Evaluation Targeting React Native in Comparison to Native Mobile Development

Oscar Axelsson Fredrik Carlström

DIVISION OF CERTEC | DEPARTMENT OF DESIGN SCIENCES
FACULTY OF ENGINEERING LTH | LUND UNIVERSITY
2016

MASTER THESIS IN INTERACTION DESIGN

Evaluation Targeting React Native in Comparison to Native Mobile Development

An assessment of Developer Impression, User Experience and System Performance

Oscar Axelsson and Fredrik Carlström

LUND
UNIVERSITY

Evaluation Targeting React Native in Comparison to Native Mobile Development

An assessment of Developer Impression, User Experience and System Performance

Published by

Department of Design Sciences
Faculty of Engineering LTH, Lund University
P.O. Box 118, SE-221 00 Lund, Sweden

Subject: Interaction Design (MAMM01)
Division: Certec
Supervisor: Kirsten Rassmus-Gröhn
Co-supervisor: Alexander Georgii-Hemming Cyon
Examiner: Charlotte Magnusson

Abstract

The app-industry today is ruled by two giants, namely Android and iOS. For companies and developers, it is essential to deliver their product to the majority of users, thus adapting to both platforms. The platforms involve their own way of developing applications and only barely resembles each other. A problem that the industry faces resides here, having to hire staff with knowledge in either Android or iOS, or both, to build two separate applications that in turn require parallel upkeep.

Cross-platform frameworks that could bridge this gap have come and gone, not succeeding in creating applications with the same visual or functional standard as the native frameworks deliver. Facebook has announced a new framework named React Native that promises to deliver a fully native experience with the use of only one code base. This thesis revolves around an evaluation of this framework, focusing on the development side, system performance and the user experience that React Native delivers.

We found that the users, when testing both a React Native application in comparison with native application separately, did not notice any difference, but when the applications were placed side by side all participants could distinguish the two systems. The overall quality of React Native encompasses more than development time and implementation, and we found a more situational recommendation to be adequate, thus addressing all underlying issues.

The end result shows that React Native applications are currently not as good as native applications regarding the user experience nor performance, but it demonstrates the capabilities of cross-platform reducing the existing gap between them. We anticipate our thesis to be a starting point for developers, deciding if they should select a native or cross-platform implementation.

Keywords

Android, Cross-Platform Development, iOS, React Native, System Performance, Testing, User Experience

Sammanfattning

Applikationsbranschen styrs idag av två giganter, nämligen Android och iOS. För företag och utvecklare är det essentiellt att kunna leverera sin produkt till majoriteten utav användarna, där av pressade att utveckla för båda plattformarna. Varje plattform har sitt eget sätt att utveckla applikationer och liknar bara vagt varandra. Problemet som industrin idag möts utav beror på detta, att behöva anställa kompetent personal med kunskap inom antingen Android eller iOS, eller båda, för att skapa två separata applikationer som behöver underhållas parallellt.

Mulitplatform-ramverk som kan minska detta avstånd har kommit och gått, men misslyckats med att skapa applikationer med samma visuella eller funktionella standard som native systemen kan leverera. Facebook har presenterat ett nytt ramverk, nämligen React Native, som lovar att leverera en komplett native upplevelse med användandet utav en gemensam kodbas. Denna masteruppsats kretsar kring en utvärdering utav detta system, med fokus på utveckling, prestanda och användarupplevelse.

Vi fann under våra tester att användarna inte upplevde någon skillnad i React Native applikationen i jämförelse med native, när de testades separat, men när applikationerna var placerade bredvid varandra kunde användarna med enkelhet urskilja skillnaderna. Den övergripande kvaliteten på React Native omfattar mer än utvecklingstid och implementation, därav finner vi en situationsbedömning vara mer passande, som även adresserar de underliggande problemen.

Slutresultatet visar att nuvarande React Native applikationer inte är lika bra gällande användarupplevelse eller prestanda, men det visar på möjligheterna för plattformsoberoende utveckling att minska avståndet till native. Vi förväntar oss att vår masteruppsats ska vara en utgångspunkt för utvecklare, i valet mellan native eller mulitplatforms-ramverk.

Nyckelord

Android, Användarupplevelse, iOS, Mulitplarforms-utveckling, Prestanda, React Native, Testning

Acknowledgements

We would like to thank Netlight Consulting AB for the opportunity to research this master thesis at their Stockholm office with support from the whole organization and employees. We would like to thank the participants in our survey, who have willingly shared their precious time during the process of interviewing. A special thanks do we target to our supervisor at Netlight Alexander Georgii-Hemming Cyon for all the valuable input and support during the whole project.

We would furthermore want to thank our supervisor at Lund University Kirsten Rassmus-Gröhn for the continuous support and advise.

——

I would foremost like to thank my family for all the support throughout this whole process, from my parents Bertil and Lotta, my brothers Christopher, Joachim and Mikael to my sisters Catthy and Karin. Whose love and understanding is nothing short of astounding and is what has fueled me these past six months, if not for my whole life.

I would secondly like to thank all my friends, whom grouped together in this crude manner are all individually a big part of me. Thank you all for the continued support, patience and encouragement during this time!

Finally, I would like to dedicate my involvement in this thesis to my twin brother Mikael, my biggest inspiration.

Fredrik Carlström

——

I want to thank my parents Inge Axelsson and Lena Gustavsson for their support throughout the years. Also, I would like to thank my family and all my friends for their patience and continuous encouragement. Last but not least do I want to thank my beloved girlfriend Helen Holmgren for all of the sacrifices that you made on my behalf and always staying supportive.

This accomplishment would not have been possible without you all, Thank you!

Oscar Axelsson

——

Stockholm, June 2016

Contents

Abbreviations and Acronyms

API - Application Programming Interface
CPU - Central Processing Unit
CSS - Cascading Style Sheets
DOM - Document Object Model
HTML - HyperText Markup Language
HTTP - Hypertext Transfer Protocol
HTTPS - Hypertext Transfer Protocol Secure
JSON - JavaScript Object Notation
OS - Operating System
POJO - Plain Old Java Object
RAM - Random Access Memory
REST API - Representational State Transfer Application Program Interface
RN - React Native
SEQ - Single Ease Question
SLOC - Source Lines of Code
SUS - System Usability Scale
UI - User Interface
UX - User Experience

Part I

Introduction

Chapter 1
Background

As seen in statistics from Developer Economics, 80% of companies still have development teams that are solely dedicated to one or the other platform [1].

Tools that allows the developer to use the same source code for two different platforms is nothing new. We've all heard the promise of cross-platform native apps driven by JavaScript or HTML5. Tools like Phone Gap, Appcelerator and Xamarin has been around for quite some time, but they often fall short in terms of performance and user experience [2], [3] and [4]. Either you're wrapping a web application in a web-view, or they are tied to HTML and CSS. The big problem is that the application, does not feel like an application, but a website. React Native, however, seems to have addressed these issues [5]. If React Native can provide a user experience in level with that of an application written in fully native code, it will most likely change the industry.

Many believe that the use of native tools is much more beneficial in building the best user experience. Native applications are faster, cleaner, and they behave and look just the way users expect them to do. The ability to utilize sensors (GPS and accelerometers), functions (Datepicker and Dialogbox) and design patterns, i.e. native implementations, are some examples of what makes a native application (just that) native.

The preconceived notion is that creating native applications is harder to learn and can also require much more time to master properly compared to the cross-platform mobile development. Additionally, native applications require a complete code rewrite with a different language before they can run on any other mobile platform.

To develop one application that runs on 96.7% (Figure 2.1) of the mobile devices (Android and iOS) you need two teams working simultaneously, or one team with twice as long development cycle, doing basically the same logic, but in different programming languages and environments.

Today many companies have both Android and iOS teams, where the team members knowledge is mostly limited to one platform. If all these companies could instead have one single team, being able to develop both Android and iOS apps, it would cut the development time and the costs dramatically.

This thesis strives to analyze the possibilities of cross-platform applications regarding React Native overtaking native development, reducing both the time and cost for many companies.

Our interest in this evolves from the industry's demands for improved solutions in the subject of cross-platform applications and our own interests in the subject. We both have several years of experience in app development and have found it difficult to cover all platforms without building two exact copies of the same application using two different programming languages. If React Native is the solution to these issues remains to be seen, but the effect this could have on the industry and future app development is what drives us to conduct this research.

The research is intended for other master thesis students with an interest in mobile development and user experience. It also targets developers or companies standing in front of the choice of creating a React Native or native application. It may also include anyone interested in the development of mobile applications and what the future may look like.

1.1 Problem Formulation

With React Native as the top emerging cross-platform solution, we aim to asses and evaluate its capabilities compared to native developed applications. We are approaching this problem of evaluation with little to no experience of the tool itself. With overwhelming positive response found from researching, the essential question we are striving to answer: How good is React Native in contrast to native development? We have below structured six questions that in more detail describes our objective.

1.1.1 Goal

We have reduced the problem of this assessment to six main questions:

- How well does React Native utilize the hardware sensors?
- How well does React Native handle interruptions that occur in everyday usage of applications?
- Is the performance of React Native applications on par with native?
- Is the user interface indistinguishable from native applications?
- Is the user experience of React Native on the same level as native applications?
- Is React Native an efficient tool for developing mobile applications?

1.1.2 Thesis Deliminators

During this research we only focus on the two major platforms on the market iOS and Android. There are other competitors, but collectively they only cover a small fraction of the market and due to limitations of time we have decided not take them into consideration, see more about the industry in Section 2.1. We have also decided not to focus on comparing other cross-platform frameworks such as Xamarin or PhoneGap. Mainly because the cross-platform frameworks are trying to conquer the native market and the interest is to find out if we could use React Native instead of a native development tool.

1.2 Thesis Outline

For the reader to get a better grasp of React Native and why this is of such interest, a more in depth look is described in Chapter 2 *Theory*: Section 2.1 *The Industry Today* and 2.2 *React Native*.

This is followed by an introduction to our approach and process for evaluating React Native in Chapter 3 *Methodology*. Section 3.2 *Testing* and 3.5 *Interview* are explanations of how the tests of the end product were structured and executed. Within the same chapter the development process' conception and structure is described in Section 3.3 *Survey* and 3.4 *Integration*.

Part II *Development* presents the whole development's evolution, split into to chapters, namely Chapter 4 *Logical Stories* and Chapter 5 *UX Stories*. Lastly the thesis ends with a presentation of results, discussion and conclusion in Part III *Result*.

1.3 Work Distribution

We have during the project worked side-by-side and divided the workload evenly between us. During the development phase did both work on React Native separately and later merged our implementations into one solution. Fredrik handled the native iOS development and the native Android was developed by Oscar. The remaining work such as the interviews, surveys, report and the presentation have we both been working on together.

Chapter 2
Theory

The purpose of this section is to introduce the reader to the fundamental theory, necessary to understand this thesis.

2.1 The Industry Today

The mobile industry is booming like never before, it is an exciting environment, but not without its issues. The industry wants to be able to build applications more efficient than what now is the standard. But at the same time you want the application to have the same feel as any other application on the market. The issue arises when you start to develop and realize that you have to set up several different environments and learn at least two different programming languages, Java and Objective-C/Swift.

One of the big giants on the market, Apple, had a revenue growing to 28% during 2015 with nearly $234 billion and the Q4 2015 they made a revenue of $51.5 billion and quarterly net profit of $11.1 billion [6]. These numbers are believed to increase during the start of 2016 thanks to the release of the next model iPhone 7. Despite that, the market for Apple could soon be saturated, caused by the growing Chinese market [7]. Apple has also received a greater interest from developers with their new programming language Swift [8], which is increasing despite that 16% of the developers working in it is not making any revenue, compared to 4% on Objective C side [1]. The big interest in Swift could be that it is making it easier for developers to create and learn iOS development compared to the standard Objective-C.

Despite the rise of Swift, only a mere one out of six developers have experience with it, according to VisionMobile, *"Developer Economics: State of the Developer Nation Q3 2015"*. This can be compared to Objective-C, which stands at 40% of developers.

All the success for Apple aside, currently the most popular language by developers is Java, with more than 65% of all developers having experience with it. This is something

that favors Android, since Java is the language used for development, and with a market share for 2015 Q2 of 82.8% it is not surprising, compared to 13.9% for Apple [9].

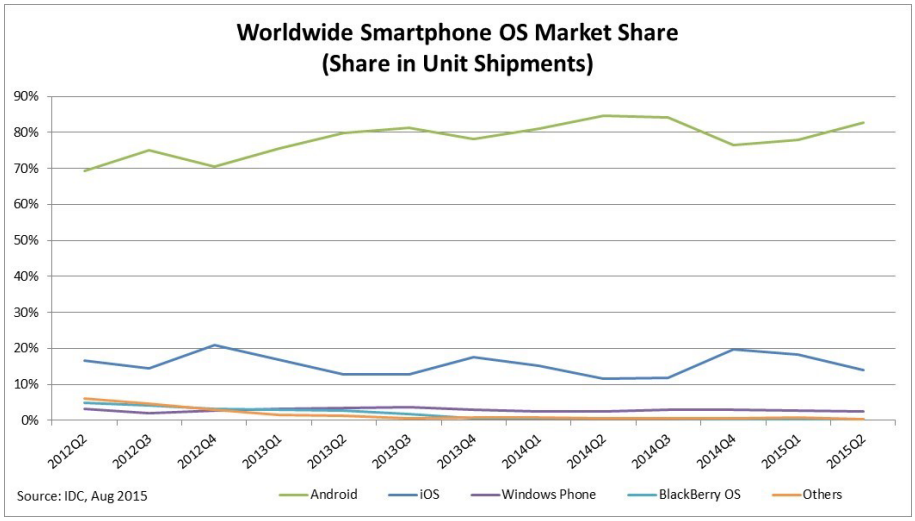


Figure 2.1: The Worldwide Smart phone OS Market Share from 2012 Q2 to 2015 Q2

The two giants on the mobile market owns 96.7% of the market which is remarkable. The competitors on the market are having trouble to break through and there is no sign of the market changing in the near future. The market share can be viewed in Figure 2.1

As a developer with a good idea for an application and a goal of reaching the vast market there is no question that it must cover both Android and iOS platforms. The average game developers builds apps for 2.6 platforms and for non-games developers it is 2.2 platforms [1]. Currently 37% of all mobile developers target both iOS and Android, but it is expected that their numbers will grow in the next few years. This means that 37% of the mobile developers has knowledge and experience in both platforms. The remaining part has experience only from one of the other platforms. Starting a new project at a company you will probably be forced to at least set up two teams, one for Android and one for iOS and create the same application twice. This is not an efficient way since it will cost both money and take time. The question then arises if we can't *"learn once, write anywhere"*.

2.2 React

The story begins with React which is a JavaScript library, built by Facebook, with the first release in 2013 [10]. React is a framework that focuses on breaking down the applications components to represent a single view. The components facilitate the possibility to only re-build the part of the application that was changed. This creates a faster and easier development cycle that facilitates for changes. React creates a lightweight Virtual DOM (Document Object Model) every time components are rendered, and diffs them to figure out the smallest amount of real DOM changes needed. With this new system Facebook has been able to quickly build and test the system, making the agile iteration process faster and

the time to market shorter. Another big change React creates is that it wraps the DOM's mutative, imperative API (Application Program Interface) with a declarative one, which raises the level of abstraction and simplifies the programming model.

An imperative style focuses on how the program operates, but a declarative style focuses on what the program should accomplish. React is able to, with a JavaScript object, declarative render a structure and then let the library figure out how to reify it into DOM objects. With a declarative approach it makes the code more predictable because the developer knows how its going to behave. This makes the developer confident to make changes because they know how it works. According to Facebook and Tom Occhino (Engineering Manager at Facebook and one of the creators of React Native), as he stated on React.js Conf 2015 Keynote, new developers that never before have worked with JavaScript can in the end of their first day make changes to the code. This has never been experienced before at Facebook [11]. With this, Facebook made their code more predictable and as a result can iterate more quickly, thus creating applications that are more reliable and does not break.

But if the web is a lot better by using React, why don't we use it on mobile as well?

A native application results in a better application thanks to all the platform-specific UI components that are available, such as a date picker and dialog box as well as asynchronous image decoding, text measurement and rendering. With a native application the developer also gets access to a sophisticated threading model that can take the design compilations of the UI thread, which makes animations and image rendering flow seamlessly.

But mobile development also has issues because we have two (if we only mention the dominant operating systems) disjointed platforms.

There also exist a large issue with native applications, that they don't compile fast enough, taking approximately 30 seconds for a new change, making the process and development velocity slow. The reason engineers are willing to go through the pain of developing to native is because they can create a user experience that is a lot better than any website (at the moment) can create. As Tom Occhino from Facebook said on F8 2015 *"We build native apps because we can create better, more consistent experiences"* [12].

So what we want to achieve is the native experience for mobile with the developer experience of the web. This can be achieved through several different ways, one of the more popular is:

WebView It is a webpage that can be displayed using WebView inside a simple native wrapper application. It is a good idea, but unfortunately it does not provide the native user experience. The WebView also lacks the ability to parallelize work on different threads and sophisticated gesture handling which can be created in an native environment. A WebView is therefore not a good replacement for native development.

2.3 React Native

The solution to this might be the newly presented framework React Native, developed by Facebook, first released in 2015, and is now and open source project on GitHub [13]. With React Native we can use the best from React (such as fast re-builds and an declarative

DOM) and be able to run an instance of the JavaScriptCore inside the application and by that create platform specific components [14]. The application logic is written and runs in JavaScript, whereas the application UI is fully native. With React Native we can use both JavaScript and native implementation, making the switch to a cross-platform development smaller for existing projects. We can also, if a functionality does not exist in React Native framework, use native code, so the user gets the best experience possible.

React Native is not aiming to be a *"write once, run anywhere"* tool. It is aiming to be *"learn once, write anywhere"*. Android and iOS are two different systems and should be treated differently, therefore not be designed in the same way. There are differences in components and special native features which makes the experience better and unique. By learning React Native's syntax and structure will the developer be able to compile targeting different platforms without the need to learn a new set of techniques. With the same code and technologies a developer will be able to deploy an application to both Android and iOS.

Chapter 3
Methodology

In this chapter we present the methodologies and approach used in our thesis to answer our previously stated problem formulation.

The approach that we chose can be divided into two segments, illustrated with Figure 3.1 and Figure 3.2 below. With the first segment being the groundwork for, and including, the development of the applications.

Figure 3.1: First Segment

Figure 3.2: Second Segment

The second segment consist of final testing and comparing the finished products in terms of user experience, efficiency and performance, see Section 3.2. Then after both segments were completed we gathered all data for analysis for the conclusion and discussion, Chapter 10 and 11. All these methods will be introduced in a more precise manner further down.

3.1 Research

Before starting with the project we researched everything involved. From ways of structuring surveys, conducting interviews to methods of testing the developed products. Everything covered in this section is thoroughly researched using literature provided to us by our supervisor and also the information that is available online. Furthermore, the application scope and features were all investigated, making sure it was possible to implement everything in all environments.

3.2 Testing

To carry out our comparison of React Native with native applications we did extensive tests. We divided it in four main approaches: **User Experience**, **Reliability**, **Interaction with sensors** and **Benchmark**. With these four approaches we are confident that we are covering all important areas that concern app development. The first part covers the user experience with the mobile device, the second part focuses on the reliability and the hardware utilization and the third with interruptions that occur during usage. These three cover everything surrounding the users interaction and lastly, benchmark addresses the specifics when it comes to the applications usage of the device resources.

3.2.1 Real User Experience

Confusion can often arise when mentioning the two terms **user experience** and **usability**. According to the International Organization for Standardization (ISO), user experience can be described as *"person's perception and responses resulting from the use and/or anticipated use of a product, system or service"* [15]. Usability, found in the same standard (ISO 9241-210: "Ergonomics of human-system interaction"), is described as *"extent to which a system, product or service can be used by specified goals with effectiveness, efficiency and satisfaction in a specified context of use"*.

In the book "Measuring User Experience" by Tom Tullis and Bill Albert, they distinguish the two terms with the simple statement *"Usability is usually considered the ability of the user to use the thing to carry out a task successfully, whereas user experience takes a broader view, looking at the individual's entire interaction with the thing, as well as the thoughts, feelings, and perceptions that result from that interaction."* [16]. UX (User Experience) is something all developers are concerned with. If measured, it can help improve the product and the impression the user gets from it. When looking at applications that are deployed to mobile devices, UX is a big concern. The operating system themselves play a major part in this, design patterns and interaction between the user and the application are different depending on the OS (Operating System) and can even differ between the updates within the same OS [17] and [18].

It is clear that user experience will affect further usage of an application, especially when there may exist a plethora of other applications that fulfill the same purpose. Thus, with a cross-platform tool such as React Native, utilizing the nature of the native design can be crucial for success (this is not stating that UX is the only factor, but simply that its role should not be minimized). When researching the developing public's opinion on

the matter of React Natives capabilities of native design implementation, ignoring development issues and focusing more on abilities, we found promising results [19], [20] and [21].

It may not be clear though how to measure UX, as it mainly deals with something subjective. The broad consensus is using **surveys** and **interviews**, but how you design these questionnaire is not a walk in the park. Our supervisor recommended the aforementioned "Measuring User Experience" by Tom Tullis and Bill Albert that deals with exactly this, the science behind measurements of UX.

The questionnaire is designed in a way that will avoid the "acquiescence bias", meaning that people are more likely to agree with a statement than disagree with it. A proposed solution to this problem is to balance a positive-phrased statement with a negative one. It is also known that people tend to rate the experience highly even though they have failed to complete the task. There has been studies which have concluded that subjects are reluctant to criticize a design or the technology because it might give an impression that they are negative people or that they are not good at adapting to computer-based technology. A person often rates his/her skills higher than what the actual test shows they are, mostly because they don't want to seem inexperienced or show a lack of knowledge.

With this we could confidently compare the user experience with the application developed in the two different environments, which we will discuss more in Section 3.3 and 3.5.

3.2.2 Reliability

In ISO/IEC/IEEE 24765: "Systems and software engineering", reliability is described in a two part manner: *"the ability of a system or component to perform its required functions under stated conditions for a specified period of time. 2. capability of the software product to maintain a specified level of performance when used under specified conditions"* [22].

For React Native we wanted to consider the reliability of the developed applications. What this means is testing how the applications handles various interruptions that occur in everyday application usage. When researching React Native we almost exclusively found discussions about the user experience, but we believed that reliability testing is just as important. If the application were to crash, when receiving a SMS for example, it would also affect the overall quality. For this purpose it was important to test the end products tolerance for interruptions with a list of common everyday disturbance factors for applications.

In the end this was compiled into a checklist which we could asses manually and below are the main tests we focused on [23]:

- Incoming and Outgoing SMS and MMS
- Incoming and Outgoing calls
- Incoming Notifications
- Cable Insertion and Removal for data connection
- Network outage and recovery
- Device Energy saving mode

3.2.3 Benchmark

Even if it is closely related to UX, a big variable that needs to be examined by itself is performance. It is described in ISO/IEC/IEEEE 24765 as *"the degree to which a system or component accomplishes its designated functions within given constraints, such as speed, accuracy, or memory usage"* [22].

While the time to retrieve information from a database is evident to the user, memory issues might not be as noticeable. When evaluating a framework, performance is a prominent property that requires its own testing.

This is called benchmark, comparing performance metrics between two sources. From Androids own "Performance Profiling Tools" page they state: "Putting pixels on the screen involves four primary pieces of hardware" [24]. This gave us a natural way of selecting how to measure the applications capabilities and divide the benchmarks into five main parts:

CPU Monitor the processing power and behavior

Memory Monitor the memory management

Battery Measuring the energy consumption

Network Noting the network performance

Application Size The application size on the mobile device

Our plan was to compare the results of these tests of the different applications, but on the same mobile to get a more accurate result. The legitimacy of benchmark testing and the results have been greatly discussed, more specifically when commercial products advertise their specifications [25]. Because systems and the conditions might effect the data from benchmark testing, the result can be a good base, but not an accurate depiction of how behavior will be at the hands of the everyday user. Since our result are only based on comparison, empirical data that we gathered with the same environment and conditions, we found benchmark to be a sufficient tool for our analysis. The conditions were as similar as possible with deployment on the same device for the specific platform minimizing the differences. The tests were conducted at the end of the development of both the Logical and UX Stories. For an even more accurate result were the tests performed several times with a calculated average result.

We also took advantage of the systems own tools for performance analysis and main focus was aimed at results within the same system [24] and [26].

3.2.4 Interaction with Sensors

Mobile devices come packed with hardware that, when used, make mobile applications more unique than websites. The interaction with these components is something that has been lacking in some older tools that have attempted to do the same thing as React Native. With a survey we attempted to pinpoint the most valuable perceived hardware interactions that users regard, and incorporated these features in our application to compare accessibility with the native application. With these important sensors in mind we checked whether

React Native had access to them and if they could be implemented. The results from the survey, along with the features most participants preferred with a mobile device, can be viewed in Chapter 6.

3.3 Survey

We used four surveys for our research, where three were made with Google Forms [27]. Google Forms allows for online distribution and feedback in a simple yet effective manner, easy to construct and are the reason we chose it. The first survey (referred to as *Online Survey*) served as a foundation on which we identified native components and features that we focused on when developing the applications, which we present further in Section 3.3.1. The other three are used in our final assessment for how React Native's user experience stands in contrast to native implementation, described in Section 3.5.

3.3.1 Online Survey

For our foundation we asked three questions to find out what users think of applications versus websites in general, specific functions that they enjoy and also asked them to list their favorite applications. This was sent out as an online survey and can be seen in Appendix B. The survey was created more specifically to narrow the scope of our implementation, since an implementation of all mobile hardware functions would require significantly more time to integrate in our applications.

With our first question we wanted to identify the users preferred mobile device, i.e. Android, iOS, Windows Phone or other. This distinction was made with question four in mind, where the participant is asked to list their favorite application, for a better understanding of which applications users appreciate. Our second question is an attempt to identify the participant's views on the subject of applications versus websites, to aid us in our development as guidelines, getting a better grasp of perceived native behavior. The third question asks the participant to be more precise in their reasoning behind native behavior, where specific functions listed in the result could be integrated in our application.

The remaining three surveys were used in a more comprehensive UX interview. To structure this we followed the pattern suggested in the book "Measuring User Experience" by Tom Tullis and Bill Albert [16]:

- *What type of participants do we need?*
- *How many participants do we need?*
- *Are we going to compare the data from a single group of participants or from several different groups?*
- *Do we need to counterbalance (adjust for) the order of tasks?*

Within these questions there are a series of questions that, in turn, require an answer. By answering these questions we had thorough understanding of what was needed for the interviews. This can be seen below, where the list of questions are divided into subsections.

3.3.2 Type of participant

First of all comes the question about
"How well the participants will reflect the target audience?"
For a survey you would want perfect reflection, but this may not be achievable and then a more approximate reflection could be sufficient.
"Is the data going to be divided by the type of participants?"
If so, it would require additional thought as to the quantity of each group. The last refers to
"Sampling Strategy".
The strategies are all different ways of reaching a more generalized conclusion about the overall user experience. The strategy is to find a way of representing the general population. For this you can use different strategies: **Random sampling**, **Systematic sampling**, **Stratified sampling** and **Samples of convenience**. Since we wanted to get as much feedback as possible, without any restrictions, we settled for a sample of convenience.

3.3.3 Number of participants

When it comes to sampling size Tom Albert and Bill Tullis state that there is no bottom factor requirement of size, but merely *"the goals of your study and your tolerance for a margin of error"*. It depends on the objective at hand, with a small sample, issues might be addressed, but compared to the relative task smaller problems may still persist. But if major problems are the only objective then a small sample size might be as efficient as a large. The other factor, tolerance for margin error, will also contribute to deciding the sample size. Confidence intervals can then be established based on the sample size and success rate, which will in turn give a more accurate reflection.

3.3.4 Comparing data

There are two choices to be made here, either to use Within-Subjects or a Between-Subjects Study. Within-Subjects refers to comparing different data for each participant (analyzing individual parts of the design) and Between-Subjects means looking at the data for each participant to other participant (analyzing collective data for different groups).

3.3.5 Counterbalancing

Lastly the need for task order might be of concern:
"Will something be gained from participants performing the task given in a certain order?"
Knowledge gained from performing a specific succession of tasks might saturate the result and can be prevented by counterbalancing. If the order might effect the result then changing the order for participants would be a solution.

3.4 Integration

Since we chose to develop applications in both React Native and native platforms it provides a more comprehensive base for measuring the differences as the applications were deployed on equal mobile devices. This also provided us with a thorough insight into the development process of React Native. With many parameters to consider, we proposed a story based iteration process, where after each completed story we could reflect and compare the different environments with comments on complexity, time and source lines of code as our main metrics, as these are good indicators of quality developing.

3.4.1 The Application

The application itself is based on the popular game "Memory". It was suggested by our supervisor at Netlight, to provide the company with an easy and fun way for their employees to put names to all the faces of their fellow colleagues. For this we were granted access to Netlight's internal REST API (Representational State Transfer Application Program Interface) that they have up and running. The application incorporated all the features listed in Section 3.4.2. With a fully functioning REST API we spent minimal time on the back-end functionality and our focus was on the main objective, front-end development.

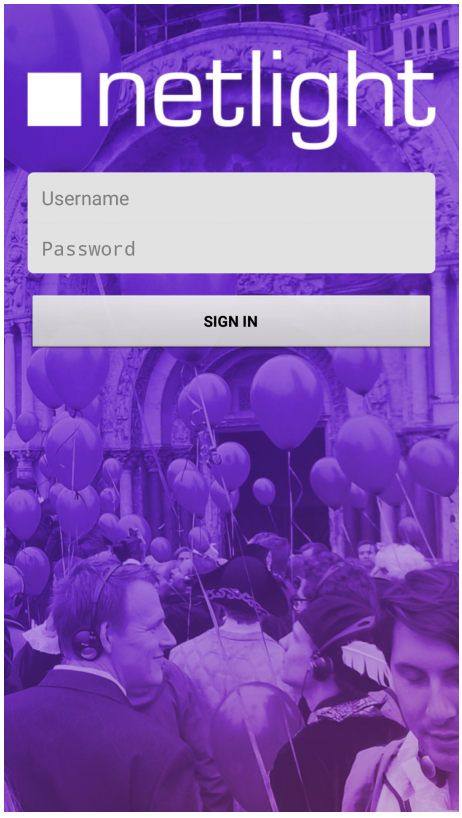


Figure 3.3: A mock-up of the login screen for our Memory Application

The authentication process is a login and start page, a mock-up can be seen in Fig-

ure 3.3, where Netlight credentials were needed, but is provided for every employee at Netlight, including us. A token is then generated by the server when authentication is accepted. With this token could we, through a call to the API, retrieve a list of all the employees at Netlight, with their respective profile information and profile picture. The information is then stored on the mobile device's internal memory.

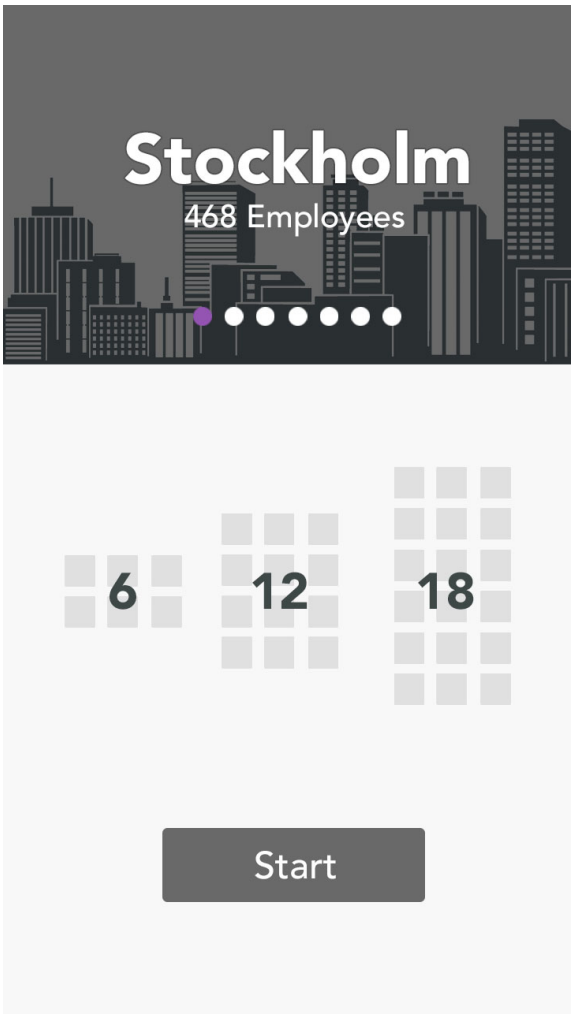


Figure 3.4: A mock-up of the setup screen for our Memory Application

The user will after a successful login be prompted with a setup screen, a mock-up can be seen in Figure 3.4. Here geolocation is utilized and with it the user will, depending on his/her position, be prompted with the closest city from which the employees in the game will be selected. This is because you often want to learn the names of colleagues at the same office, but if the user already knows all the employees at one office, it is possible to change the city, by swiping between the different offices. The user will also be able to select a different degree of difficulty. The setup process is then completed and the game can begin by clicking the start button.

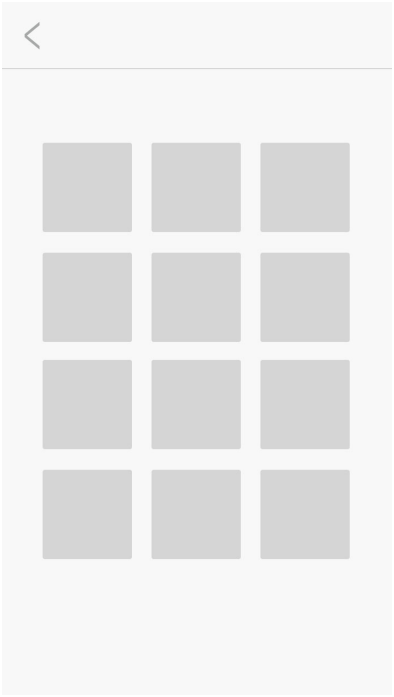


Figure 3.5: A mock-up of the game screen for our Memory Application

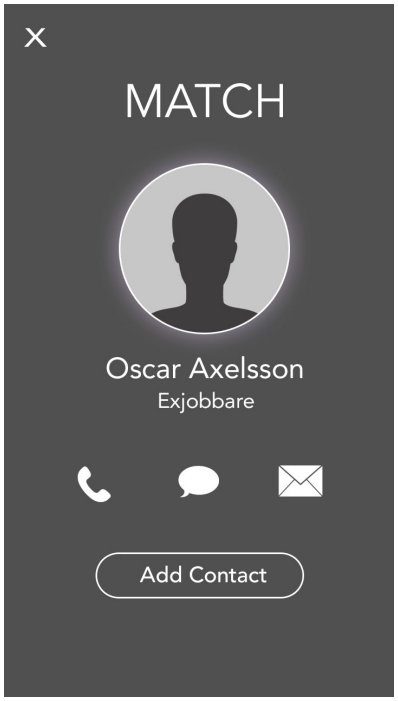


Figure 3.6: A mock-up of the match screen for our Memory Application

The game is modeled after the popular game "Memory", layout seen in Figure 3.5. The objective of the game is to match pairs of cards, that are shuffled and placed face down. These cards contain profile pictures of Netlight's employees. When a match occurs a screen will be presented, mock-up seen in Figure 3.6. This screen will show the matched employee's profile information, alongside actions that enable the player to call, sms, mail or add the employee to their contacts. The memory game will persist until all cards are matched. When the game ends the user will be prompted with another screen and then, with the touch of a button, be sent back to the setup screen.

3.4.2 Development Process

When choosing which development methodology that would suit this project the best, we concluded that we needed short cycles, so the process and results could quickly be analyzed in a continuous delivery. With only two developers, communication was easy and efficient, therefore we settled on an agile development process. In short, agile is a development process that corresponds to our needs, an iterative incremental way of development that opens up for quick changes during the iterations [28]. With this process we were able to implement and test continuously for each new implementation of a feature, which will further on be referred to as a *Story*.

Logical Stories

For our application we identified six initial stories, that were complemented by the first survey regarding interaction with sensors, see Chapter 6. From a discussion with our supervisor at Netlight and a couple of meetings internally were we able to determine six distinct features that our application required. These stories also served as milestones for the progress of our development phase.

Story 1 - Login Contains the task of connecting to Netlights REST API via a HTTPS connection. The user is authenticated with corresponding username and password. If the network is available and a correctly formatted input is provided then a HTTPS POST request is sent to the API. A response code is then returned by the server. If the response is OK (200) the user will be logged in and a token is provided used for accessing employee data. An error message is provided if the user is Unauthorized (401) or Forbidden (403).

Story 2 - Information retrieval It contains the task of retrieving information from a remote server. Using the token from previous Story to download the all the employee information from Netlight.

Story 3 - Internal storage During this Story we store the information retrieved in Story 2 to the mobile devices' internal memory. For security reasons we are not storing any sensitive information such as the token received from the API or the password used to sign in. Only a list of all employees containing public information such as email address and name are stored. This is done for improved user experience, eliminating the task of login for the user.

Story 4 - Interaction with sensors This Story includes implementation of hardware functions such as GPS and accelerometer. We select the closest Netlight office, proximity wise, based on coordinates of the user. We also implement a way of recognizing a shake gesture based on the accelerometer by using a threshold value based on the x,y and z-values in the accelerometer.

Story 5 - Animation Utilizing animations within the application. The animation itself is a grid of boxes which flip themselves when touched. If two boxes are flipped in a consecutive order and they contain the same value then they will stay in this flipped state for the remainder of the game; if the values differ then they will both simultaneously flip back to the initial state.

Story 6 - Other applications Capabilities of launching other applications. When a match occurs in the memory game the profile information along with the ability to call, sms, email or add the contact information is shown. So for this story we tested the ability to launch these applications from the application itself.

After the completion of each Story we marked down the comments, time and source lines of code it required and also tested the stories with benchmark and reliability, described above in Section 3.2.2 and 3.2.3.

UX Stories

The next phase of our implementation included improving the design and user experience in the application. Together with our supervisor at Netlight a first draft was designed. We are following the structure and development methodology from the Logical Stories because it was found successful and worked for our purpose. The stories for this phase are for simplicity divided by the five different screens in the application.

Story 1 - SetupView The first Story concerns the setup screen as can be seen in Figure 3.4. This screen includes creating a board size selection with a board size of 6, 12 or 18 and an office selection containing the name of the office with an image from the corresponding city including the number of employees. It also contains the ability to swipe between the offices. There was also some structure implementation needed to be improved for the ability to sort employees for the offices and select a random number of employees that will be included in the created game.

Story 2 - ShakeView The profile images for the employees selected to be a part of the game are downloaded in the background. During this will the user be presented with clear instructions to perform a shake motion. If the ongoing download of profile images are not completed will a progress bar be presented asking the user to wait.

Story 3 - GameView This Story mainly concerns the design of the cards in the board, further more using the usernames of the flipped cards to check whether a match has occured. When a match has taken place the employee information is packaged and then sent to the following screen.

Story 4 - MatchView The MatchView is an transparent overlay of GameView and will present the employee that has been matched in the game. The view contains the employee's name, level and profile image. The user also has the option to call, text message, mail or add the person to his/her private phone book.

Story 5 - FinishView To finish the game we added an additional screen, composed of static text messages and a return button. When the button is clicked the user will finish the game and return to the SetupView, where a new game can be initiated.

3.5 Interviews

In the same sense as surveys, interviews can be constructed and conducted in several different ways. Our approach was selected with guidance from our supervisor and a semi-structured interview was deemed the best method for our comparative user study. We used three surveys as our basis for the interviews: **Single Ease Question**, **System Usability Scale** and a **Wordlist**, which are explained further down in this section. Single Ease Question helped us score the individual implemented components while System Usability Scale served as an overall indicator. The Wordlist helped us get even more feedback on what the perceived differences were. With the comparative nature of our thesis we found that these tools would provide concrete results to stand on when interpreting the overall thoughts and perceptions of the applications compared to each other. This was also complemented by using a video capturing tool called **Lookback** to record the user expressions

and voice, described in Section 3.5.4.

When entering into the interviews we were aware of the mindset needed for conducting a successful interview. For this we used an article written by Isabelle Peyrichoux, *"When Observing Users Is Not Enough"* [29]. With psychology at its basis, taking inspiration from psychology giants like Carl Jung and Carl Roger, it delves into personality types and structuring questions for honest answers and also getting the most from the participants. Most notable points are being genuine and honest yourself, adapting the pace and being aware of the participant involved. It can be viewed as a guidebook and was used as our foundation when conducting the interviews.

3.5.1 Single Ease Question

Single Ease Question (SEQ) is, in its entirety, very simple [30]. After each completed task the participant is asked the question *"Overall, how difficult or easy was the task to complete?"* and the reply is given with a number between 1 and 7 (with 1 being **Very Difficult** and 7 being **Very Easy**). The survey is presented in Appendix C.

Although incomplex, the results are shown to work as well, or even better, than similar tests, such as Subjective Mental Effort Questionnaire or the ratio scaled Usability Magnitude Estimation [31].

The average score has been calculated over extensive studies, more specifically 200 tasks with 5000 participants, and an average score of 5 has been found. Therefore it is encouraged to question the participant's reasoning behind a score of 5 or less to get a better understanding of the shortcomings regarding the system.

3.5.2 System Usability Scale

SUS (System Usability Scale) is a widely used technology independent usability test and the results are shown to be equal to, or even more reliable, than other commercial evaluations [32]. With a standardized way of scoring participants' answers, SUS can be relied upon to deliver consistent results, even with a smaller sample size. It is important to state that the test was not constructed to diagnose and delve into the actual usability issues, but grade the overall usability of the intended system. The test itself is administered through a questionnaire where the participant's reply range from **Strongly Disagree - 1** all the way to **Strongly Agree - 5**. The questionnaire can be viewed in Appendix D and is structured as such:

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.

6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

The answers are then converted through a set of specific rules:

- For odd items: Subtract one from the user response.
- For even items: Take the value five and subtract the user response.
- Finally add all response for a user and multiply it by 2.5. This new converted range of possible values are now from 0 to 100 instead.

With data collected from 500 studies and 5000 individual replies, a score of 68 (deviation of 12,5) is found to be a sufficient threshold when determining if a system is usable. By normalizing and dividing the score into percentile groups the system in question can also be given a grade that ranges from F to A.

3.5.3 Wordlist

To measure the users satisfaction are we using a questionnaire as a part of the usability test. We are using a generated and randomized word list created by Dr. David Travis [33]. The subject is asked to read through all the words and select all words that best describe the experience with the application. From the selected words are five adjectives highlighted that later will become the basis of the post-test interview questions where the selected words can be explained and discussed more. The metrics received from the questionnaires are presented in a word cloud where the font size of each word is directly proportional to the number of times a word was selected by participants. The words presented are only from the five highlighted words selected by the user. From this can the opinion and reflection from the participants easily be viewed, showing what users perception of the application was. The survey used is presented in Appendix E.

3.5.4 Lookback

For our interviews we received a recommendation from our supervisor of a tool named Lookback [34]. Lookback is a UX research tool for building better mobile applications and websites. Lookback lets you capture the mobile screen and users face, via the front facing camera, voice and all the touches/clicks on the device. This let us not only see their touches and interaction, but also their emotions and expression during the task. We also combined this with an additional camera in the periphery to get a more comprehensive view of the interview. This camera recorded the entire session including the oral questionnaires which creates an ability for us to replay the interview and not be committed to writing everything down.

3.5.5 Structure

All interviews were conducted in a equal structured fashion. The participant was first asked to choose the platform that they felt the most accustomed to (iOS or Android) and was then given two mobile phones of the same brand and model based on this choice. One mobile phone contained the React Native application and the other the native application. The starting phone was selected through a predetermined order with Lookback installed, where the order was simply switched for every new participant. For every application the user tested Lookback recorded the whole session. For our interview process we constructed a use case that would force the user to interact with most of the application in a time effective manner. The use case was structured as following:

- Select the Munich office
- Select a board size of 12
- Shake to continue
- Play the game
- When a match occurs, choose Mail and then navigate back to the application
- Play until end and return to the setup screen

For each of these tasks the participant was orally asked a Single Ease Question, rating the difficulty of the task performed 1 to 7. After the use case was completed the participant was handed the System Usability Scale and asked to fill it out. Subsequently the Wordlist was presented and asked to be filled out. All this was then repeated for the second mobile phone with the other application, executed in the exact same order without the participants knowing which platform they were testing.

When both use cases were completed we asked the participant to reflect and elaborate on the different scores and words selected in the surveys. If any questions had arose from the questionnaire or if we had any uncertainties of the responses it was also discussed. The participant were also asked to reflect on the systems and deliver a final conclusion on the experience and their differences. At the end the participant was given both applications side by side, asked to interact and talk about what their impressions were with both systems next to each other. As a last question the participant was asked if they could identify which system was native and motivate their choice.

For the interviews an invitation was sent out internally at Netlight to all employees at the Stockholm office covering over 500 people. We were not making any selection or prioritizing, but simply following the service policy first-come, first-served. A conference room located at Netlight's office in Stockholm was booked for the meetings. For convenience and flexibility, a whole day was also booked for the possibility of drop-in testers.

Part II

Development

Chapter 4
Logical Stories

In this part we will describe our process when it came to developing the application, breaking it down into stories for a better overview of our process. Every Story will include a time summary and commentary on how the development process has been. For our first iteration we tried to follow the motto *"Simplest and fastest implementation possible"*. This had a direct impact on the design of the interface, as we focused on design in our second iteration. The time appended to every Story is the effective time and does not include any breaks.
The first survey was conducted early in the process and can be viewed in Appendix B. These responses serve as basis for our selection of native functionality.

The native development was divided by the two of us, one covering iOS and the other Android. React Native however was a combined development, often involving pair programming and alternatively going separate ways, but then merging solutions. The order in which we implemented the systems also alternated for every new Story, starting with native implementation first and React Native implementation second, then switching and implementing React Native first for the second Story. However, to maintain a consistent structure, we have chosen to always present the systems in the same order for every Story below.

4.1 Login

Story 1 - Login. Contains the task of connecting to Netlights REST API via a HTTPS connection. The user is authenticated with corresponding username and password. If the network is available and a correctly formatted input is provided then a HTTPS POST is sent to the API. A response code is then returned by the server. If the response is OK (200) the user will be logged in and a token is provided used for accessing employee data. An error message is provided if the user is Unauthorized (401) or Forbidden (403).

React Native

Total time: 5 hours and 45 minutes
Quick Summary: The first 2 hours and 30 minutes were spent learning the environment and familiarizing with the React Native mindset through development of the shell for the Story (two input fields and a button). The next 2 hours consisted of learning about the REST API: what to send, how to send and what we received from a correct message. Lastly the remaining 1 hour and 10 minutes consisted of error handling when connection was lost or wrong credentials were used. All areas were pretty straightforward, the main reason for the time can be contributed to our non-existent experience of this framework.

iOS

Total Time: 6 hours and 15 minutes
Quick Summary: Just like React Native this was almost an uncharted territory, so the first 2 hours and 30 minutes were spent in the same manner, setting up the user interface and altering the mindset to Swift development. With knowledge of the REST API most time was spent for parsing JSON (JavaScript Object Notation) and then navigating to the next page with this retrieved information. This did not come as easily as first predicted, but was managed in the end through extensive searches on the web. Also an unanticipated problem of centering all components contributed to this time, which has not been present in older versions of Xcode.

Android

Total Time: 5 hours 45 minutes
Quick Summary: With former experience in Android development there was no need for acclimatization to the same degree as in the aforementioned developing environments, but there were still some spaces of knowledge that needed to be filled. Most time was spent with a library that we had no former experience with, helping to make the asynchronous call to the REST API, namely RetroFit2 [35]. This would facilitate future development regarding scaling and introducing new calls to the API and receiving JSON data.

4.2 Information retrieval

Story 2 - Information retrieval. It contains the task of retrieving information from a remote server. Using the token from previous Story to download the all the employee information from Netlight.

React Native

Total Time: 2 hours 45 minutes
Quick Summary: For Story 2 we decided to go the same route as Story 1 and work in parallel on the same task and later on merge our progress. The Story was fairly similar with the previous Story of retrieving information, only that the amount of data was larger

and formatted differently. This Story required very little focus on GUI and we could therefore focus on the task of retrieving all employees from the REST API. After 2 hours and 14 minutes we had a working retrieval and could display any employee information we wanted. The last 30 minutes were spent turning the information into a scrollable list, for a better overview of the result. The only bump in the road for this Story was the structure we had in our previous declaration of the class from Story 1. The reason for this was our inexperience in React Native forcing a wrong structural decision for the task. With some refactoring of the code and some new features the application worked smoothly.

iOS

Total Time: 2 hours 15 minutes
Quick Summary: In this Story we used a framework called Alamofire, which is a library that enables easy asynchronous information retrieval and parsing which was more than welcome for this task [36]. The first 30 minutes included installing and fixing issues that arose with this. But after getting over the hump everything was pretty easy. The final problem was considered a blunder and concerned displaying the profile picture. If the employee selected did not have a picture it simply displayed nothing, which resulted in questioning of the code instead of realizing this problem.

Android

Total Time: 1 hour 45 minutes
Quick Summary: The large amount of time spent on the previous Story 1 and using the library RetroFit made this task easier. RetroFit is a Type-safe HTTP client for Android and with the use of a serialization library named gson was the conversion from JSON text to java object very easy [37]. The time spent on this Story was reduced as a result of the implementation of the previous Story. Although we experienced some issues regarding the conversion from JSON to a POJO (Plain Old Java Object), was it later solved.

4.3 Internal storage

Story 3 - Internal storage. During this Story we store the information retrieved in Story 2 to the mobile devices' internal memory. For security reasons we are not storing any sensitive information such as password or the token received from the API. Only a list of all employees containing public information such as email address and name are stored. This is done for improved user experience, eliminating the task of login for the user.

React Native

Total Time: 2 hours 45 minutes
Quick Summary: The approach was to create a new class that checks if data has been stored and depending on the result either opens the login screen or the setup screen. The class FetchStoredData handles the connection to the internal storage via an AsyncStorage

class. The implementation of storage was simple and easy to understand. An issue that emerged was how to handle the navigation from the FetchStoredData View because it should only forward information to the next View. We received some help from a Netlight employee who has previous experience in JavaScript and the problem was solved.

iOS

Total Time: 5 hours 30 minutes
Quick Summary: Using Core Data for internal storage in iOS turned out to be a bigger challenge than expected. Mainly because when we initiated the project we did not check the "Use Core Data"-box that would have made the whole experience a lot faster. The first 2 hours and 34 minutes were spent going in circles, following various tutorials that were dated. After giving up an afternoons work we began anew the following morning. Now everything went much faster and after 2 hours we had a working internal storage. The last 40 minutes were spent refactoring the code, making it more readable.

Android

Total Time: 3 hours
Quick Summary: All the data downloaded was stored temporarily in a Java structure converted from JSON to POJO, we now needed to store it permanently. One approach was to keep the structure and save it complete with a database, but a simpler solution was found. Since a library was and the structure for it was set up for the ability to convert JSON to Java objects, were it easier to permanently store the JSON file in internal storage. One issue we had was how to get the original JSON file before it was converted to a Java object by the library RetroFit. The issue was solved and minor changes in the structure was needed before the Story was complete.

4.4 Interaction with sensors

Story 4 - Interaction with sensors. This Story includes implementation of hardware functions such as GPS and accelerometer. We select the closest Netlight office, proximity wise, based on coordinates of the user. We also implement a way of recognizing a shake gesture based on the accelerometer by using a threshold value based on the x,y and z-values in the accelerometer.

React Native

Total Time: 8 hours 30 minutes
Quick Summary: When implementing GPS we found that React Native had a GPS module that could be utilized for both iOS and Android. It retrieved both the latitude and longitude for the user and with this we tried two different paths; one that would use Google's implementation of maps to get the closest office and the basic way where distances were calculated by simply using distance between two points. Importing and using Google

Maps turned out to be a challenge and so the naive way of looking up coordinates for all offices available and then calculating the minimum distance was deemed the best way. Accelerometer on the other hand turned out to be an issue with compatibility for the platforms. For iOS there existed an available library, but the same did not exist for Android. This put us in a new position when testing the React Native environment. We knew native Android has the capability to access their sensors so our goal was to write native Android (Java) code which would be able to build and run in React Native. The first module we created was an Android text notification and it was a success, the next step was to include the sensors. An issue that arose was to create a callback when a sensor was changed. It was later solved and React Native includes two different accelerometer solutions regarding the system context. So after 2 hours we had a functioning React Native iOS application with Android taking about 8 hours and 20 minutes to complete.

iOS

Total Time: 2 hours and 15 minutes
Quick Summary: Implementing GPS for iOS was a breeze once you get over the challenge of dealing with CLLocations. The first 1 hour and 29 minutes were spent getting the underlying mechanisms (GPS and accelerometer) to work and the last 38 minutes were dedicated to the choice of nearest office. Since coordinates for offices had previously been looked up, during React Native the development time was also shortened.

Android

Total Time: 2 hours
Quick Summary: Since we had an Android module in React Native with native code to access the sensors were it not a problem just copying that to the native Android application, taking only 13 minutes to complete the part. The remaining task to access the GPS was fairly simple when you have done a research on the subject. We choose to utilize the Google Play Service for location access and an own implemented function to calculate the distance between the offices from the coordinates.

4.5 Animation

Story 5 - Animation. Utilizing animations within the application. The animation itself is a grid of boxes which flip themselves when touched. If two boxes are flipped in a consecutive order and they contain the same value then they will stay in this flipped state for the remainder of the game; if the values differ then they will both simultaneously flip back to the initial state.

React Native

Total Time: 17 hours 30 minutes
Quick Summary: When we started working with React Native we found a library for flip

animations that was said to work with both Android and iOS. We found this library lacking, in terms of control. For an extended period of time we tried implementing the logic with the functions and callback provided, in hope of completing the Story. What we found, after hours of head scratching and wasted efforts, that the library simply was not sufficient enough for our intentions. So we edited the library after our needs, which in retrospect should have been the initial approach. We implemented variables and callbacks to our main view and could finally mark this Story finished. The other large implementation was to create a working Grid layout to handle the cards. A library was used but small issues with the layout and mapping of cards were troubling, resulting in longer development time.

iOS

Total Time: 3 hours 45 minutes
Quick Summary: The biggest struggle for implementation of iOS was finding the right way of presenting a grid. Getting the actual animation of flipping boxes to work was a breeze since there are several tutorials available online. After game logic also was added there persisted a problem of centering the grid layout. This was achieved by calculating the height of the device screen and then giving the collection a margin offset based on this.

Android

Total Time: 10 hours 30 minutes
Quick Summary: The development of the animation for Android experienced several issues and setbacks resulting in the long development time. At first we started with a fragment animation where the whole fragment worked as a card. This was troubling and the result was to only flip the image for the card. We also had issues with the grid layout, trying different solutions. The last and most time consuming issue that we experienced was that the first card in the grid was not clickable due to reinitiation. The issue was later solved but a lot of time was spent on debugging this behavior.

4.6 Other applications

Story 6 - Other application. Capabilities of launching other applications. When a match occurs in the memory game the profile information along with the ability to call, sms, email or add the contact information is shown. So for this story we tested the ability to launch these applications from the application itself.

React Native

Total Time: 2 hours 30 minutes
Quick Summary: To be able to open other applications on both Android and iOS, using the same codebase, we found a library called Linking. The only issue was that it was only usable in React Native version 0.20 and above and our project only had version 0.19. We came to the realization that an upgrade was justified, but in previous attempts of upgrade

we had experienced errors and then simply reverted the changes. This time we made more of an effort on debugging and a solution was found fairly quick within the hour, so the new version was now 0.21. For adding contacts we had to use a whole other library called Contacts, since Linking did not provide methods for this. The implementation of this took 30 minutes and so we had completed our first iteration of React Native.

iOS

Total Time: 3 hours
Quick Summary: Opening other applications with Swift is easy, just specify an URL and open it. So for instance, using the phone is opened with the string: "tel://12345". The time mostly reflects the issue of saving the contact information, more specifically if you want to open up the Contacts application in editing mode with some prefilled fields. For iOS 9 there exists a Contacts framework, compared to the old implementation AddressBook which is more unstructured. After 35 minutes everything was working: making a call, text and adding contact (without opening up the application in editing mode). The remaining 2 hours and 25 minutes were spent trying implement this last step. We decided after this time that, because of React Natives limited contact capabilities, simply adding the contact (without editing mode) would suffice.

Android

Total Time: 45 minutes
Quick Summary: The ability to open other applications is fairly simple on Android. You can start a new application based on some pre-set tags from Android. If the "EMAIL" tag is set will Android provide the user with all applications available on the system to perform the task. The user can then select the preferred application or set a default application that on next occasion will launch immediately. This is handled very simple and easy from Android's side, minimizing the work for developers, contributing to a fast implementation.

Chapter 5
UX Stories

The next step in our development meant UX and aesthetical implementation. By separating the logic and UX development into two parts we believed a clearer picture of the effort required was achieved. With the logical implementation done we were in for some light touch-up implementation, except for some hiccups along the way that meant some restructuring of the code.

For this part we have not included source lines of code, only time consumed for each Story. This is because of the previously mentioned restructuring, which proved to produce a very inconsistent result, that did not reflect the effort put in.

5.1 SetupView

Story 1 - SetupView. The first Story concerns the setup screen as can be seen in Figure 3.4. For this screen includes creating a board size selection to be able to select a board (6, 12 or 18) and an office selection containing the name of the office with an image from the corresponding city and number of employees. It also includes the ability to swipe between the offices. There was also some structure implementation needed to be improved for the ability to sort employees for the offices and select a random number of employees that will be included in the created game.

React Native

Total Time: 8 hours
Quick Summary: The first task was to create a board selection with three buttons. This was not as simple as expected. The structure we went with included an update to the whole state whenever a button was pressed and was something we weren't completely happy with, but settled for. The office selection was implemented by a library we found that worked with both Android and iOS. We made some small modifications and had a functioning

selector. What took time was figuring out how to get the selected office, which proved to be quite difficult with the structure we had set up. We finally solved it after some hours of implementing different callbacks.

iOS

Total Time: 10 hours
Quick Summary: Working with Storyboards has been somewhat of a challenge. It started as a small obstacle that was overcome. Or so we thought. Storyboards is very forgiving until you want to add a more complex structure. The layout editor is built in such a way that supports all screen sizes and this lead to confusion. The layout of preview was nothing like the layout of the application. Without a deeper understanding of Storyboards, many hours were spent in limbo. After some reading and testing we finally understood this visual editor and the next problem could be dealt with, office selection. This was not as easy as first predicted. Some hours were spent here as well, working with tutorials that only bared a smaller resemblance to the problem. The last hour was spent adding more attributes to the Employee data structure, since that had been left out in previous stories.

Android

Total Time: 10 hours 15 minutes
Quick Summary: The main obstacle of Story 1 was the component created to swipe between different offices. We had no similar experience within the subject so major time was spent on searching for a good approach. A solution was found but needed a lot of modifications both for implementation and for the layout to make it fit dynamically for every screen. The total time for only implementing the swipe locations were 6 hours with a lot of them spent on solutions that in the end was not usable. The other tasks was boardSelection which was fairly simple taking only an hour. With the remaining part being the structure modifications of the data due to new variables and a new list for every office. A large part of this was how to send a custom object to another activity. Hours was spent on finding a good solution and the question was also discussed internally at Netlight. The solution found laid the foundation for other activity's and Stories.

5.2 ShakeView

Story 2 - ShakeView. The profile images for the employees selected to be a part of the game are downloaded in the background. During this will the user be presented with clear instructions to perform a shake motion. If the ongoing download of profile images are not completed will a progress bar be presented asking the user to wait.

React Native

Total Time: 4 hours
Quick Summary: The underlying structure we set up for shake gesture included two different versions of the same function, differing on the platform used. This called for a little

bit of circumventing when implementing the callback of shake recognition. The first approach was using promises, which proved to be insufficient when it came to the Android structure, so this idea was scratched after some implementation and switched out for a more straightforward prop callback, that was successful for both platforms. The issue here was that React Native had not yet chosen a clear callback structure to use.

iOS

Total Time: 2 hour
Quick Summary: With new found understanding of Storyboards everything went by quickly. The shake screen has a simple design, just two text fields and a progress bar. Adding these elements and aligning them was easy and time was spent doing the background work, i.e. downloading all the images required and saving them in the objects that are sent to the next screen. The logic of choosing the default images was created as a simple check. Checking whether or not the image data was empty, rendering either an employee's profile picture or a default colored background with the employees username centered. All in all, it was a quick Story.

Android

Total Time: 3 hours
Quick Summary: The layout of the shake screen was simple and was finished in a couple of minutes. The large part was how to download the employee images and cache them properly for usage in the game screen. An issue that appeared was the problem to pass the downloaded profile images to the next Activity without causing an 'OutOfMemoryError'. It was thrown because the images needed to be sent were too large when selecting a large board. The issue was solved by compressing the images before sending them and on the other screen decompress them. The other problem was how to create a default image for employees without a profile image. The response status from the REST call was checked and if it was not 'OK' a default image with the employees username was created.

5.3 GameView

Story 3 - GameView. This Story mainly concerns the design of the cards in the board, further more using the usernames of the flipped cards to check whether a match has occured. When a match has taken place the employee information is packaged and then sent to the following screen.

React Native

Total Time: 3 hours 45 minutes
Quick Summary: This Story proved to be harder than expected. First we needed an overhaul of the previously implemented card-structure. Then came downloading images, or more specifically checking if the image was available. After some reading and trial and

error printing, we found a bug on the Android system of React Native that gave us unforeseen problems with JSON conversion of the received body when a picture was null. This required some ingenuity and we simply used the HEAD method of the HTTP-protocol, but that was done after some hours of trying to locate and fix the problem on the Android side. This solution circumvented the problem and the response header was all we needed when deciding if a default picture was to be used or not.

iOS

Total Time: 1 hour 30 minutes
Quick Summary: With everything already set up, the Story required very little additional code. The pictures were downloaded in the previous Story and the cards were simply supplied with the employee information. The time here was spent on the internal memory structure of employees.

Android

Total Time: 1 hour
Quick Summary: The Activity did not include any large changes, only to change the cards from the previous set default image to the new downloaded profile image and also prepare the data for the subsequently MatchView. A redesign was also justified since the design and size of the cards did not match the other platforms.

5.4 MatchView

Story 4 - MatchView. The MatchView is a transparent overlay of GameView and will present the employee that has been matched in the game. The view contains the employee's name, level and profile image. The user also has the option to call, text message, mail or add the person to his/her private phone book.

React Native

Total Time: 3 hours 15 minutes
Quick Summary: Implementing the match structure was pretty straightforward. Here we used manual insertion of the items and added a helper class for the action logic, to achieve more readable code. The problem here consisted of adding the screen over the game when a match was found.

iOS

Total Time: 3 hours 45 minutes
Quick Summary: The time spent during this Story concerned finding the best approach of displaying the MatchView overlay. The design and layout of MatchView was not a complicated matter, but took some time since a dynamic method with enumeration of

creating the buttons was chosen. In retrospect, a more static approach could have been used, which would have resulted in shorter time consumption, but this method opens up for future expansion of actions.

Android

Total Time: 1 hour 15 minutes
Quick Summary: Almost all time was spent on making the XML layout look as intended. Making the overlaying structure was not difficult with Android's dialog class. Getting the layout correct and structuring the component caused some trouble but was later solved.

5.5 FinishView

Story 5 - FinishView. To finish the game we added an additional screen, composed of static text messages and a return button. When the button is clicked the user will finish the game and return to the SetupView, where a new game can be initiated.

React Native

Total Time: 1 hour 45 min
Quick Summary: The overlay modal was similar to the MatchView Story with just some few design changes. The time was spent on composing a correct navigation via the return button displayed on the overlay. The navigation previously created needed some changes to be able to clear the top and start a new game from setup screen. It was also a initial step on creating a navigation for the application.

iOS

Total Time: 1 hour 45 min
Quick Summary: Navigating backwards was something we left out until the very last Story. The ability to navigate forward was in place, but backwards required some thinking. The issue here consisted of making the setup screen the root, even when the user had used the login screen. This was solved by setting the setup screen as the navigations root whenever it was loaded.

Android

Total Time: 30 minutes
The structure of FinishView was similar to MatchView, time was spent on making the layout look as intended. The navigation was also implemented creating a return button and making the SetupView the new root by clearing the history stack.

Part III

Result

Chapter 6
Online Survey

A short summary of the results from the online survey is presented below, described in Section 3.3.1, where we received around 40 responses from employees at Netlight.

The first question of the survey concerned the users operating system. From the responses can we see that it is close between Android and iOS, with 19 for Android and 21 for iOS, also with an honorable mention of one Windows phone user.

For our second question we asked:
"Would you rather use an app instead of a website? If so, what do you consider to be the reason for this?"
All answers to this question overwhelmingly correspond to time and efficiency. We illustrate the first part of this question as a pie chart, see Figure 6.1

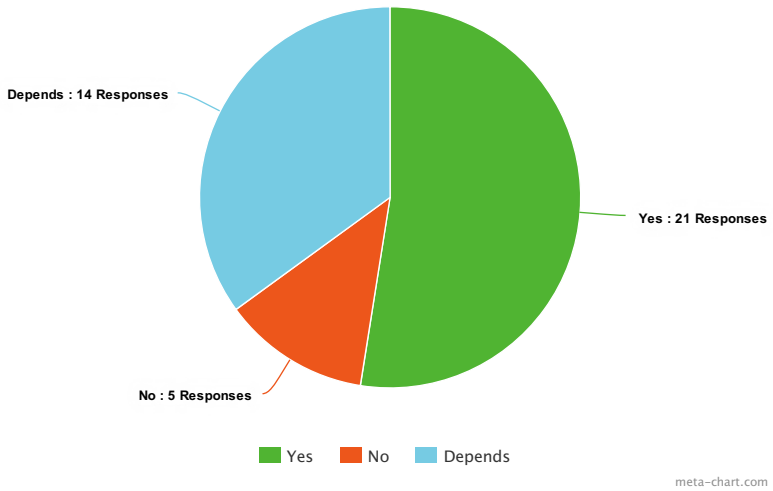


Figure 6.1: A pie chart representation of the answers to the question *"Would you rather use an app instead of a website?"*

When it comes to the majority—the ones who answered **Yes**—everything came down to easy interaction. Keywords found in the responses are: *navigation, scrolling, smoother, native integrated, loads faster, responsive, offline, native functions and overall experience*. Accessibility when it comes to login was also something that was mentioned throughout, e.g. credentials being stored in a more efficient way on the mobile device and not having to type the address to the website.

The 35% that answered **Depends** stated, with the exception of time, that the frequency of usage was a determining factor for the choice. If the need was once or twice they would rather use the website instead of an application; for more frequent usage they would prefer the application. The functionality and purpose of the applications would also be a deciding factor.

The minority—replying with a **No**—had three issues, with the required installation of applications as their main concern. Having too many applications on their mobile device and a feeling of being "Closed in" were the other two concerns.

Next was the question regarding what functions on a mobile device they appreciate, in contrast to the web.
Are there any specific functions within applications that you enjoy, in contrast to websites?
The responses we received were better than expected, though the answers where very extensive. It gave us a better picture of what the user views as native functionality and appreciates with mobile devices.

By analyzing the results we discovered some interesting facts. 15 people out of the survey has answered that the main functionality they enjoy more on a mobile device is the smoothness and flow within the application. Of these, five users specifically stated the scroll functionality. A lot of users have also mentioned the ability within mobile devices to keep track of the users settings and be able to continue where they left of. One participant answered:
"The main function is that I can be logged in all the time, and the app can remember me and my credentials".
Closely touching on the same subject have four participants said the ability to customize the applications to suit their own needs and make it feel personal.

17 users have also mentioned the direct access to the hardware such as GPS or accelerometer, but also the ability to utilize the limited amount of hardware spec on a mobile device. One user also stated:
"An app can integrate with the hardware in a much better way".

9 other participants have also mentioned the easy access to camera, alarms, offline support, voice control and easy multitasking.

The last question of the survey asked the participants to name some of their favorite mobile applications when it comes to look and feel. This was included in our development as a reference to what the users perceive as good user experience and design. The word cloud below lists every application that was mentioned by more than one participant. The most popular application is shown with the largest font.

Figure 6.2: A word cloud answering the question: *What is your favorite mobile application, more specifically when it comes to look and feel?*

Deliminators: The survey was sent out internally to a company where the majority of employees have a broad technical experience, the result may have been different if another target group was selected, since this is not a good reflection of the general public.
Slack is the internal communication system used by all employees at Netlight and this may be the reason for its large representation in the word cloud.

Chapter 7

Development

In this chapter we present all results gathered from both our development phases. This includes time consumption and source lines of codes that have been recorded throughout development. The results have been split into two sections involving the Logical Stories and the UX Stories.

7.1 Logical Stories

Table 7.1 contains the amount of time spent developing all Logical Stories.

Story	React Native (h)	Android (h)	iOS (h)
1	5.75	5.75	6.25
2	2.75	1.75	2.25
3	2.75	3.00	5.50
4	8.50	2.00	2.25
5	17.50	10.50	3.75
6	2.50	0.75	3.00
Total Time	39.75	23.75	23.00

Table 7.1: Table of corresponding hours of development for Logical Stories

Figure 7.1 shows the data from Table 7.1 in a grouped column, where the total time consumption for both Android and iOS is combined and presented as a gray area behind the columns.

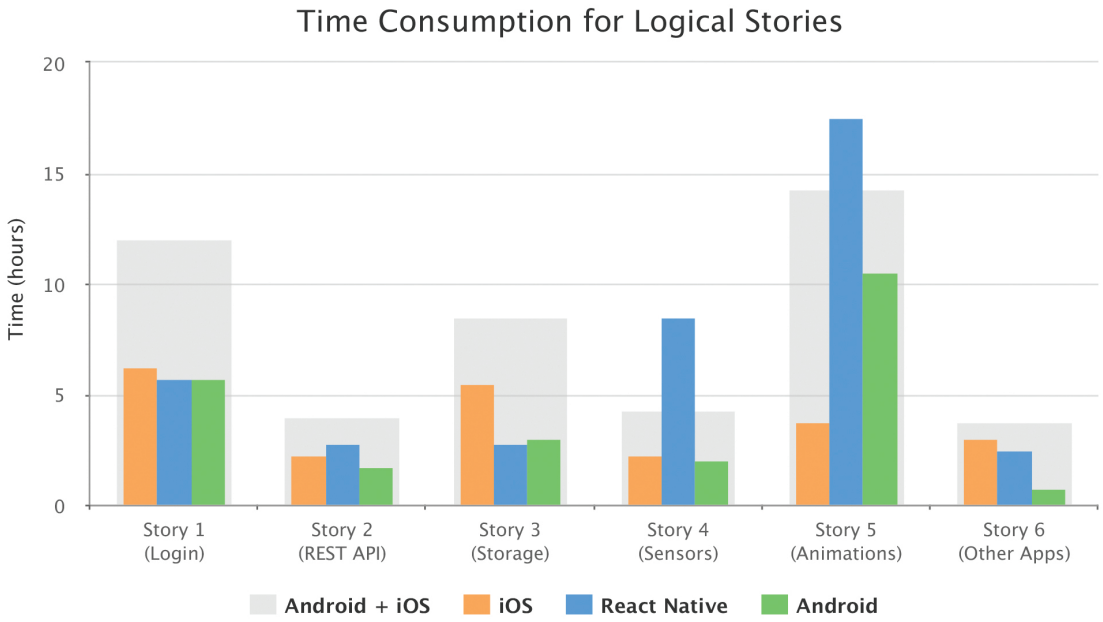


Figure 7.1: Graph presenting the corresponding development time for Logical Stories

Below, in Table 7.2, are the source lines of code for each Logical Story presented.

Story	React Native (SLOC)	Android (SLOC)	iOS (SLOC)
Initial	2458	1077	925
1	+360	+565	+264
2	-32	+406	+201
3	+92	+89	+269
4	+699	+235	+77
5	+265	+402	+256
6	+110	+84	+99
Added SLOC	+1494	+1781	+1166
Final SLOC	3952	2858	2091

Table 7.2: Table of corresponding source lines of code for the development from initiation and all logical Stories

Figure 7.2 presents the data from Table 7.2 in a line graph. Excluding the added lines of code after the creation of each project.

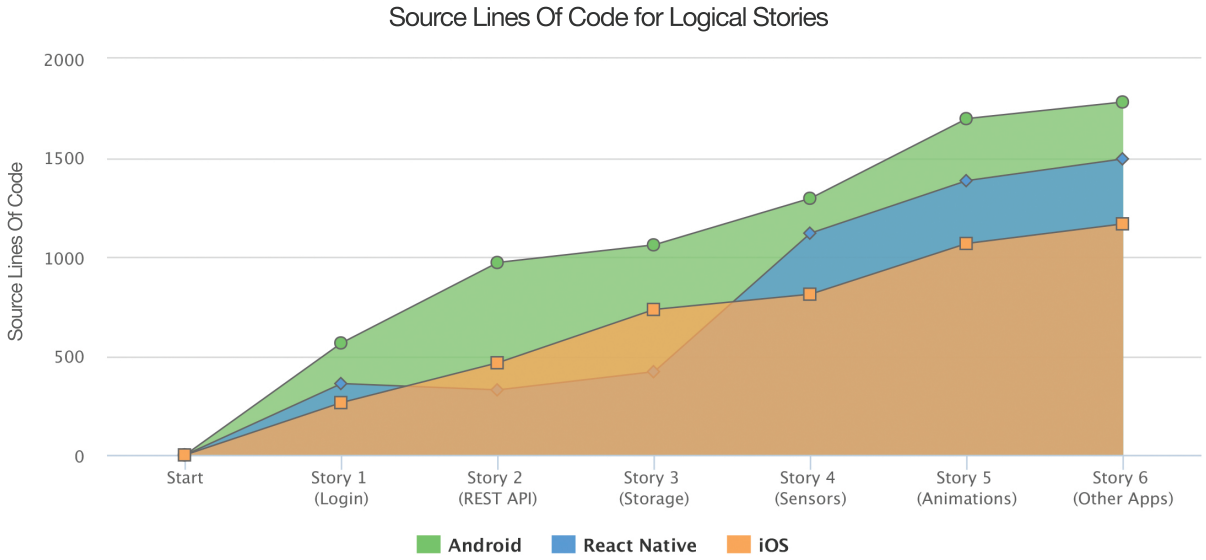


Figure 7.2: Graph presenting the corresponding source lines of code for all Logical Stories

7.2 UX Stories

Table 7.3 contains the amount of time spent developing all UX Stories.

Story	React Native (h)	Android (h)	iOS (h)
1	8.00	10.25	10.00
2	4.00	3.00	2.00
3	3.75	1.00	1.50
4	3.25	1.25	3.75
5	1.75	0.50	1.75
Total Time	20.75	16.00	19.00

Table 7.3: Table of corresponding hours of development for UX Stories

Here below, in Figure 7.3, is the time spent developing each UX Story presented in a grouped column, where the gray area behind the columns represents the combined development time of Android and iOS.

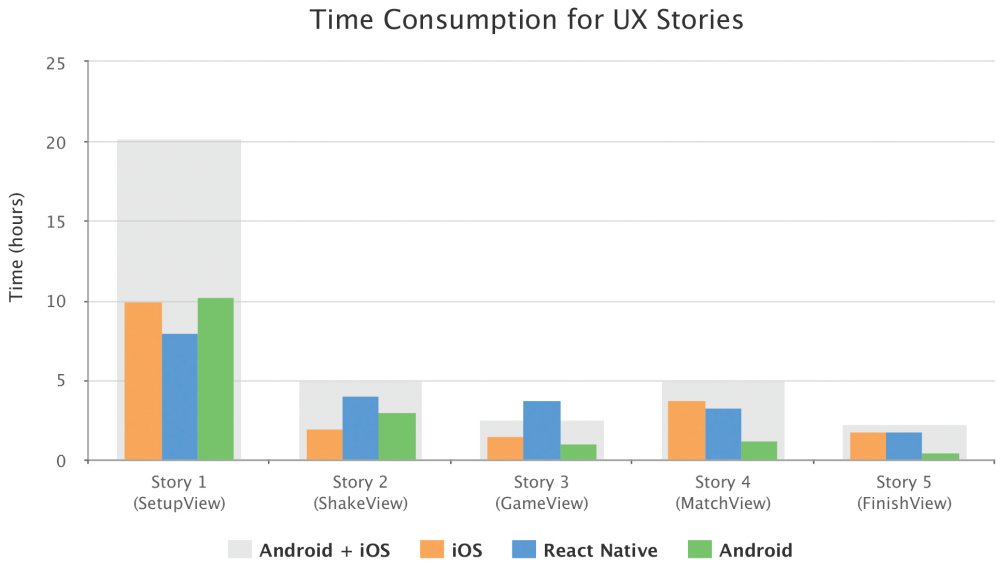


Figure 7.3: Graph presenting the corresponding development time for UX Stories

7.3 Comparison

The end-product can be seen in Appendix F. The applications is presented as a collection of screenshots for each screen, where they are taken from the two different platforms, iOS and Android.

Figure 7.4 presents the total time consumption for Logical Stories, UX Stories and the combined time. Android and iOS is shown as a stacked grouped column to visualize the time comparison better.

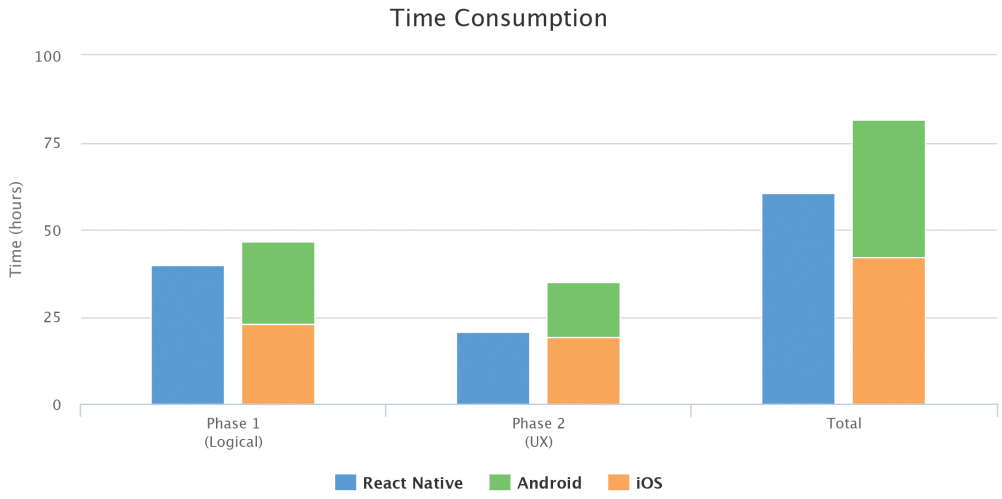


Figure 7.4: Graph presenting the total development time

Chapter 8

Performance

In this chapter we present the results from both **Reliability** and **Benchmark** testing.

The reliability checklist is presented in Table 8.1. Checkmarks are used if the system completed the test unaffected. The *Device Energy Saving Mode*-test proved to influence the smoothness in rendering of React Native Android animations and is therefore marked *x*.

Test	React Native Android	React Native iOS
Incoming and Outgoing SMS	✓	✓
Incoming and Outgoing calls	✓	✓
Incoming Notifications	✓	✓
Cable Insertion and Removal	✓	✓
Network Outage and Recovery	✓	✓
Device Energy Saving Mode	x	✓

Table 8.1: Table showing the outcome of the test cases performed on React Native for both Android and iOS.

Our benchmark tests conducted on the final product can be seen in Table 8.2. The values for CPU are gathered at the high points of consumption. The Memory consumption is an average value that the application is using during the entire test session. We have not included our results from network or battery tests, since these metrics were found to be irrelevant and did not add anything to our comparison.

Test	Android	React Native Android	iOS	React Native iOS
CPU	10%	35%	21%	45%
RAM	32 MB	46 MB	25 MB	42 MB
Application Size	14.6 MB	18 MB	24.8 MB	7.9 MB

Table 8.2: Table presenting the results of selected benchmark tests.

Chapter 9

Interviews

In this chapter we present the compiled result from out interviews, described in Section 3.5. We received a total of 11 participants for the interviews which we were pleased with since we projected between 10-15 participants. During the interview we used three different surveys for each participant to answer and create a concrete result base. The other part includes the feedback participants contributed with during the test in the form of oral discussions. From this we gathered some very useful information and future improvements of the applications. At the final stage, when the test was completed, we asked the users how they felt about the systems and if they noticed any differences.

The first thing that almost all of participants expressed was the large similarities in the systems and that they did not notice any differences. But when the participants were presented with both the systems side-by-side and had the ability to feel and compare the systems they fairly quickly discovered differences. The first screen they had the ability to compare was the setup and many of them noticed that the office selection was slightly slow and in some cases lagging for React Native. Some also noticed that the buttons used to switch board size had different colors when it was focused. During the shake screen did no participant have any comments to add, but regarding the succeeding game screen were several of the participant attentive of the card animation. What they noticed was React Natives inability to save images, causing the profile image to download during the flip animation and in some cases causing the card to be white before the image has downloaded. The participants on the React Native Android platform also discovered some bad performance on the flip card animation. The final question that the participants answered was if they after the ability to test the systems side-by-side could identify which system was React Native or native platform. All of the participants made the correct choice. Their decision was based on the above mentioned deviations.

9.1 SEQ

In the graph below, Figure 9.1, are the results presented from the Single Ease Question questionnaire, shown as a grouped columns for each task. Each group consists of the different systems that were developed and used during the interviews: **iOS, React Native iOS, React Native Android** and **Android**. A more detailed description can be found in Section 3.5.1.

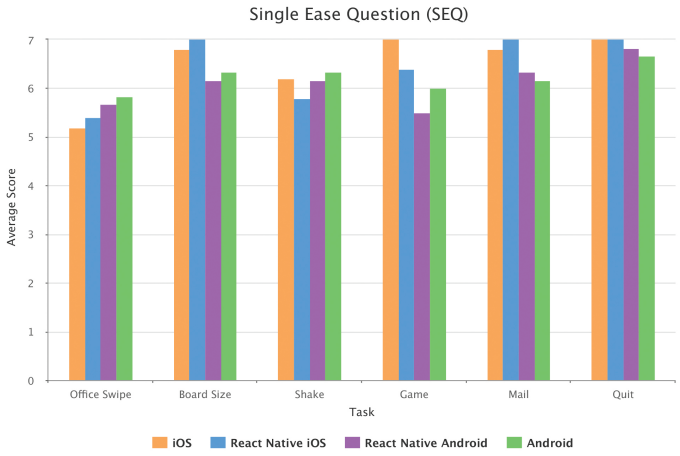


Figure 9.1: The average score given for each task

The standard deviation for the different tasks within the same system is presented in Table 9.1.

Task	Android	RN Android	iOS	RN iOS
Office Swipe	0.75	0.82	1.10	0.55
Board Size	0.82	1.17	0.45	0
Shake	0.82	0.75	1.79	1.64
Game	1.26	1.05	0	0.89
Mail	2.04	1.63	0.45	0
Quit	0.82	0.41	0	0

Table 9.1: Table of standard deviation of the SEQ score for the different systems

Figure 9.2 shows the average Single Ease Question-score from each participant is presented as grouped column for each user. The colors are representative of the system used in each of the interviews.

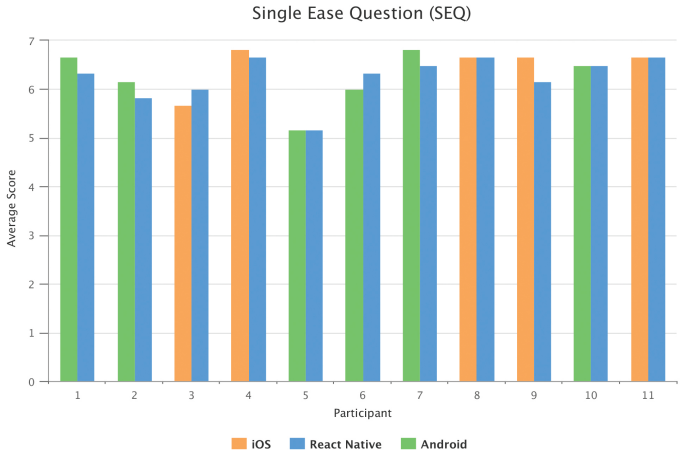


Figure 9.2: The average score given by each participant

9.2 SUS

The graph below, Figure 9.3, presents the total averaged score from the System Usability Scale, shown as four columns where the color represents the different systems used in the interviews. The average SUS score according to Jeff Sauro in "A practical Guide to the System Usability Scale" is 68 [32]. The score can then be divided into different percentiles presented as A-F. Even though a SUS score can range from 0 to 100, it isn't a percentage. A score of 70 would mean that it is just above the average 50%, the grades can be viewed below in Table 9.3. A more detailed description can be seen in Section 3.5.2.

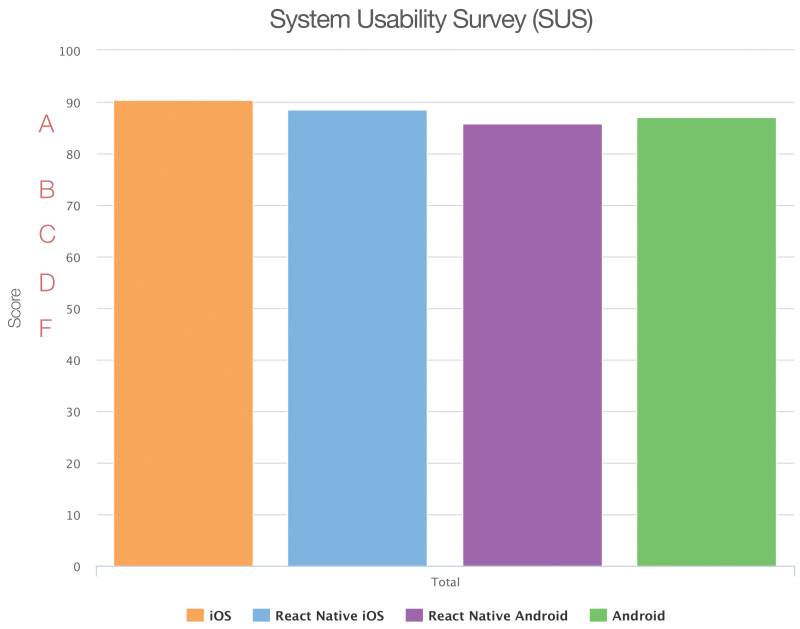


Figure 9.3: The total averaged score given for each system

The standard deviation of the responses from the Single Usability Scale for the different

systems is presented in Table 9.2.

System	Standard deviation
iOS	10.52
RN-iOS	9.78
Android	7.81
RN-Android	8.90

Table 9.2: Table of standard deviation of the SUS score for the different systems

Grade	SUS score	System
A+	95-100	
A	93-94	
A-	90-92	iOS (90.5)
B+	87-89	Android (87.1), RN-iOS (88.5)
B	83-86	RN-Android (85.8)
B-	80-82	
C+	77-79	
C	73-76	
C-	70-72	
D+	67-69	
D	63-66	
D-	60-62	
F	0-59	

Table 9.3: Table of corresponding SUS score with the grade

Figure 9.4 presented the individual score from each participant in the System Usability Scale. The score is shown as a grouped column with colors indicating the different systems used in each interview.

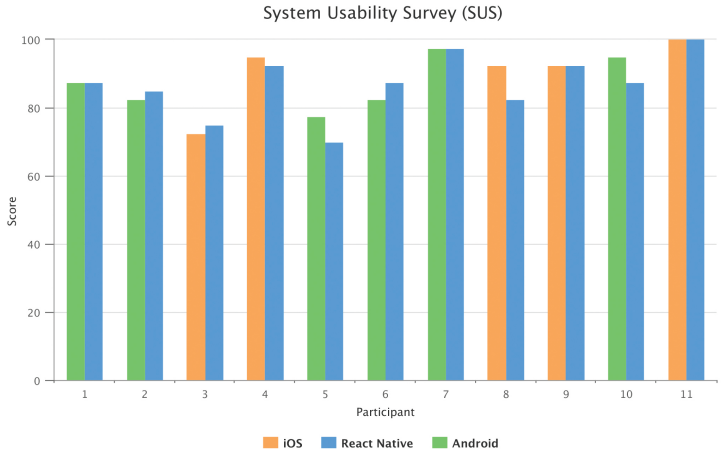


Figure 9.4: The score given by each participant

9.3 Wordlist

The collective result from the Wordlists are presented beneath in the form of a word cloud, Figure 9.5 and Figure 9.6. Where the size of each word is directly proportional to the frequency in which the words were selected by the users. The outcome was divided into two individual word clouds where Figure 9.5 shows the shared results for both native system and Figure 9.6 shows the complete results for React Native. A more detailed description can be seen in Section 3.5.3.

Figure 9.5: A word cloud presenting the most frequently selected words when describing the native platforms

Figure 9.6: A word cloud presenting the most frequently selected words when describing the React Native platform

Chapter 10
Discussion

We have divided our discussion into six sections, delving further into the questions that are outlined as Goals in *Problem Formulation*, Section 1.1.

10.1 Utilization of Hardware Sensors

Mobile applications work very close with the hardware and its functions, utilizing these unique possibilities that exists on mobile devices. Using hardware functionality, such as camera and vibration, enriches the user experience by making it both easier and better. This is something that is lacking on websites, often because it can differ vastly depending on what platform the user is located on.

One of our goals for this thesis included an investigation into React Native's capabilities of utilizing these hardware sensors. We used an online survey, see Chapter 6, to help identify two hardware sensors that could be implemented in our applications. The choice of only two sensor was done because of our limited time for development, narrowing the scope of our application. The survey concluded that accelerometer and GPS were the most used hardware sensors when it came to mobile devices. For our development we included a Logical Story that contained the implementation of these functions, namely *Story 4 - Interaction with Sensors*.

The results in Chapter 7 show a spike in the development time during Story 4, as can be seen in Figure 7.1. This higher time consumption pertains to the difficulties we faced when implementing the accelerometer, more so than the GPS. We can also see that this time is even greater than both native applications put together, outlined as a gray area in the background. The time consumption here was mostly spent on bridging native Android code to React Native, since a library for the accelerometer did not exist at the time. This bridging is also shown in Table 7.2, with this Story containing the most added lines of code.

But we would expect these time consuming tasks to decrease as the framework receives

more updates, libraries and patches.

10.2 Interruption Handling

To investigate React Native's interruption handling we set up a fairly simple test on the different platforms, outlined in Section 3.2.2. The results shown in Chapter 8 reveal a single error on React Native Android's Energy Saving Mode. This was marked with a 'x' because of the decrease in animation performance. The rest of the tests are marked with a check mark since we did not experience any drop in performance or any other problems during execution.

10.3 Performance

Performance is split into three subsections, for a better overview and concise discussion regarding the results we received.

10.3.1 CPU

We can see a significant difference between the native systems and React Native when looking at the CPU usage in Table 8.2. We found that the CPU usage is fairly stable when in idle mode, as only minor percentage changes occur. Since the application does not use any push notifications or other parts requiring connection will the application be idle in parts when not used. The CPU usage peaks at the flipcard animation in GameView and the loading of SetupView, this is expected since they are image and resource heavy parts. We can see that React Native shows a higher CPU usage than both the native systems, this is due to the animation part when access to native functions is crucial and affects performance tremendously.

10.3.2 Memory

The memory usage can be dependent on were the application is deployed. A phone with larger resolution will use larger images resulting in a higher value for the memory consumption. It should not be compared between the two platforms, but within the same device for a more accurate representation, where we found consistently larger memory usage from React Native. It's also interesting to note that on iOS, this value is almost double compared to the native platform.

10.3.3 Application Size

We also see a significant difference for iOS and React Native iOS in application size in Table 8.2. This is partially due to us using Swift, which means that the entire Swift library is embedded in the application file. Further inspection of this file reveals that the framework

library for Swift is about 17.3 MB for our project. If we were to subtract all Swift libraries this would leave us with an application size of 7.5 MB, much smaller than the original 24 MB seen in the Benchmark Table. Roughly about the same size as React Native iOS, but that is of course without any native iOS frameworks at all.

The application sizes for Android is fairly similar with native Android 3.4 MB smaller than React Native. The main part in the Android APK-file are classes of 8 MB and 5 MB of resolution files such as images and design elements. The React Native application has a smaller folder class with 4.3 MB, but has a large part for the libraries. This part mainly consists of libraries for React conversion to a runnable native Android application.

10.4 User Interface

From our interviews we received a resounding answer regarding the similarities between the user interfaces. A majority of the participants could not differentiate React Native from the native application by simply looking at them. This is something we also experienced when developing, the innate ability that React Native possesses of creating a native exterior. The end product was not identical, but even we had problems distinguishing them.

10.5 User Experience

The feedback from the interviews, that at first glance it was hard to differentiate the two systems, was followed by a 100% guessing rate when it came to identifying the systems once they were used next to each other. This situation describes almost all interviews we held. That the system in question was not easily identifiable until the other system was introduced right next to it. Differences in animation smoothness and responsiveness were perceived as the biggest indicator when the applications were used simultaneously. This criticism was mainly directed at the office selection and the card flip animation located inside the actual memory game. It has been a recurring issue that we have known for a while during development and have been struggling to implement equally across the different platforms. We were therefore not surprised with the outcome and agree fully with the participants' assessment.

Below we have included discussions regarding rest of the methods and results from our interviews.

10.5.1 Single Ease Question

First of all the validity and relevance of Single Ease Question come into question. Performing short and simple tasks and then rating the challenge with a number of one to seven is not the most indicative method of *why* something works or not, but we would argue that this method provides a pretty clear picture of *where* the advantage and problem areas reside. Looking at the average task score, seen in Figure 9.1, the overall lower scored tasks are **Office Swipe**, **Shake** and **Game**. These results correlates with what we received in form of oral discussion when participants were asked to motivate why they believed one

system was native compared to the other.

Additionally we have constructed another graph, Figure 9.2, to showcase the average SEQ score given by each participant. We believe this graph provides us with a better understanding of how the actual difference was perceived between the two systems, participant by participant. Here it can be viewed that the average score given for the React Native and native systems was almost identical when it came to the individual scoring, only differing slightly. Participant 9 was the one who scored the most varying result for the different systems, as seen in the graph. In the SEQ survey filled out by the participant it reveals that the task of shaking was the only answer that differed, scoring iOS with a seven and React Native iOS with a four. This difference in score was due to a perceived higher shake threshold in React Native compared to native.

We can see from both graphs that the Single Ease Question survey yielded similar results between the two systems used during each interview session, although the overall averaged score for each task in Figure 9.1 seems somewhat scattered. We can also see that the averaged score in Figure 9.2, although dispersed among the participants, shows a consistently close score between the two tested systems.

10.5.2 System Usability Scale

The System Usability Scale was used in our evaluation as an overall score of the different systems. The questions from this survey were not always completely relevant, making this test rather impractical on some points. But we used it in a broader sense, trying to find differences in the systems on a more official and standardized level. What can be seen in Figure 9.3 is a consistent superior result for the native systems compared to React Native. Just like SEQ we also presented the SUS score for each participant, gaining a better understanding of the perceived difference for each participant.

The biggest difference in score is given by Participant 8, that scored the systems equal in the Single Ease Question survey. This participant stated that the responsiveness and speed of animations caused this variation in scoring, commenting that the difference was minuscule.

With the average SUS score we found the overall grades to differ, even though they were closely related, which can be seen in Table 9.3. Although different grades, the difference between the best and the worst is only five points. The difference in grades between the React Native applications is something we attribute to the animation setbacks of React Native Android, as this drop in performance was reiterated numerous times. This was also something that we were aware of and was not surprised of the outcome. But with an application receiving a score above 70 as the accepted level did all application perform well and have made the cut. An application rating as high as B and above would be seen as well performing and should definitely be published. Although native Android has been rated lower than iOS can we see that the difference between React Native and native are equal within the same system.

10.5.3 Wordlist

If we compare the results from the two word clouds are they at first glance very similar. Many of the participants did not experience any difference in the systems and explicitly said during the interview that they tried to mark the same words for both surveys.

The small differences can depend on the first impression, what comes to mind, which we encouraged the participants to elaborate on. Another variable could be which system they received first, making the second survey differ in aspects of what they had previously learned by using the first system.

When taking a closer look at the word clouds we notice some words that establish a difference between the systems. Five participants found the native platform **Fast** compared to none for React Native and we also found one participant who answered with the antonym **Slow** for React Native.

For the native platform we also found words as **Responsive**, **Clear** and **Understandable** which can't be viewed for the React Native platform. But in comparison we found words as **Simple** and **Fresh** stand out for React Native. This is a very similar responses and only varies marginally in the word selections.

React Native has also received some less positive reviews that we can't find for the native platforms, such as **Annoying** and **Unpredictable**. It displays the users' requirements on a system and that even if there are small differences the user will notice them and consequently reduce their rating of the application.

The result should still be considered in a positive light since the majority of the participants seem to feel that there is no difference in the systems and would prefer either one.

10.6 Development

When starting this project neither of us had any major experience in JavaScript nor React. A lot of the issues we faced had the origin in us not knowing how the structure of React should be set up. We have on some occasions been forced to ask for help from our fellow colleagues at the office for help with JavaScript. They have, without major problems, been able to understand and help us in our situation, showing us how effortless it is for a developer with previous knowledge in JavaScript and React to not only understand, but also contribute to the project. The threshold from knowing JavaScript to start programming in React Native is low if we compare it to our own experience when adapting from Java to Android. We found this conversion harder due to a whole new Activity lifecycle and layout managers with Android XML that needed a new mindset.

Our biggest challenge during the development has been the animations of card flip. Although animations have been smooth with iOS, there has been significant problems with rendering on the Android side. This was solved with React Natives lifecycle methods, where implementation was needed to make underlying modules decide whether to update or not, thus heavily influencing the rendering work done and directly impacting the smoothness of said animations. Controlling these functions is of course a strength if used

properly, but can be a hassle when you are forced to even reflect upon something that is done so seamlessly on the other platforms.

The actual application result, from our point of view, still needs improving though. The libraries we used were in some cases lacking the specific functions we needed, but were created by independent users, not with an official Facebook stamp of approval. So the initial approach we used of simplest first lead to our frustration in development when dealing with the UX-development. This aside, the amount of code in our project that is shared between the platforms is nothing less than astounding. If we look at the *Added SLOC*-row in Table 7.2 we see a clear advantage of shared logical lines of code, where React Native contains fewer lines of code than Android, but executable on two platforms. The time consumption is also impressive were React Native is more efficient than developing to both native platforms. It can be seen in Figure 7.4 that React Native during Phase 2 (UX) is similar to iOS regarding the development time, then it should be remembered that a Android application is also produced.

10.6.1 Community

One of the major strengths in React Native is the open source community. Contributing to the React Native development and supporting the employees at Facebook with bug fixes and new ideas. To this day have over 750 developers been a part of the development of React Native [38]. We would even venture to say that this community is the strongest advantage React Native has. The community frequently contributes and helps this product blossom, thus making it apparent that it is not only Facebook who wants this framework to become great, but also the community as a whole. The community is also contributing with free components and modules, available for anyone to include in their application. React Native for Android is very new to the market with its release September 2015 and iOS more than a year earlier. The head start that iOS has received is most noticeable in the amount of external libraries and modules created by the community.

This accessibility has been tricky since the libraries do not always work as intended or equally across both platforms. Since there is no control of the libraries, the use is at own risk, whereas Facebook controls and decides what features should be implemented or not in the official React Native release. This has in some case resulted in us using an existing iOS library, but needing to create our own for Android. It is a great feature (if something is missing you can always extend native code), but it's time consuming, seen in the Figure 7.1 *Story 4 (Sensors)*.

10.6.2 Developer Support

Although Facebook has an extensive documentation of React Native, it is sometimes not sufficient. Online questions and solutions from other independent programmers is a vital part of the development process, opening up other perspectives to the issues, at least when you are new to the framework. This is something that is missing for React Native as the tool is practically untouched compared to the giants like Android and Swift, where you can expect to find multiple solutions and discussions for the same issue. For example, a problem we faced during development was using the state variables in an efficient way.

It was hard to get a good understanding regarding the best approach for updating these variables throughout underlying modules and using callbacks efficiently for communication between parent and child views. Since the documentation was fairly limited regarding best usage of states, callbacks and lifecycles, it were hard to figure out which the *"right way"* to go was.

Even if the time consumption looks very promising for React Native in Figure 7.4, it can be misleading. The total time spent during the design phase does not factor in the copious amount of time that bug-fixes and small adjustments required. These debugging periods can partially be attributed to our inexperience, but we would argue that this is a consequence of the unstructured approach the documentation yields. *"This section is more experimental than others because we don't have a solid set of best practices around callbacks yet"* is an example of this, quoted from the React Native website [39].

10.6.3 Potential Errors

The potential errors from the interviews can be traced back to the sampling size. With only six participants testing Android and only five testing iOS, it is hard to argue that these figures are an absolute truth. But for SUS it has been shown that a sample size of five, the sample mean is within six points of a very large sample SUS score 50% of the time [40]. Even though we had a small sample size the result could still get close to the actual SUS score in more than half of the cases. As can be seen in 9.2 are the standard deviation not so big, with the largest of iOS with 10.52. An even smaller standard deviation would of course be better but with the small sample size we have are the results good.

Looking at the standard deviation from the Single Ease Question survey in Table 9.1, we see a pretty scattered result. Android had a consistently high deviation, as well as React Native Android, for every task. This shows an uncertainty when it came to setting a score for the completed task, or at least a non consensus when it came to the difficulty. In retrospect, we realized that the tasks were too easy altogether, which made the scoring a tad confusing for the participants. But since we were comparing between the systems, the scoring was almost equally scattered, as seen in the table. What we found, despite only having a handful of testers, was that the conversation and thoughts on the applications were pretty similar for every interview. This is something we figured was useful, not relying on SEQ score for a definite answer, but serving as a catalyst for this vital conversation.

Chapter 11

Conclusion

A conclusion regarding React Native is something that encompasses more than just the final product. In our thesis we included the developer's side of it all, an often forgotten part when it comes to the output. React Native is something that thrives on the restlessness and willfulness of the community, the people that selflessly contribute time and ideas to make React Native a successful framework. We firmly believe that if the environment in which developers spend their time is welcoming and supporting, then it would spring about more products in this framework. But just like we mentioned, this is a two-way street, the end product must be something that the general public would use, and in the same sense the framework needs to contribute with a friendly environment for developers.

11.1 Developer Impression

The biggest readjustment we faced was JavaScript and React coupled with React Native's way of thinking. Moving away from Object Oriented Programming and closing in on module implementation 'callbacks and promises, states and render methods'. Here we faced question marks regarding which path was the wisest, were the documentation proved insufficient.

For developers the threshold from knowing JavaScript to start programming in React Native is low compared to other systems, making it easy for developers to be apart and contribute to a new project fast. The React Native implementation also shows a smaller time consumption than Android and iOS combined, thus proving a more time efficient option than creating two native applications.

11.2 User Experience

What we found in our interviews was an overwhelming consensus, that the differences were minuscule. Users found React Native's capabilities of producing a superficial replica close to perfect and did not notice any difference from a native application. There are only minor differences in the surveys regarding usability SUS and SEQ. These results are also reflected in the wordlist were the users are homogeneous with only small changes in their selection.

A closer look at the results shows us a slightly worse score for React Native. With problems according to the test participants, residing in the functionality of three components, **Shake**, **Office Selection** and **Card Flip**. Two of them involving animations, which were issues we struggled with during development and would even venture to write off as inabilities from our side. Note that everything worked, but not performing identically. Which we believe is caused by the one-year head start that React Native iOS has over React Native Android.

11.3 System Performance

During the implementation of hardware sensors did we experience React Natives easy implemented connection and approach to access mobiles hardware sensors. The issue we encountered was the lack of an accelerometer for React Native Android, were another side of React Native was revealed, namely the possibility of native implementations. When a library is not provided by React Native or a function needs native control the user can by writing native code, develop their own module, creating a more modifiable and customizable environment.

The measured performance from all systems are similar with only minor differences. The native platforms performance regarding animations are slightly better which is something we have been struggling with during the development phase and it is also a part users have commented on. We believe that this is due to the short time React Native Android has been out on the market compared to React Native iOS and are according to us requiring some improvements in the performance area.

11.4 Recommendation

After developing and experiencing the results from the different frameworks first hand, it is still not easy to give a clear recommendation. We would argue for a more situational recommendation. This is because of the very early stage React Native is in at the moment, where the progress in the framework and knowledge from team members might dictate usage.

For us, the prior knowledge of JavaScript is the definite determining factor when it comes to deciding whether to develop an application in either native or React Native. This is because we disregard the perfomance metrics, since the CPU and RAM usage is in an

acceptable range. And the hurdles one faces with components and responsiveness is something we strongly believe can be solved or even avoided if prior knowledge of JavaScript or React already exists. If you have a team with knowledge in JavaScript or React we would definitely recommend React Native.

If experience in native platforms already exists then the migration to React Native can be confusing at times, but far from impossible. Trading in visual editing for performance management in form of React lifecycles, auto-generated classes and design patterns for a simple text editor and Facebook's insufficient documentation. This needed change in mindset can lead to small inconsistencies and unforeseen issues throughout the project. Many of the issues that we faced during development can be narrowed down to this inexperience in React and the JavaScript language.

With no experience in either of the above would we recommend starting with React if you would like a fast and easy way to deploy two systems. The learning curve of React Native compared to native is fast. React Native enables quick prototyping at a very high initial velocity since the rules for doing something is not forced upon the developer and these regulations can be perceived as cumbersome when starting out with native.

For native however, the connection between UI and logic takes some time of getting used to, opposed to the fast and seamless React Native with its UI-code placed in the same class. But with this you can loose the structure that comes with a visual layout editor which, can be something efficient and powerful. This question in trade off is something that can be reduced to the need of visual complexity in the application to develop. If the application itself only needs basic functions and basic visual elements, then React Native might be the way to go.

For more intricate visual graphics, maybe think twice and research React Native's capabilities on these particular tasks. There always exist the possibility to extended React Native with native code and native views, and also integrate well with other controllers.

Chapter 12
Future work

Although our research has solely focused on apps regarding smartphones could this be expanded. Tablet is a platform we have left untouched mainly because of the time frame limiting the project size. The market of tablets and user base are big would it be a creditable path to follow. It is possible to use React Native applications on tablets, but how well does it perform in comparison to mobile, are some interesting questions that could examine. React Native has recently implemented support for development in Apple tvOS, but not the complete feature set as in React Native for smartphones. Still an interesting market with better and faster performance in SmartTVs.

At the F8 Developer conference 2016 Facebook (the same company behind React Native) announced that both Microsoft and Samsung are committed to bring Microsoft's Windows 10 and Samsung's Tizen to React Native. This would open a new playground for React Native with two major players on the market supporting the system. It would also mean that the developer can create applications for the Universal Windows Platform and for Samsung's Tizen which mostly powers their Smart TVs and smartwatches.

React Natives user base and interest are increasing every day, with growing amount of contributors and partners, will it be interesting to follow the journey ahead.

Bibliography

[1] VisionMobile Ltd. (2015). *Developer Economics: State of the Developer Nation Q3 2015* London, Great Britain: VisionMobile.

[2] Adobe Systems Inc. (2016). Adobe Phone Gap. Retrieved January 28, 2016, from http://phonegap.com/

[3] Appcelerator Inc. (2016). Mobile App Development Platform & MBaaS | Appcelerator. Retrived January 28, 2016, from http://www.appcelerator.com/

[4] Xamarin Inc. (2016). Mobile App Development & App Creation Software - Xamarin. Retrieved January 28, 2016, from https://xamarin.com/

[5] Facebook Inc. (2016). React Native | A framework for building native apps using React. Retrieved January 28, 2016, from https://facebook.github.io/react-native/

[6] Apple Inc. (2016). Apple - Press Info - Apple Reports Record Fourth Quarter Results. Retrieved January 28, 2016, from http://www.apple.com/pr/library/2015/10/27Apple-Reports-Record-Fourth-Quarter-Results.html

[7] Rivera, J. Meulen, R. (2015, August 20). Gartner Says Worldwide Smartphone Sales Recorded Slowest Growth Rate Since 2013. The Gartner

[8] Apple Inc. (2016). Swift - Overview - Apple Developer. Retrieved January 28, 2016, from https://developer.apple.com/swift/

[9] IDC Research, Inc. (2016). IDC: Smartphone OS Market Share 2015, 2014, 2013, and 2012. Retrieved January 28, 2016, from http://www.idc.com/prodserv/smartphone-os-market-share.jsp

[10] Facebook Inc. (2016). A JavaScript library for building user interfaces | React. Retrieved February 1, 2016, from http://facebook.github.io/react/

[11] Facebook Inc. (2016). React.js Conf 2015 Keynote - Introducing React Native | Engineering Videos | Facebook Code. Retrieved February 1, 2016, from https://code.facebook.com/videos/786462671439502/react-js-conf-2015-keynote-introducing-react-native-/

[12] Facebook Inc. (2016). F8 2015 - React Native and Relay - Bringing Modern Web Techniques to Mobile | Engineering Videos | Facebook Code. Retrieved February 1, 2016, from https://code.facebook.com/videos/931163756933706/f8-2015-react-native-and-relay-bringing-modern-web-techniques-to-mobile/

[13] GitHub, Inc. (2016). facebook/react-native: A framework for building native apps with React. Retrieved February 1, 2016, from https://github.com/facebook/react-native

[14] Occhino, T. (2015). React Native: Bringing modern web techniques to mobile [blog post]. Retrieved February 1, 2016, from https://code.facebook.com/posts/1014532261909640/react-native-bringing-modern-web-techniques-to-mobile/

[15] International Organization of Standardization (2010). ISO 9241-210: Ergonomics of human-system interaction. Retrieved June 15, 2016, from http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=52075

[16] Albert, B. & Tullis, T. (2008). *Measuring the User Experience Collecting, Analyzing, and Presenting Usability Metrics* (2nd ed.). Waltham, MA, USA: Morgan Kaufmann.

[17] Google Inc. (2016). Design | Android Developers. Retrieved January 28, 2016, from http://developer.android.com/design/index.html

[18] Apple Inc. (2016). iOS Human Interface Guidelines: Designing for iOS. Retrieved January 28, 2016, fro https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/

[19] Schaaf, H (2015). Diary of Building an iOS App with React Native [blog post]. Retrieved January 28, 2016, from http://herman.asia/building-a-flashcard-app-with-react-native

[20] Petrash, A (2015). A developer's review of React and React Native's use and potential [blog post]. Retrieved January 28, 2016, from http://www.ekreative.com/blog/a-developer-s-review-of-react-and-react-native-s-use-and-potential/

[21] Lassen, A (2015). How Facebook's React Native Will Change Mobile Apps | TechCrunch [blog post]. Retrieved January 28, 2016, from http://techcrunch.com/2015/04/20/how-facebooks-react-native-will-change-mobile-apps

[22] International Organization of Standardization (2010). ISO/IEC/IEEE 24765: Systems and software engineering. Retrieved June 15, 2016, from http://www.iso.org/iso/catalogue_detail.htm?csnumber=50518

[23] Nimbalkar R. (2013). *Mobile Application Testing and Challenges. International Journal of Science and Research (IJSR)*. Retrieved January 29, 2016, from http://www.ijsr.net/archive/v2i7/MDIwMTM1OA==.pdf

[24] Google Inc. (2016). Performance Profiling Tools | Android Studio. Retrieved January 29, 2016, from http://developer.android.com/tools/performance/index.html

[25] Zhang X. (2001). *Application-Specific Benchmarking*. PhD Thesis, The Division of Engineering and Applied Sciences, Harvard University, Cambridge, MA, USA. http://www.eecs.harvard.edu/~syrah/application-spec-benchmarking/publications/thesis.pdf

[26] Apple Inc. (2013). Performance Tools. Retrieved January 29, 2016, from https://developer.apple.com/library/ios/documentation/Performance/Conceptual/PerformanceOverview/PerformanceTools/PerformanceTools.html

[27] Google Inc. (2016). Google Forms. Retrieved February 1, 2016, from https://www.google.com/forms/about/

[28] Beck K.; et al. (2001). Manifesto for Agile Software Development. Retrieved February 1, 2016, from http://www.agilemanifesto.org/

[29] Peyrichoux, I. (2007). When Observing Users Is Not Enough. Retrieved January 28, 2016, from http://www.uxmatters.com/mt/archives/2007/04/when-observing-users-is-not-enough-10-guidelines-for-getting-more-out-of-users-verbal-comments.php

[30] Sauro, J. (2012). 10 Things To Know About The Single Ease Question (SEQ). Retrieved April 22, 2016, from https://www.measuringu.com/blog/10-things-SUS.php

[31] Sauro J. & Dumas J. (2009). Comparison of Three One-Question, Post-Task Usability Questionnaires. In CHI 2009, *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (p. 1599-1608). New York, NY, USA.

[32] Sauro J. (2011). *A Practical Guide to the System Usability Scale: Background, Benchmarks & Best Practices* Denver, CO, USA: CreateSpace Independent Publishing Platform.

[33] Dr. Travis D. (2008). Measuring satisfaction: Beyond the usability questionnaire. Retrieved April 22, 2016, from http://www.userfocus.co.uk/articles/satisfaction.html

[34] Lookback (2016). Super simple user research with Lookback. Retrived January 29, 2016, from https://lookback.io/

[35] Ruesch J. (2013). Android Async HTTP Clients: Volley vs Retrofit. Retrieved February 9, 2016, from https://instructure.github.io/blog/2013/12/09/volley-vs-retrofit/

[36] Github Inc. (2016). Alamofire. Retrieved February 16, 2016, from https://github.com/Alamofire/Alamofire

[37] Github Inc. (2016). Google-Gson. Retrived February 16, 2016, from https://github.com/google/gson

[38] Github Inc. (2015). Contributors to facebook/react-native. Retrieved May 11, 2016, from https://github.com/facebook/react-native/graphs/contributors

[39] Facebook Inc. (2016). Native Modules – React Native | A framework for building native apps using React. Retrieved April 28, 2016, from https://facebook.github.io/react-native/docs/native-modules-ios.html

[40] Jeff Sauro. (2013). 10 Things To Know About The System Usability Scale. Retrieved June 14, 2016, from http://www.measuringu.com/blog/10-things-SUS.php

Part IV

Appendix

A Timeplan

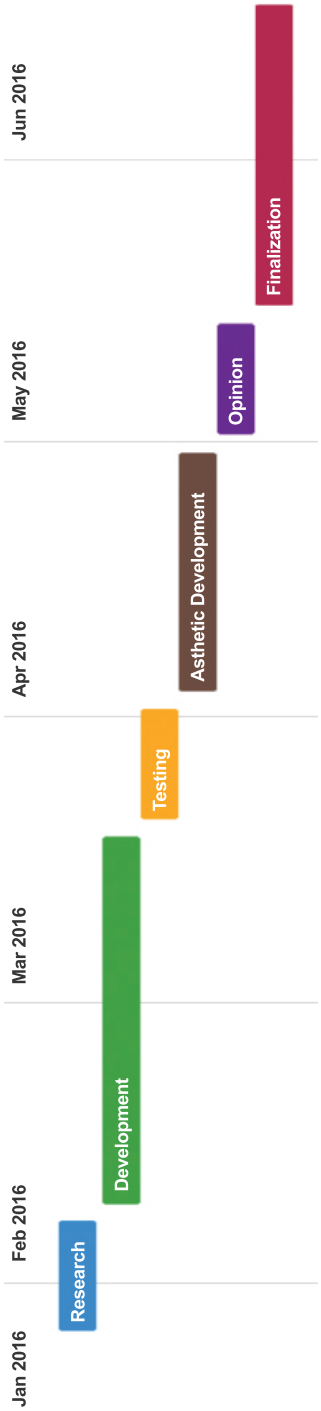
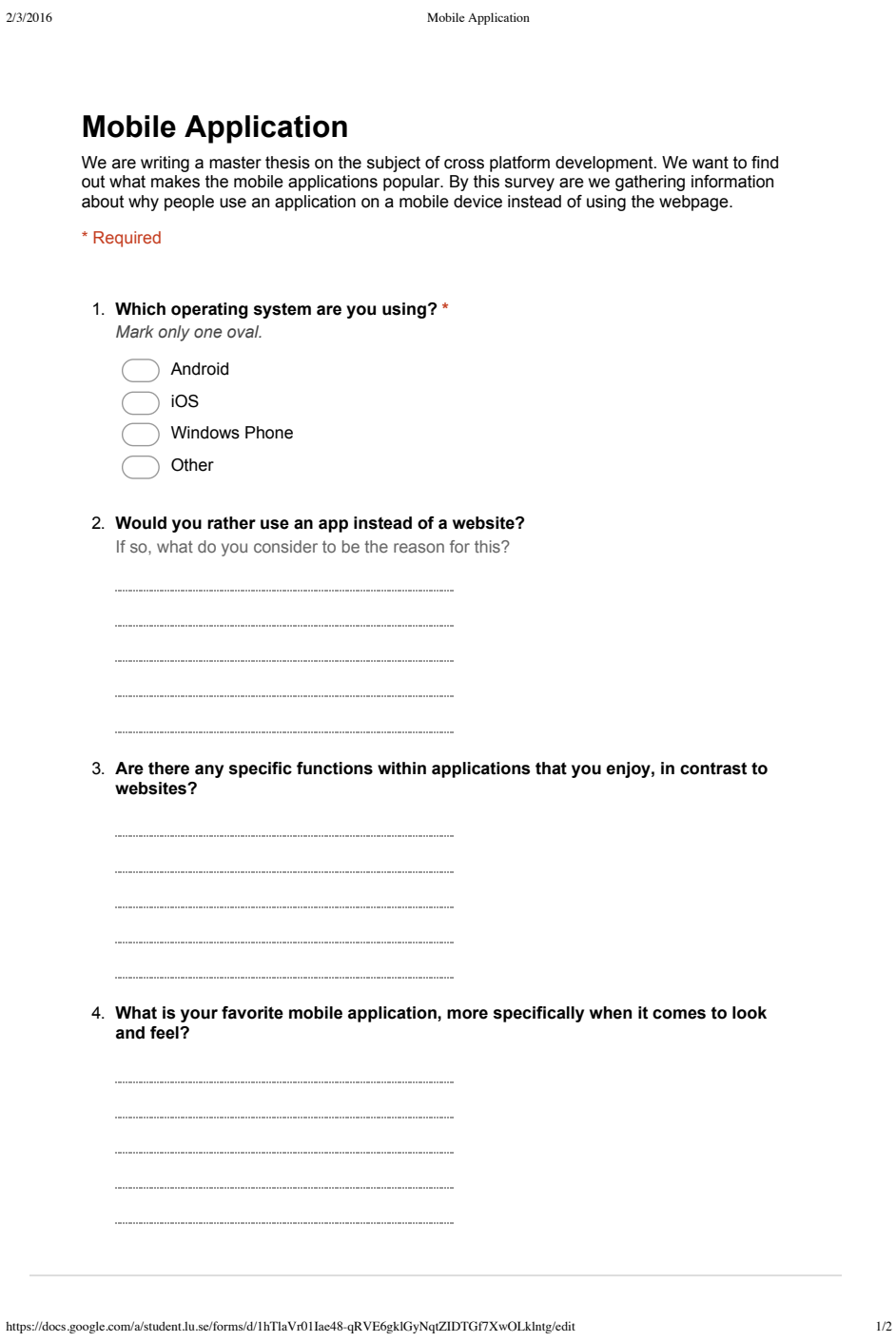


Figure A.1: Our timeplan for this project

B Online Survey



2/3/2016 Mobile Application

Mobile Application

We are writing a master thesis on the subject of cross platform development. We want to find out what makes the mobile applications popular. By this survey are we gathering information about why people use an application on a mobile device instead of using the webpage.

* Required

1. **Which operating system are you using?** *
   Mark only one oval.
   - Android
   - iOS
   - Windows Phone
   - Other
2. **Would you rather use an app instead of a website?**
   If so, what do you consider to be the reason for this?
3. **Are there any specific functions within applications that you enjoy, in contrast to websites?**
4. **What is your favorite mobile application, more specifically when it comes to look and feel?**

https://docs.google.com/a/student.lu.se/forms/d/1hTlaVr01Iae48-qRVE6gklGyNqtZlDTGf7XwOLklntg/edit 1/2

Figure B.1: The online survey used to identify native behavior

C Single Ease Question Survey

SEQ

Generellt, hur svårt eller lätt var det att utföra uppgiften?

1. **Välja kontor**
   Markera endast en oval.

	1	2	3	4	5	6	7	
Väldigt svårt	◯	◯	◯	◯	◯	◯	◯	Väldigt lätt

2. **Välja brädstorlek**
   Markera endast en oval.

	1	2	3	4	5	6	7	
Väldigt svårt	◯	◯	◯	◯	◯	◯	◯	Väldigt lätt

3. **Skaka**
   Markera endast en oval.

	1	2	3	4	5	6	7	
Väldigt svårt	◯	◯	◯	◯	◯	◯	◯	Väldigt lätt

4. **Spela**
   Markera endast en oval.

	1	2	3	4	5	6	7	
Väldigt svårt	◯	◯	◯	◯	◯	◯	◯	Väldigt lätt

5. **Mail**
   Markera endast en oval.

	1	2	3	4	5	6	7	
Väldigt svårt	◯	◯	◯	◯	◯	◯	◯	Väldigt lätt

6. **Avsluta**
   Markera endast en oval.

	1	2	3	4	5	6	7	
Väldigt svårt	◯	◯	◯	◯	◯	◯	◯	Väldigt lätt

Figure C.1: Single Ease Question survey used in each interview

D System Usability Scale Survey

System Usability Scale Survey

1. **I think that I would like to use this system frequently.**
   Mark only one oval.

	1	2	3	4	5	
Strongly Disagree	◯	◯	◯	◯	◯	Strongly Agree

2. **I found the system unnecessarily complex.**
   Mark only one oval.

	1	2	3	4	5	
Strongly Disagree	◯	◯	◯	◯	◯	Strongly Agree

3. **I thought the system was easy to use.**
   Mark only one oval.

	1	2	3	4	5	
Strongly Disagree	◯	◯	◯	◯	◯	Strongly Agree

4. **I think that I would need the support of a technical person to be able to use this system.**
   Mark only one oval.

	1	2	3	4	5	
Strongly Disagree	◯	◯	◯	◯	◯	Strongly Agree

5. **I found the various functions in this system were well integrated.**
   Mark only one oval.

	1	2	3	4	5	
Strongly Disagree	◯	◯	◯	◯	◯	Strongly Agree

6. **I thought there was too much inconsistency in this system.**
   Mark only one oval.

	1	2	3	4	5	
Strongly Disagree	◯	◯	◯	◯	◯	Strongly Agree

Figure D.1: System Usability Scale Survey used in each interview, Page 1

7. **I would imagine that most people would learn to use this system very quickly.**
Mark only one oval.

	1	2	3	4	5	
Strongly Disagree	◯	◯	◯	◯	◯	Strongly Agree

8. **I found the system very cumbersome to use.**
Mark only one oval.

	1	2	3	4	5	
Strongly Disagree	◯	◯	◯	◯	◯	Strongly Agree

9. **I felt very confident using the system.**
Mark only one oval.

	1	2	3	4	5	
Strongly Disagree	◯	◯	◯	◯	◯	Strongly Agree

10. **I needed to learn a lot of things before I could get going with this system.**
Mark only one oval.

	1	2	3	4	5	
Strongly Disagree	◯	◯	◯	◯	◯	Strongly Agree

Figure D.2: System Usability Scale Survey used in each interview, Page 2

E Wordlist Survey

Participant #: YourCompanyName

Step 1: Read over the following list of words. Considering the product you have just used, tick those words that best describe your experience with it. You can choose as many words as you wish.

- ☐ Simplistic
- ☐ Entertaining
- ☐ Impressive
- ☐ Friendly
- ☐ Ambiguous
- ☐ Relevant
- ☐ Fun
- ☐ Creative
- ☐ Compelling
- ☐ Accessible
- ☐ Cutting edge
- ☐ Cluttered
- ☐ Professional
- ☐ New
- ☐ Overwhelming
- ☐ Motivating
- ☐ Innovative
- ☐ Flexible
- ☐ Limited
- ☐ Fresh
- ☐ Clear
- ☐ Ordinary
- ☐ Comprehensive
- ☐ Annoying
- ☐ Busy
- ☐ Contradictory
- ☐ Stimulating
- ☐ Unpredictable
- ☐ Empowering
- ☐ System-oriented
- ☐ Complex
- ☐ Poor quality
- ☐ Vague
- ☐ Effortless
- ☐ Inadequate
- ☐ Exciting
- ☐ Time-consuming
- ☐ Trustworthy
- ☐ Satisfying
- ☐ Ineffective
- ☐ Unattractive
- ☐ Usable
- ☐ High quality
- ☐ Inconsistent
- ☐ Engaging
- ☐ Effective
- ☐ Desirable
- ☐ Illogical
- ☐ Confusing
- ☐ Business-like
- ☐ Time-saving
- ☐ Stressful
- ☐ Familiar
- ☐ Advanced
- ☐ Clean
- ☐ Intuitive
- ☐ Fast
- ☐ Attractive
- ☐ Incomprehensible
- ☐ Appealing
- ☐ Meaningful
- ☐ Unconventional
- ☐ Understandable
- ☐ Distracting
- ☐ Misleading
- ☐ Difficult
- ☐ Useful
- ☐ Irrelevant
- ☐ Unrefined
- ☐ Bright
- ☐ Approachable
- ☐ Boring
- ☐ Old
- ☐ Consistent
- ☐ Hard to Use
- ☐ Counter-intuitive
- ☐ Energetic
- ☐ Reliable
- ☐ Intimidating
- ☐ Credible
- ☐ Predictable
- ☐ Too technical
- ☐ Powerful
- ☐ Rigid
- ☐ Faulty
- ☐ Organised
- ☐ Convenient
- ☐ Simple
- ☐ Frustrating
- ☐ Obscure
- ☐ Dull
- ☐ Expected
- ☐ Slow
- ☐ Controllable
- ☐ Sophisticated
- ☐ Non-standard
- ☐ Straightforward
- ☐ Efficient
- ☐ Responsive
- ☐ Easy to use
- ☐ Awkward

Step 2: Now look at the words you have ticked.
Circle five of these words that you think are most descriptive of the product.

Figure E.1: Word list used in each interview

F Comparison

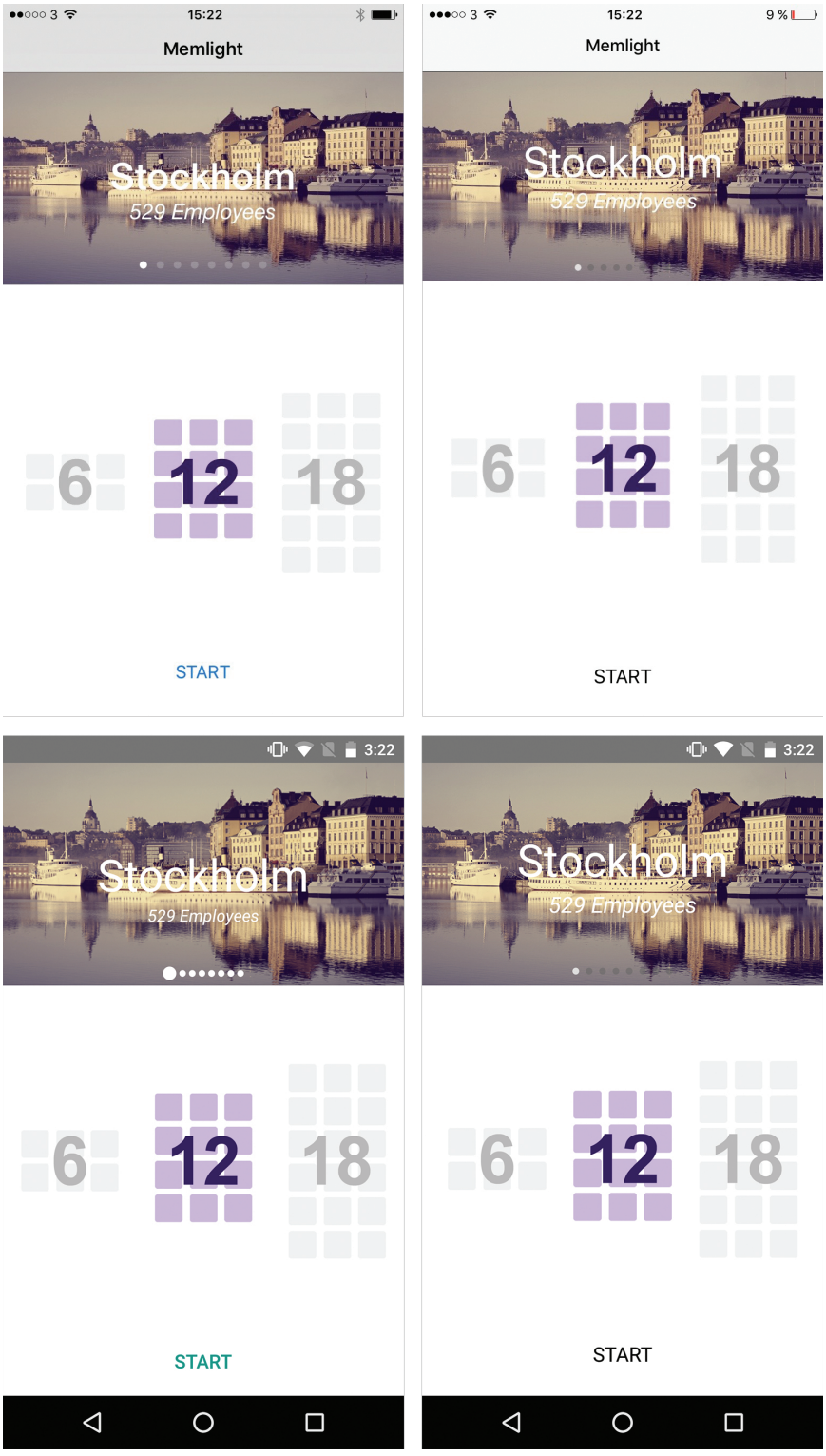


Figure F.1: Setup screen. Upper Left: iOS, Upper Right: React Native iOS, Lower Left: Android, Lower Right: React Native Android

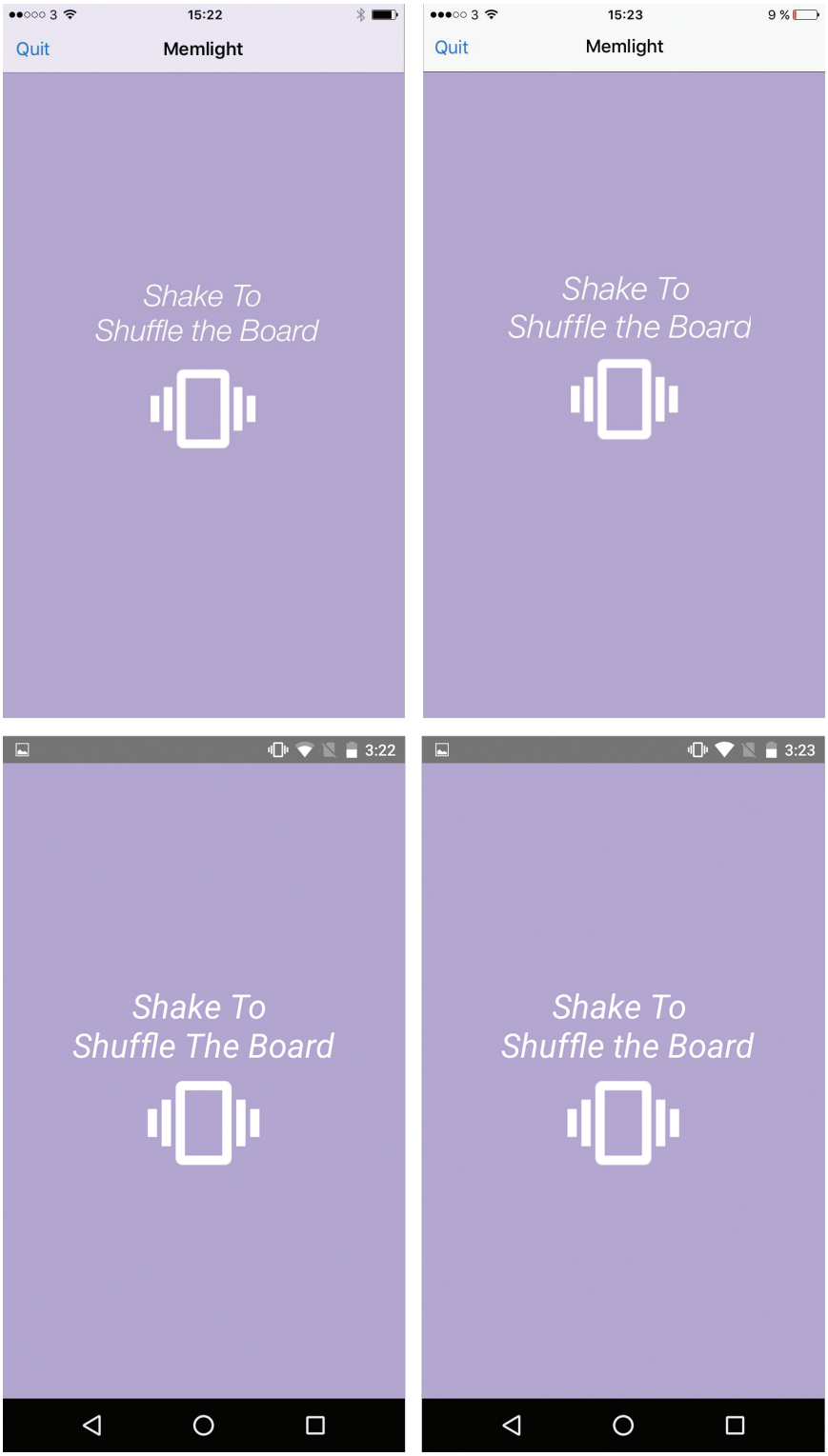


Figure F.2: Shake screen. Upper Left: iOS, Upper Right: React Native iOS, Lower Left: Android, Lower Right: React Native Android

Figure F.3: Game screen. Upper Left: iOS, Upper Right: React Native iOS, Lower Left: Android, Lower Right: React Native Android

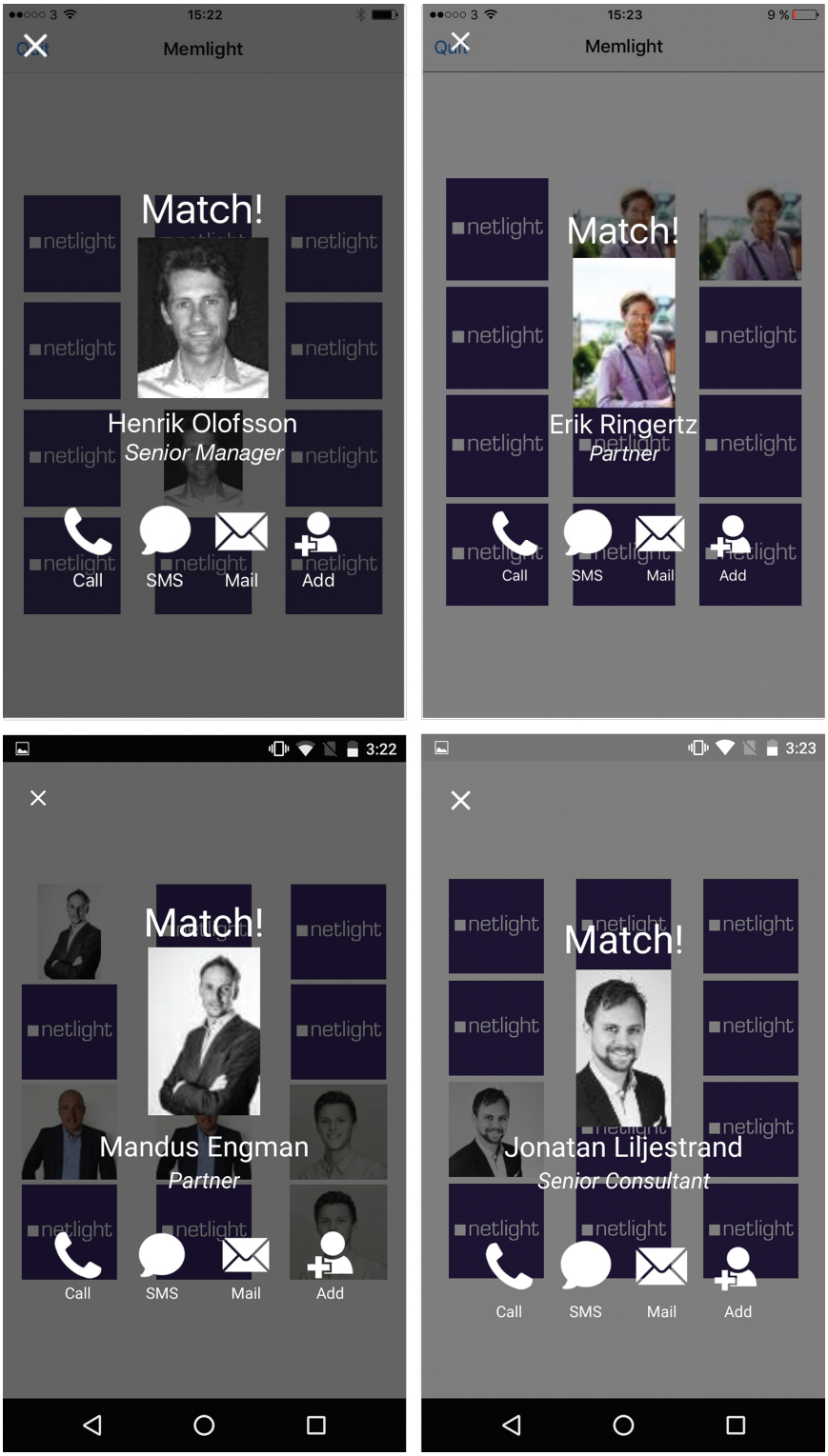


Figure F.4: Match screen. Upper Left: iOS, Upper Right: React Native iOS, Lower Left: Android, Lower Right: React Native Android

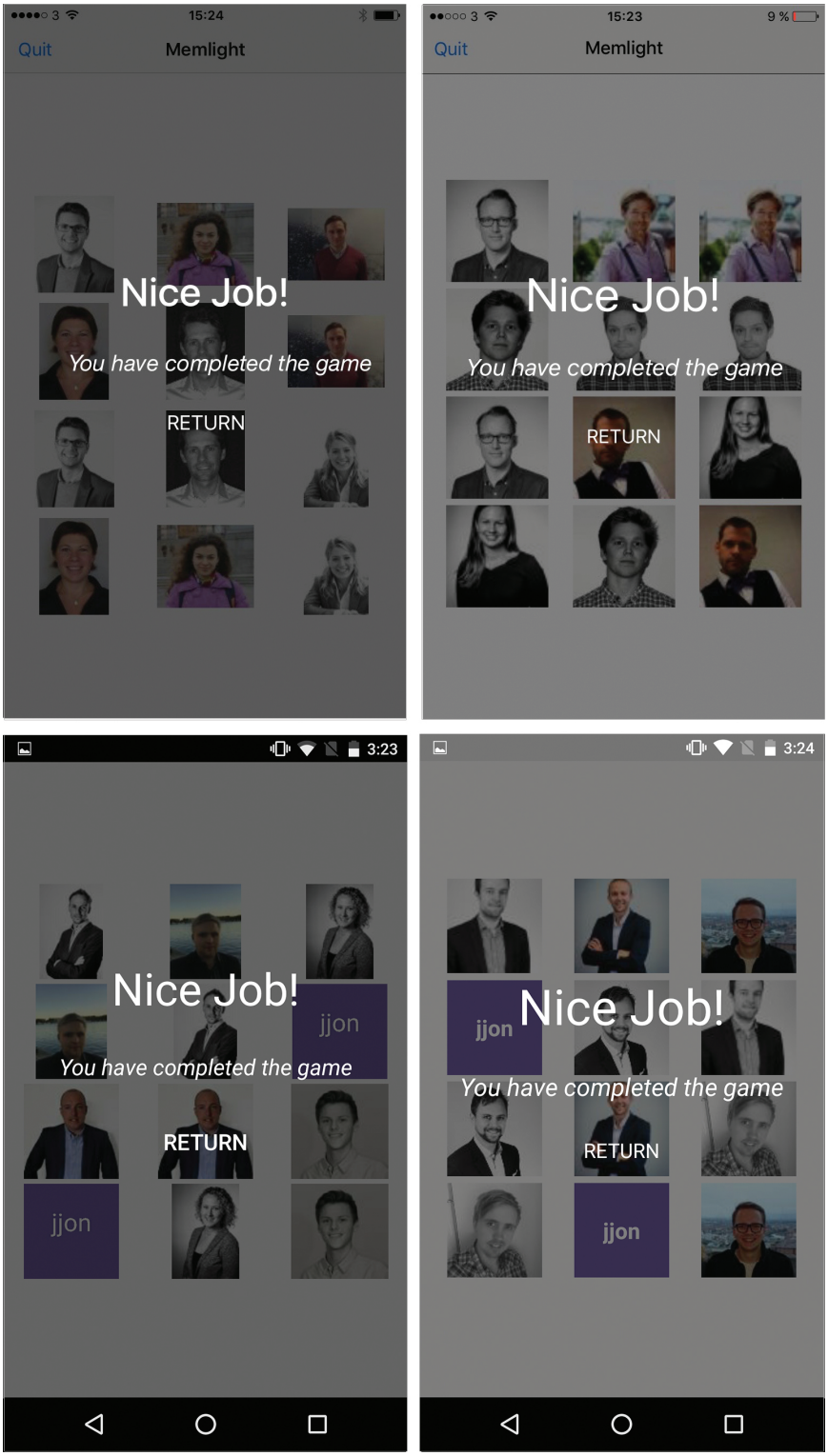


Figure F.5: Finish screen. Upper Left: iOS, Upper Right: React Native iOS, Lower Left: Android, Lower Right: React Native Android