Is It The End Of Native Development?", 2016). Whilst this is not a direct opposition to the previous argument, I do feel that this is a relevant piece of information to consider. If it takes up to a month and possibly more for a native developer to familiarize themselves with the framework then this may effectively negate any improvements in development time, and thus an implicit reduction in cost. However this is a fairly subjective claim, even though the React-Native framework is relatively new, there are those who have been using the Web framework React for quite some time, and they would take far quicker to acclimatize themselves to using React-Native.  Ultimately, cost is a category that I will only be able to estimate by comparing the amount of time It will take for me to complete the development of both applications and average developer salaries, so it is more than likely that my research will not yield any results that are different from the sources above. However, I may be able to ascertain which parts of the application development would cost the most, and produce a relationship between cost, development-time and performance, which I believe would be a valuable new piece of information.  **Stability**  The stability of an application has the ability to significantly impact the cost and development-time of an application, so it must be closely monitored. In the article “Why I’m not a React-Native developer” by Ariel Elkin, she states that: *“JavaScript lacks these safeguards against programmer error, making preventable runtime crashes and preventable programmer errors part of your routine.”* (Elkin, 2016). Implying that the use JavaScript is a disadvantage and will adversely affect the stability of an application. Mostly this is a reference to the absence of type checking in JavaScript, and the ability to send data over a bridge from JavaScript to Native code that has not been type-checked, which will in most cases cause a fatal runtime error. These “Preventable programmer errors” that Elkin suggests may however be negated by the decreased development time in React-Native. Earlier in the article Elkin also stated that: “*React Native’s feedback loop is bewitchingly low. It takes less than one or two seconds between you saving a file and seeing the change in your app. That’s easily ten times less than the typical Build and Run cycle we’re used to in Xcode.”* (Elkin, 2016). Do these two points contradict each other? Is an error produced when compiling an application different to when an error is found when testing an application? Whilst the compiled application may catch the error quicker (Only if the time it takes to compile the application is low), it may only be relevant if those bugs are deployed into the production application. it seems that React-Native may be more vulnerable to developer error than native applications, and thus React-Native has a greater dependence on Unit-Testing and User-testing than Native development.  React-Native is a relatively new Framework, made public in 2015, but it still does not have an official release as the current version is 0.42 and it may not get a stable 1.0 release for some time. However, being an open source framework, when bugs are found, they are rapidly fixed and integrated. “*While it is good that they bring more features and push the framework towards maturity, often they bring breaking changes as well*” ("React Native: Is It The End Of Native Development?", 2016). Whilst new features and bugs are rapidly integrated into the framework, there’s also the possibility of producing breaking changes, i.e. Major changes in the API that will cause previous code to break. This may be viewed as a positive light, when assuming that breaking changes are only introduced when they are a vast improvement, however, depending on an applications reliance on these features, an equally large amount of time may have to spent on refactoring and redesigning code to be able to take advantage of these latest features, which in-turn would increase the costs of supporting the application. This quote from the same article validates this assumption: “*These changes sometimes where a burden to overcome, as we had people spending a lot of time fixing things when upgrading the framework.*” ("React Native: Is It The End Of Native Development?", 2016).  You may have noticed that all my sources have come from either, independent blogs, or from a company blog. This is because there are no existing scientific reviews, or articles covering React-Native, there were however a plethora of blogs and web-based articles. Many of these I had to exclude as sources entirely because of their sensationalist or inaccurate nature. I also chose to exclude any sources that were over one year old, this is because of the quick release schedule of react-native, the version of React-Native that existed a year ago, is entirely different from the one available today, and thus most of the information in the sources is similarly out-dated.  In summary, there is a large amount of informal data surrounding the advantages and disadvantages of using React-Native, apart from cost however, there is no statistical data available to create an accurate depiction of the differences between Native and non-Native development. My research aims to produce this statistical data. |

**2.3 Bibliography (key texts for your literature review)**

**Help:** Please provide references, in correct Harvard style, for at least three key texts that have informed your literature review. If you are implementing an application, select texts which demonstrate how other researchers have tackled similar implementations? The references should be recent and sufficiently technical or academic. Your markers will be looking for you to identify technical reports, conference papers, journal papers, and recent text books. Avoid *Wikipedia* entries, newspaper reports that do not cite sources, and general or introductory texts.

|  |
| --- |
| * Elkin, Ariel. (2016). *Why I'm Not A React Native Developer* - *Arielelkin.github.io [online]* . Available at: <https://arielelkin.github.io/articles/why-im-not-a-react-native-developer.html> /[Accessed 1 Mar. 2017]. * Novoda . 2017 *React Native: Is It The End Of Native Development? – Novoda.com*. [online] . Available at: [https://www.novoda.com/blog/is-it-the-end-of-native-development /](https://www.novoda.com/blog/is-it-the-end-of-native-development%20/) [Accessed 2 Mar. 2017]. * Kol, Tal. 2016. *Performance Limitations Of React Native And How To Overcome Them*. *Medium*.com. [online] Available at: <https://medium.com/@talkol/performance-limitations-of-react-native-and-how-to-overcome-them-947630d7f440#.ayr73y94x/> [Accessed 3 Mar. 2017.] * Panneer, Sriraman. 2016 *What We Learned After Using React Native For A Year - Hashnode*. *Hashnode.com*. [online] Available at: <https://hashnode.com/post/what-we-learned-after-using-react-native-for-a-year-civdr8zv6058l3853wqud7hqp/> [Accessed: 3 Mar. 2017] * Narusis, Vytenis. 2016. *Pros And Cons Of React Native - A Course Of Building Native Apps With Javascript*. - *Devbridge.com*. [online] Available at: <https://www.devbridge.com/articles/pros-cons-of-react-native-crash-course/> [Accessed at: 2 Mar. 2017.] * Keerti. 2016 *React Native At Walmartlabs – Walmartlabs*. *Medium*.com [online] Available at: <https://medium.com/walmartlabs/react-native-at-walmartlabs-cdd140589560#.e8esee7za> [Accessed 4 Mar. 2017.] * Kohan, Bernard.2015. *Mobile App, Iphone App Development Cost, Pricing*. *Comentum.com*. [online] Available at: <http://www.comentum.com/mobile-app-development-cost.html> [Accessed 4 Mar. 2017.] * Chen, Robin. 2016 *Using React-Native: One year later*, *Discord.com. [online] Available at:*<https://blog.discordapp.com/using-react-native-one-year-later-91fd5e949933#.sqsegx7nm> *[Accessed 1 Mar. 2017]* * Gomez, G. (2017). *React-Native: The Good, The Bad & The Ugly - 3 SIDED CUBE*. [online] 3 SIDED CUBE. Available at: <https://3sidedcube.com/blog/2016/04/react-native-the-good-the-bad-the-ugly/> [Accessed 8 Mar. 2017]. |

This is the end of section two.