Oliver Gill

Using React Native for mobile software development

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

English Report

Metropolia

Author Title Number of Pages Date	Oliver Gill Using React Native for mobile software development 34 pages 12th April 2018
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Mobile Solutions
Instructors	Zoltan Balogh, Line Manager Petri Vesikivi, Software Engineering Teacher

The topic chosen for this thesis is “Using the React Native framework for mobile software development”. The goal of this thesis is to demonstrate how React Native can be used to create a well-designed, multi-platform mobile application. React Native is a framework that is specifically designed for native mobile software development; however, it originates from React.js, which is used for the development of web applications. Although the programming languages are almost identical, the use of React Native allows for a more comprehensive application, which can integrate more with a mobile device.

Articles written about both React.js as a web development tool and mobile software development processes were used when writing this thesis. Having used these resources, the thesis is able to explain the benefits of using React as a language for mobile specific software development. React Native is a framework which has been gaining popularity since its release in 2015, because of its versatility as a programming language allowing for development on both mobile platforms, iOS and Android, whilst benefiting from native mobile application behaviours.

The thesis will document the processes and decisions made in the development of a React Native application for Nokia Solutions and Networks.

Keywords	Mobile, Software Development, JavaScript, React Native

Contents

List of Abbreviations

MVC	Model-View-Controller. An architectural pattern commonly used for developing user interfaces that divides an application into three interconnected parts.
DOM	Document Object Model. A cross-platform and language-independent application programming interface that treats an HTML document as a tree structure wherein each node is an object representing a part of the document.
IDE	Integrated Development Environment. A software application that provides programmers comprehensive facilities for software development.
NPM	Node Package Manager. A package manager for JavaScript.
UI	User Interface. The space where interactions between humans and machines occur.
API	Application Programming Interface. A set of subroutine definitions, protocols and tools for building application software.
CPU	Central Processing Unit. the electronic circuitry within a computer that carries out the instructions of a computer program by performing the basic arithmetic, logical, control and input/output (I/O) operations specified by the instructions.

1 Introduction

The aim of this thesis is to demonstrate how React Native can be used as a framework for mobile software development. This thesis will compare React Native against the existing native mobile frameworks. A demo application has been built, for Nokia Solutions & Networks, to present the feasibility of React Native in mobile software development as an alternative to native development. React Native provides a native framework in which well-designed applications for multiple mobile platforms, mainly iOS and Android, can be developed.

React Native is an extension of React, also written as ReactJS or React.js, which was released in 2015 and designed for native mobile software development. React Native is written with JavaScript, a well-known programming language, and aims to allow for use of native mobile functionalities and integration. [1.]

However, at its core React is essentially a library framework for the programming language JavaScript, which is used in web development. Facebook initially released React for general availability in 2013. Facebook developed React as an alternative to the standard MVC frameworks that were available at the time. Facebook set out to make a library that would be scalable and work in a more responsive manner to traditional directive driven programming languages. Pete Hunt states that React is designed to encourage the reusability of components that the developer creates. React utilizes JavaScript, as it is a flexible and powerful programming language, which is important when building large-scale applications. [1.]

Each component in React works separately from one another, allowing for minimal and efficient changes to the DOM. Another feature that Facebook developed for React is the use of JSX, an alternative to standard JavaScript Syntax when creating components. JSX shares similarities with HTML and HTML5, making it readable to software developers who are familiar with those coding languages. [1.]

Since 2013, React has been gaining popularity amongst software developers for its flexibility, reusability and scalability. These factors make it ideal for developing newer soft-

ware that is future proof and the code base is not wasted when developing newer projects. Facebook then went onto develop React further and released React Native in 2015. [2.]

React Native proves beneficial to developers or teams who are on a budget as the application can support both major platforms with the same code base. Kociecki writes that in his latest React Native project, the development time took 33% less than if they had been developing iOS and Android separately [3]. Development of a React Native application also does not require the need for native developers, who have a higher salary on average (See Figure 1).

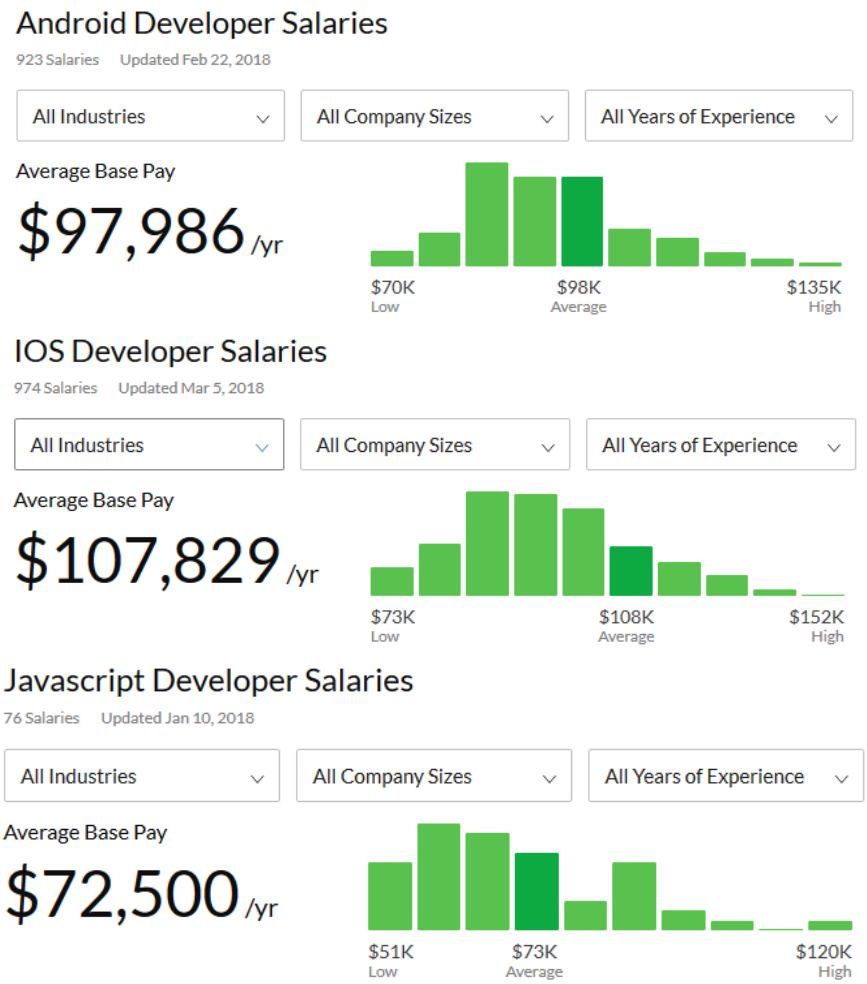

Figure 1. Comparison of JavaScript vs Native developer's salaries (US). Source: glassdoor.com

Most of the time, a project aimed to build a mobile application will require both a website and a native mobile application. React and React Native can be written by the same developer and both applications are able to share code as the applications are both written in JavaScript. With no requirement for a native platform expert, React Native development teams tend to be smaller and more manageable for a product owner or project manager. [3.]

Amongst languages that are used for web development, JavaScript continues to be one of the most popular languages. The community contribution to improving JavaScript is overwhelming, as can be seen in Figure 2, with over two million pull requests on GitHub in 2017. Each pull requests represents some form of contribution to the code that is on GitHub.

Figure 2. Most pull requests 2017. Copied from Most Popular and Influential Programming languages of 2018.

Part of this contribution went towards developing React Native even further. This can be seen by the efforts of software engineer Leland Richardson at Airbnb, who has been

working on developing a fully cross-platform version of React Native, that was named React Native Primitives. This library attempts to reduce the React Native framework just down to components that do not use platform-specific APIs. This library can be used to make code that is useable regardless whether it is for a web or native application.

With the ongoing efforts of JavaScript and React Native developers, the framework is constantly improving and proving itself to be an increasingly popular option for software engineers who have experience with JavaScript or React, and wish to develop native mobile applications.

2 Fundamentals of mobile software development

2.1 Smartphone industry

For many years now, the smartphone industry has been dominated by two main competitors, Android and Apple. As of 2016, these two companies make up more than 99% of all smartphone sales globally. Google purchased the Android operating system in 2005 from Android Inc. and although Android devices are distributed by many different manufacturers, they all use the same operating system. [4.]

Apple's mobile devices use the iOS operating system, which during its conception was based on their existing MacOS. This meant that many of the features that MacOS had during the early 2000s were incorporated into iOS, then known as OS X, making them feel familiar to iPhone users. Due to Apple manufacturing their own devices, Apple can create software which is targeted specifically for their devices. Whereas, Android has a harder time fine tuning their operating system and software updates to support multiple devices with different hardware capabilities. [5.]

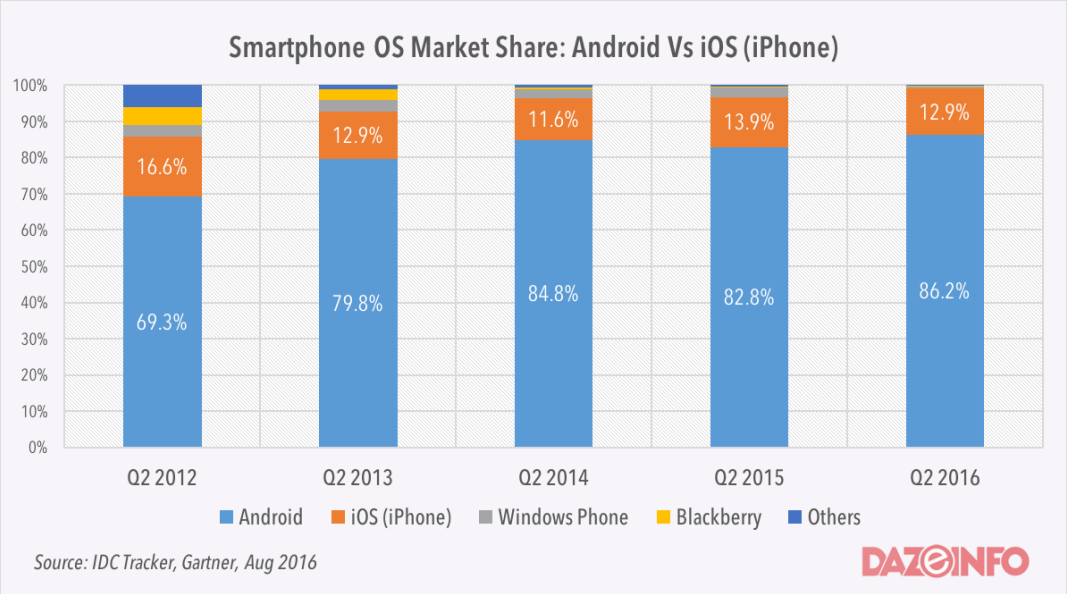

Figure 3. Smartphone OS Market Share. Copied from Apple Vs Android - A comparative study. (2017) [4].

Figure 3 demonstrates the increase in Android's total market share percentage from 2012 to 2016. However, the bar chart only represents a percentage of total smartphone users and does not account for the ever-increasing number of smartphone owners. Therefore, although it seems that Apple's market share percentage has decreased, the actual number of people that own Apple devices had risen within those four years. [4.]

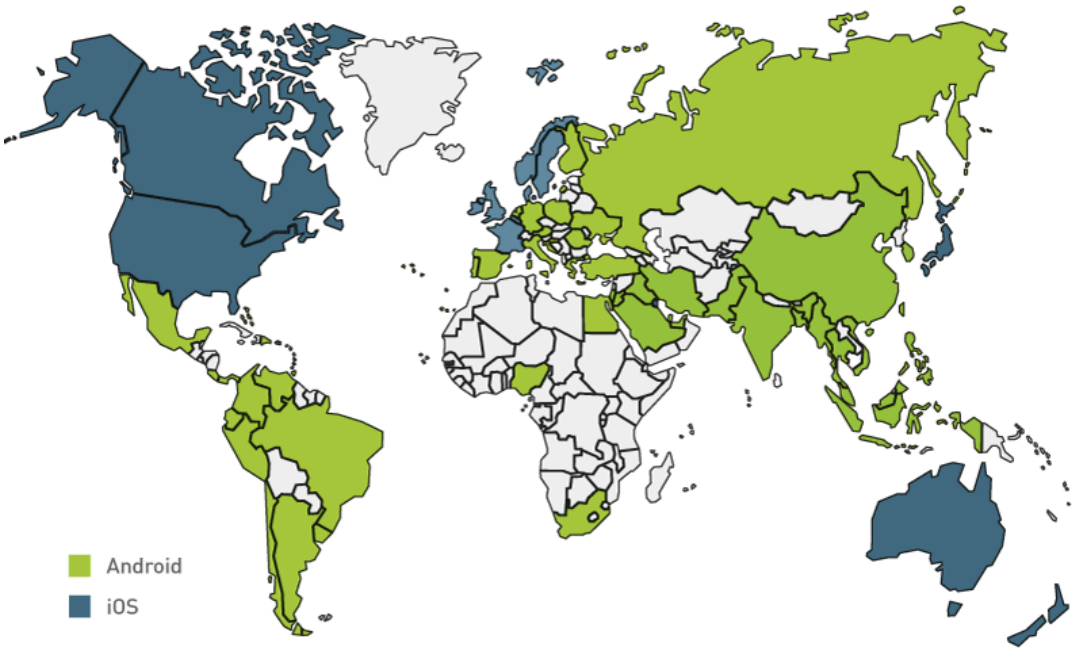

Figure 4. Geographical Distribution of Smartphones 2017. Copied from Apple Vs Android - A comparative study. (2017) [4].

Figure 4 highlights the different regions in which the majority of smartphone users either own an iOS or Android device. It is apparent that most of the First World countries tend to favour iOS devices, whereas developing countries prefer Android devices. This may be mainly due to the price difference in the devices, with iOS devices being on average significantly more expensive.

Figure 4 can also help explain the large difference between the total market share percentages, seen in Figure 3, as countries with extremely high populations including China, India and Russia seem to have a larger portion of people using an Android device.

With both Android and Apple devices becoming increasingly popular, it is more important for software developers to be able to support both types of devices. However, it can be expensive and time consuming for companies to try to create and maintain software applications which are individually coded for two different platforms.

2.2 Considerations before development

As the goal of the final year project is to develop a well-designed mobile application using React Native this section of the thesis will focus on the basic concepts and knowledge requirements for developing mobile applications. Canavesi highlights the importance of targeting the mobile platform in his article and he states, “Mobile marketing has become one of the most important, if not the most important, parts of just about any marketing strategy. People rely on their mobile devices for just about any activity imaginable and any company that is not part of this global trend seems to be out of touch.” [6.]

As the public relies more on their mobile devices, the industry continues to grow and a demand for businesses to have well-functioning applications increases. The mobile technology revolution has highlighted the importance of good usability in an application. This involves ensuring that the user has a pleasant experience while using the applications. User experience (UX) designers have become a core part of good application development, especially for mobile applications, as they are required to consider all the different possibilities in which a user may use the application most effectively.

In order to overcome this hurdle of understanding what the user requirements are, the application should be developed using agile software development. This technique provides the user and customer with an update to date understanding of the development process of the application. Agile as a development tool provides the programmers and designers with a lot of knowledge of what the user requirements are for the application.

The user interface designers of the application should take into consideration a lot about ergonomics whilst designing, as there are many key aspects to how to improve the user’s experience of the application. A well-designed user interface will position the most important, or most frequently used parts of the application towards the left side of the lower

half of the screen as most users are right-handed and use their right thumb when holding their mobile devices.

2.3 Mobile software development

As a developer, it is required to have a good channel of communication with the user interface designer as there may be times that the developer can become focussed on one certain way of implementation and not be able to consider other designs. However, the roles may be reversed and a developer could point out user experience improvements that could be made, as they are working with the application the most during development.

Mobile software engineers were restricted to developing in certain languages, which made the entry requirements to become a native application developer rather limited. The two largest mobile platforms, iOS and Android, have been competing against one another since the late 2000s and developers have struggled ensuring that their applications receive visibility on both platforms. With iOS applications programmed with Objective-C, and now Swift, and Android applications written in Java or Kotlin, it can take developers an inefficient amount of time to rewrite their application in an entirely different coding language that may behave significantly differently as well. Ideally, a developer would want to be able to write and maintain a single application for a universal platform. [5.]

For a solution, software engineers turned to web applications, as they could be created without the restriction of being limited to a single platform. Many frameworks for developing multi-platform applications began to arise, promoting the use of hybrid applications. Essentially a hybrid application is a web application that runs inside a chromeless browser window called WebView. Developers were able to take web development another step further, allowing a hybrid application to look and act almost indistinguishable from a native application. [7.]

Hybrid applications can be written with JavaScript, HTML5 and CSS which makes them appealing to create for web developers and entry-level developers. Due to the availability of developers with this particular skill set, hybrid applications tend to appeal to small and

medium sized businesses as multi-platform capabilities allow their application to reach a wider audience. [6.]

One of the main frameworks used for deploying hybrid web applications if Apache Cordova. This tool is an open source framework tool designed to aid in the deployment of a web application directly onto a mobile device. Using Cordova can benefit developers who wish to deploy their existing responsive web application directly onto a mobile device to mimic the behaviour of a native mobile application. The developer does not need to rewrite any code or learn any additional languages or frameworks to achieve this as the deployment instructions are present on the Cordova website.

There are however limitations to hybrid applications that are clearly visible to a developer. Some of these disadvantages can be observed when measuring the performance difference of a hybrid application against a native one, as hybrid applications add a layer between the source code and the target mobile platform via the mobile device's own browser. Hybrid applications tend to be unable to compete against native applications that are using newer device features as the hybrid applications lack the abilities to access all the variety of features that native applications are able to utilize during development. This is due to the dependency on the frameworks that the hybrid applications are built in. [6.]

In conclusion, hybrid applications have gained popularity with developers or businesses that wish to develop an application for multiple platforms. Hybrid applications provide a great alternative to native applications as they are cost and time efficient to create and maintain. They are however limited and may be unable to compete against the newer native applications, due to the hybrid application's reliance on a third-party framework.

Facebook founder Mark Zuckerberg, stated in an interview in 2012 "Our biggest mistake was betting too much on HTML5" when discussing Facebook's mobile strategy and they moved over to using a native application [6]. It was clear that Facebook felt restricted by the capabilities of a hybrid application so they instead began development on their own native framework that allowed them to fully utilize all the functionalities of the mobile device. This native framework was later to be known as React Native.

3 Developing with React Native

3.1 React architecture

React and React Native both use Flux architecture to handle the data flow within the application. This was created by Facebook as an alternative to the traditional MVC model, during the development of React. Flux varies from MVC by following the concept of a unidirectional data flow making it easier to locate any errors [8].

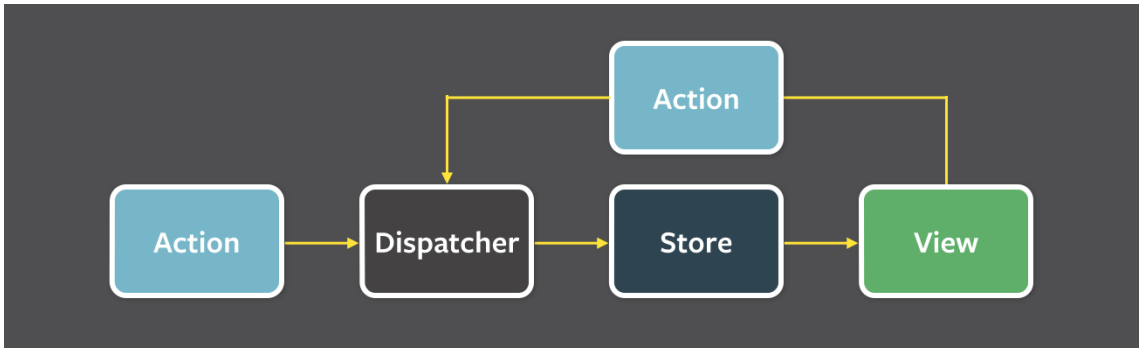

Figure 5. Data flow in a React application that uses Flux architecture. Copied from Flux, application architecture for building user interfaces [8].

The Flux data flow reduces the complexity of the MVC data flow by not using two-way bindings and requiring all actions to be passed to the centralized dispatcher (See Figure 5). Flux prevents cascading updates which often occur in the MVC model. Flux shares similarities with functional relative programming and flow-based programming. The single dispatcher handles all the changes to the different stores, within the data layer of the application. No extra actions are triggered until the store has finished updating with the data values. The View of the application is the re-rendered with the according changes.

3.2 Virtual DOM

Bray and Holmes [9, p. 15] write that within React and React Native, the UI is expressed as a function of the data within the application. Therefore, the developer does not need to be concerned with how the content of the UI will be rendered to the DOM. This is achieved by having an intermediate DOM within the React framework that is known as the Virtual DOM. By batching any required changes together, the Virtual DOM can compare the before and after states of the Virtual DOM and determine the minimum number

of real DOM manipulations that need to be made. Developers at Facebook then took this concept another step further by applying the same principal to a native mobile platform to create React Native.

3.3 Setting up a development environment

To get started with React Native, the developer may choose any IDE of their choice and then download Node, package management software, that allows the user to download additional software development tools and libraries. Once Node is installed, the developer should navigate to where they wish to create their application and use the following `create-react-native-app` command line utility: [10.]

```
npm install -g create-react-native-app
```

Listing 1. Command line code that will use NPM to install the react native library.

```
create-react-native-app <Insert Project Name>
```

Listing 2. Command line code that will create a React Native Project folder.

That is essentially all that is required to create the application and then it can be started with `npm start` from the command line, which will provide the developer with a QR code. The QR code can be scanned with the Expo client application on an iOS or Android phone as long as both the computer and phone are on the same wireless network.

3.4 Building projects with native code

React Native has the option to extend and advance the code base from existing native application. This is beneficial to existing native projects that wish to take advantage of the features React Native can offer.

To do this however, requires the use of a native development IDE, which in the case of an Android application is Android Studio. A few extra steps must be taken to configure

this correctly, which are dependent the on the operating system of the developer's computer. A full list of the different installation processes can be found in the official React Native documentation on Facebook's website. [10.]

Another benefit to building the project this way, is the additional use of an emulator, provided by the IDE, which can prove useful when developing without the need for a physical device.

3.5 A React Native application

This section of the thesis will demonstrate how to create a simple example application using React Native. If all the prerequisites have been followed, the application will now be running on the mobile phone. The following instructions will guide a developer on how to create a simple component for the application that can demonstrate the fundamentals of React.

Firstly, the developer should create a new file and according to correct naming conventions, name the file the same as the component class name. Afterwards, the developer should import React and Component from the react library, and then inherit from the Component class.

```
import React, { Component } from 'react';
import { Text, View } from 'react-native';

class WhyReactNativeIsSoGreat extends Component {
     render() {
          return (
               <View>
                    <Text> If you like React on the web, you'll like React
                    Native.
                    </Text>
                    <Text> You just use native components like 'View' and
                    'Text', instead of web components like 'div' and 'span'.
                    </Text>
               </View>
               );
          }
}
```

Listing 3. Code block that shows an example of how to create a basic React component. Copied from Facebook React Native [10].

The code shown above, demonstrates clearly how to structure a JavaScript file when writing React code. The render method on the fourth line is the main function that is called that will display the View component on the UI visually.

React components often have parameters by which they can be customized, these parameters are known as props. Theses props are set by the parent and are fixed throughout the lifetime of the component [10]. However, these props allow for the reusability of the same component in many ways. It is important to check the documentation of React Native components that are being used in an application to know what props they contain and how to customize the component properly.

React Native, unlike React, does not use existing standard HTML element tags, instead it has specifically built components that render in a similar way. The “View” component is very similar to the “ ” tag as they both act essentially as containers for their content. React Native also does not use individual CSS files for element styling, instead it uses a class called “StyleSheet”. This class can be imported from React Native and creates an object, which looks similar to a JSON object, that can be passed to the style prop of the element components as seen below in the renderFormItem function.

```
import React, { Component } from 'react';
import { Text, View, StyleSheet } from 'react-native';

class Form extends Component {
    constructor(props) {
        super(props);
    }

    renderFormItem(item) {
        return (
            <View key={item.name} style={styles.itemRow}>
                <Text>{item.name}</Text>
            </View>
        )
    }

    render() {
        const items = this.props.data.map(item => this.renderFormItem(item));

        return (
        <View>
            {items}
        </View>
        )
    }
}

const styles = StyleSheet.create({
    itemRow: {
        marginLeft: 15,
```

```
        marginBottom: 10,
    }
})
```

Listing 4. Code block to demonstrate how to style components in React Native.

The code block in Listing 4 demonstrates how the StyleSheet is used to add styling to the View component within the render method. The StyleSheet object should be instantiated outside of the class, so that it is accessible within all classes of that file. React Native allows for styling that is contained within the same file as the component, something which can make the styling of elements more straight forward to customize rather than using standard CSS.

Within React and React Native applications each component has two types of data that control it, props and state. The state of the component is known as the internal state, and it is initialized in the constructor of the component and then modified via the setState function call.

However, with the addition of the React-Redux library it is possible to have a global state contained in the redux store. This redux store is accessible from all parts of the application and provides good visibility on how the data within the application is structured.

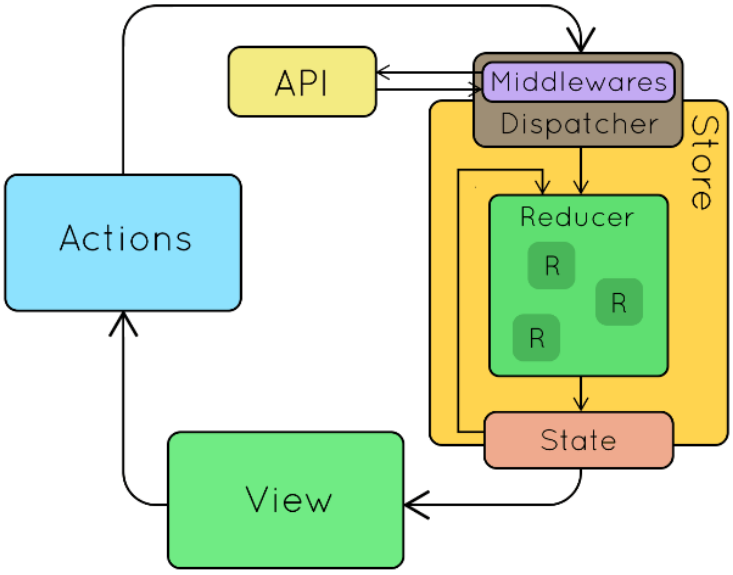

Figure 6. React-redux flow diagram. Copied from Redux from twitter hype to production [11].

When a change in the View (User interface) occurs, an action is dispatched. When actions are dispatched the values within the redux store will change (See Figure 6). Actions

contain a payload object which is handled by the reducer to determine how to update the state accordingly. This alteration to the state is then reflected in the view, usually with a visual change to the components.

React-Redux functions the same in both React and React Native which makes it an ideal choice when designing the architecture of an application that requires both a website and a mobile application as the actions, reducers and redux store can all be identical, further increasing the speed of the application's development.

3.6 Section summary

To summarize, React Native allows a developer to set up a simple working application with ease. It is a time efficient way of creating an application that can be shown directly onto the screen of a mobile device. This allows for the developer to see their changes update in real time on the mobile device, which infers that the user interface and user experience improvements can be tested effectively with other people.

A developer may also use React Native as an extension of an existing native project making it easy to support multiple platforms with a single code base. The tools within React Native allow for platform-specific features that are isolated from the other platform and will not interfere with one another.

4 React Native VS Native development

4.1 Performance experiment

This chapter aims to compare React Native to other forms of native development and investigate the differences. An experiment was conducted by Calderaio, which tested the memory usage of two different programming languages, Swift and React Native (see Figure 7). The purpose of the experiment was to discover which was the more memory efficient coding language and the tests were conducted with applications that looked and functioned identically. [12.]

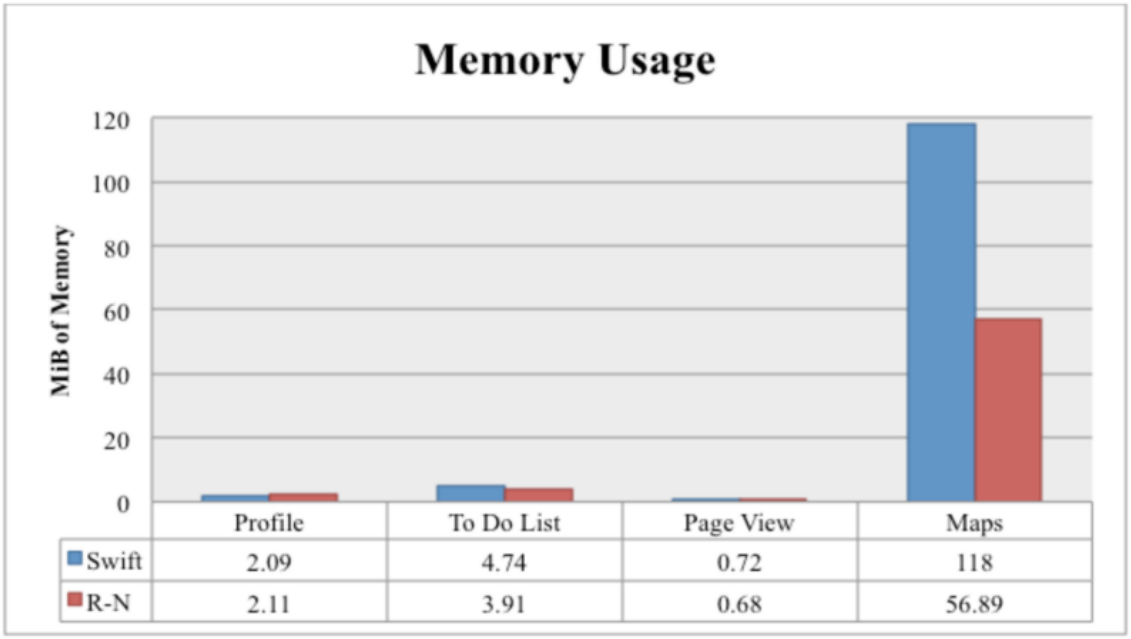

Figure 7. Swift vs React-Native Memory Usage. Copied from Comparing the Performance between Native iOS (Swift) and React-Native (2016) [12].

Within the first three views of the application the difference between the two coding languages were of a minimal percentage that would not be noticeable by a user. However, when moving onto the highly memory demanding view of the Maps it was clear that React-Native outshone Swift by using 61MB less memory than Swift. [12.]

React-Native has shown to be able to outperform Swift when it comes to memory usage with demanding processes as seen in Figure 6. The tests were performed in a fair manner with repeated results provided and the average has been represented as on the bar

chart in Figure 6. This is a surprising discovery as it shows that a widely used programming language such as JavaScript can be used to develop efficient applications for mobile devices that can even outperform native coding languages. [12.]

4.2 Disadvantages of React Native

Although React Native has many benefits to it, the framework is not without its shortcomings. The framework contains clear flaws that can make React Native difficult for some developers to use. Firstly, there is currently only a small amount of documentation available for the framework, with the majority of it being provided by Facebook via the framework's release notes. [13.]

Another issue that React Native has is that the tool is always adapting and changing to newer versions meaning that it is a high maintenance framework. A developer must be ensuring that they are able to know how to use current version of React Native that is required. [13.]

When writing a React Native application it is not always truly cross-platform as the same application cannot make use of both native functionalities and act as a standalone web application simultaneously. React also has a steep learning curve to it, with it being structured and written differently to a usual HTML and JavaScript web application. This learning curve may hinder the production of the application.

As React Native is able to utilize native components and functionality, in order to use it to its full potential, a developer must have pre-existing knowledge of native development on that platform. However, these disadvantages are all able to overcome by a developer who is committed to the application that they are developing. [13.]

5 React Native project - mobile workflow client

5.1 Initial requirements

The following chapter describes the development process of a React Native application. The application was built as a prototype for Nokia Solutions & Networks, as a concept for a mobile workflow client tool which will mainly be used by engineers and field technicians who work for telecommunication companies.

The mobile application is a demo application based on an existing React web application owned by Nokia. The original application was designed as a tool to be used by an engineer on a large monitor with full customizability to the user interface. The new mobile application should meet the same user needs as the existing application, so that any information that the user wishes to access within the original application would also be viewable in the new mobile application.

It was stated by a solution architect at Nokia that the application should act as a tool for field engineers to be able to efficiently update information regarding their manual tasks while performing a manual installation job. The field engineers are users who will be travelling to different locations to perform manual tasks, for example; travelling to a customer's house, to install, repair or update the customer's network devices and services.

5.2 Application design

After the initial specifications for the mobile application were presented, a few user interface designs for the mobile application were drawn as mock-ups. The mock-ups were based on an existing persona that was created by another UX designer for the possibility of having the application. A variety of mock-ups for each of the different application screens were drawn and presented to a UX designer for approval.

It was decided on with a UX designer that the application will follow the style guide set out by Nokia to better match their existing and upcoming future applications. This would give the feeling of unity between the different applications to allow for a more familiar

feeling to the user, with immediate recognition that the application was one created by Nokia.

A more in-depth design meeting, regarding the new application, was later conducted between the application developer and a UX designer from Nokia. It became apparent during the design meeting that it was not realistic to expect a mobile application to achieve all tasks that the existing application already could during the prototype stage. The meeting concluded with the UX designer proposing that the application should focus mainly on the field engineer's tasks whilst it was in the prototype phase. The application should make use of the existing mobile exclusive features to aid the field engineer in their manual tasks.

It was decided that it could be possible for the application to make use of the mobile device's camera to scan the barcode of the device being installed. The UX designer also suggested putting in a speech-to-text functionality if there was no barcode present to scan and instead a long device identification number. Both functionalities were implemented as part of the application to demonstrate the benefits of creating a native mobile application which take advantage of the mobile device's capabilities.

An example form that the field technician would fill out, was used as a guide for developing the dynamic form within the application. Due to the limited space on a smaller screen, it was decided to create as many dynamic or collapsible components as possible. This needed to be done whilst ensuring that all "pressable" elements were visually clear to the user. This is something that went through a few iterations of testing and design to finalize.

To better understand the role and tasks of the field technician, an imaginary situation was created to best replicate in detail what the field technician would do (See Figure 8). This helped when designing the work flow of the application as it was more easily understood what the technician needs the application for.

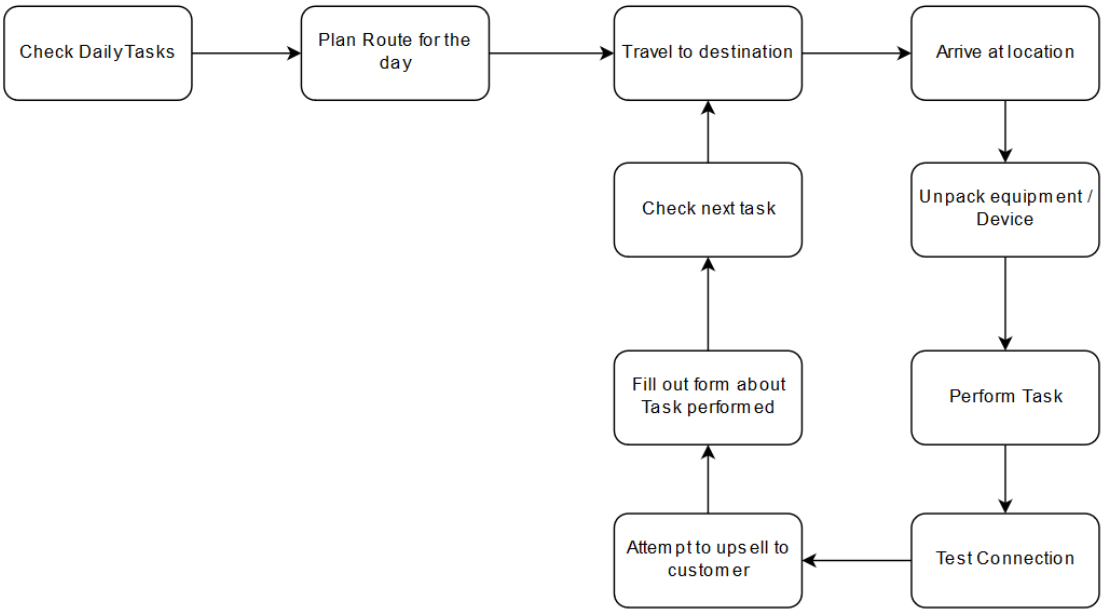

Figure 8. Conceptual workflow of a field technician's work day.

Due to the limitations of being unable to speak directly to the user, there was little understanding of how the end user would use the workflow application. Therefore, it was best to conceptualize an imaginary workflow that would make sense for when a field technician would be performing manual tasks.

Analysing the process of managing the tasks gave a better insight into what the field technician would require the tool for. It was suggested that the application should provide a form of navigation to the user. This would instruct the user on the optimal route to take during their day when performing the tasks. The navigation tool would place markers on each address and plan the route use the existing Google Maps API.

The user should also be notified of deadlines that they must uphold. This can be achieved by using push notifications. These push notifications alert the user of an upcoming event or task that they must perform at the given time and location.

5.3 Application implementation

Firstly, the development environment for the application was instantiated as shown in chapter 3.1 and the implementation began with the initial view of the application. This

view is the primary entry point of the application after the user would login to the application. As the application was developed without a server connection and would eventually require authentication, the user interface was developed as if the user was already logged in.

The goal of the project was to create a prototype, so there was no requirement for a connection to a backend or server, meaning all data that is displayed in the application is read from a text file which contains demo data, to simulate the connection to a database. It should be possible in the future versions of the application to create localization files, which would be changed according to the native language of the user or device.

It proved beneficial to focus on a single mobile platform during the development of the proof of concept application. It was chosen to target Android devices rather than iOS, as most of testing on physical devices could be done on Android. To develop the application for iOS devices, the developer requires access to a MacOS device. This was not a feasible choice, as no MacOS device was provided to the developer during the implementation phase of the application, which limited the amount of experimentation that could be done to demonstrate the cross-platform capabilities of React Native.

During the first stage of development, the layout of the application was decided on with a focus on ensuring navigation was consistent throughout. React Navigation is an official library within React Native that provides common stack navigation and tabbed navigation patterns for both iOS and Android [10].

React Native’s navigation mimics the behaviour of the native platforms by handling the device’s back button correctly on Android and making use of the navigation toolbar in iOS. In addition to using the React Navigation library, it was decided that the use of a side bar menu would increase the usability of the application, allowing the user to freely navigate between the main modules of the application.

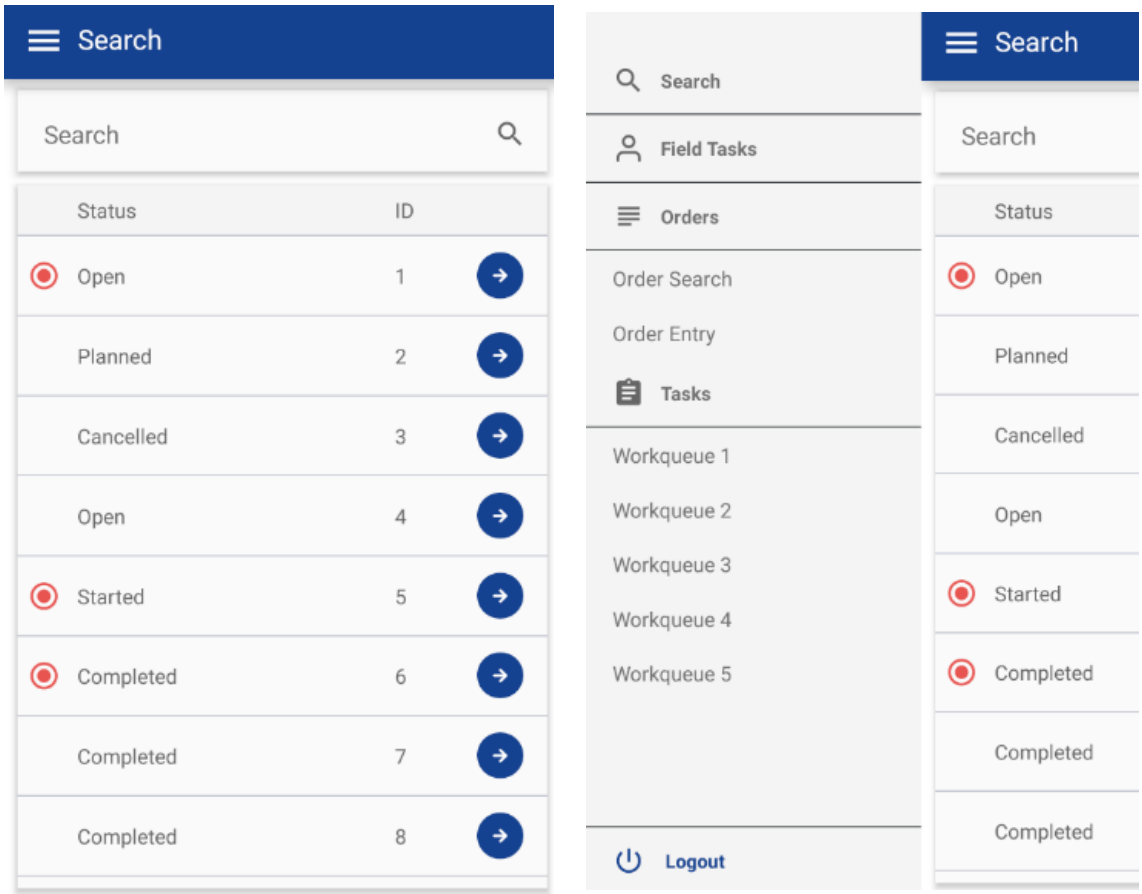

Figure 9. Search screen of demo application with collapsed (left) and opened (right) side bar menu.

Figure 9 displays the implemented “SideBarMenu” component, both in its collapsed and opened states. The side bar menu is opened by pressing the hamburger menu icon in the top left corner of the screen. The hamburger menu icon is a well-known icon used to represent menus or lists. This association of the icon will make it easier for users to know how to open the menu within the application. The navigation flow of the application is presented to the user so that they may find what they are looking for with ease.

Once expanded the side bar menu presents the user with four menu items and a logout button. The logout button is fixed at the bottom of the screen and will not move when the list is scrolled, separating it from the rest of the menu items, indicated by the thin border above it. The “Orders” and “Tasks” items within the list are expandable to show more items that belong to that section. Each item under the “Tasks” item can then be further expanded to reveal more items. The use of collapsible items and scrolling within a list, maximizes the amount of information that can be presented to the user on a smaller interface.

Within the demo application, the view which was developed the most was the Field Tasks view. This view was the main part of the demonstration Nokia was interested in, allowing them to see the potential React Native could bring when creating an application which was able to match their style guide and function as if it were an ordinary native application.

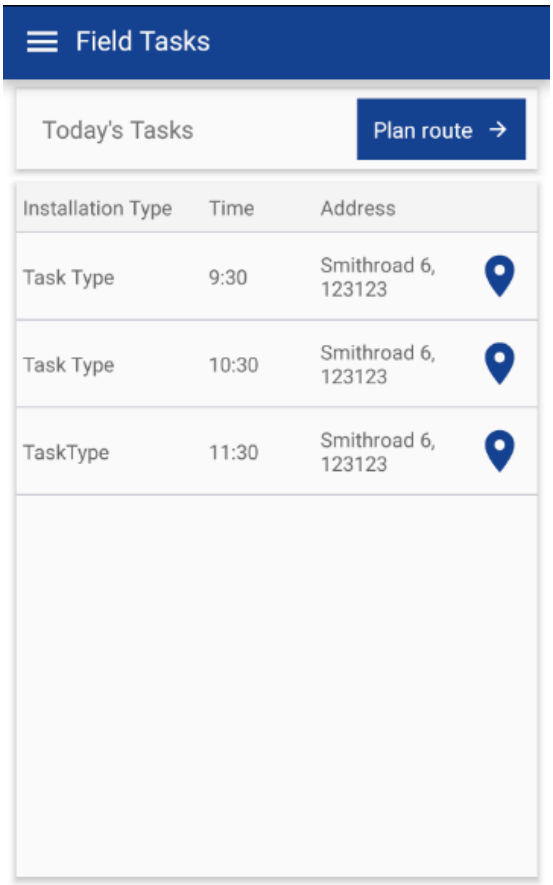

Figure 10. Field Tasks view within the application.

The above screenshot displays one of the first iterations of the user interface for the Field Tasks view. This part of the application displays the current logged in user's tasks for the day, sorted in a chronological order. This view presents the user with the most urgent tasks at the top of the list, displaying the type of task that must be performed, the expected time of the task and the address. The icon on the right of each item on the list will open a map view, displaying the location of the address and the current location of the user (See Figure 10). This map is created using the React Native maps library in addition to the Google Maps API.

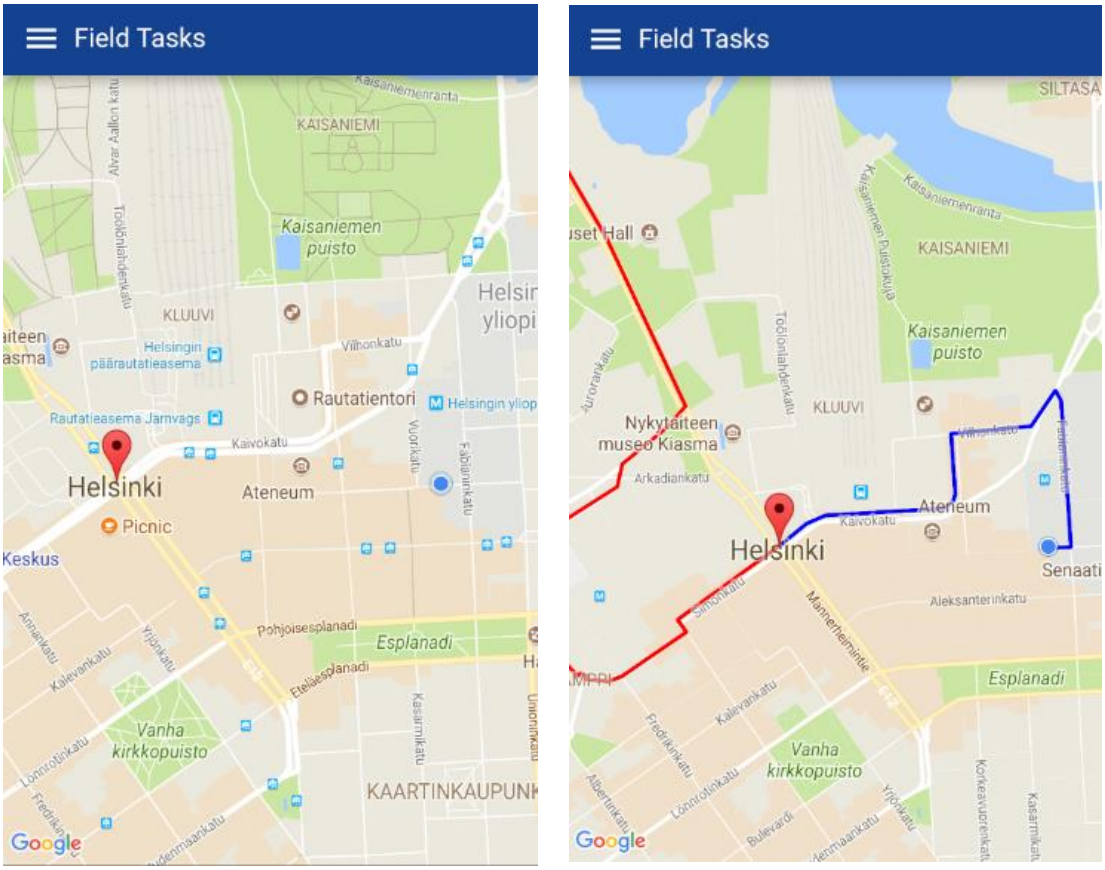

Figure 11. Map view displaying user's current location as a blue dot and destination represented by the marker.

The "Plan route" button will render a map that displays the optimal route the user should take to get to each destination starting from their current location (See Figure 11) The API will create routes, represented by the different coloured lines, between each of the tasks' addresses in the order of expected task completion. This feature will help the user navigate between the tasks by planning their route in advance. Within future implementations this feature could provide expected time of travel between the locations, using more of the APIs available.

Returning to the previous view where the field task list is shown, pressing on any of the task items will display the detailed task view, with the on-field task form. This field task detail view (See Figure 11) was designed to best replicate the contents of existing task forms that were presented by the solution architect at Nokia.

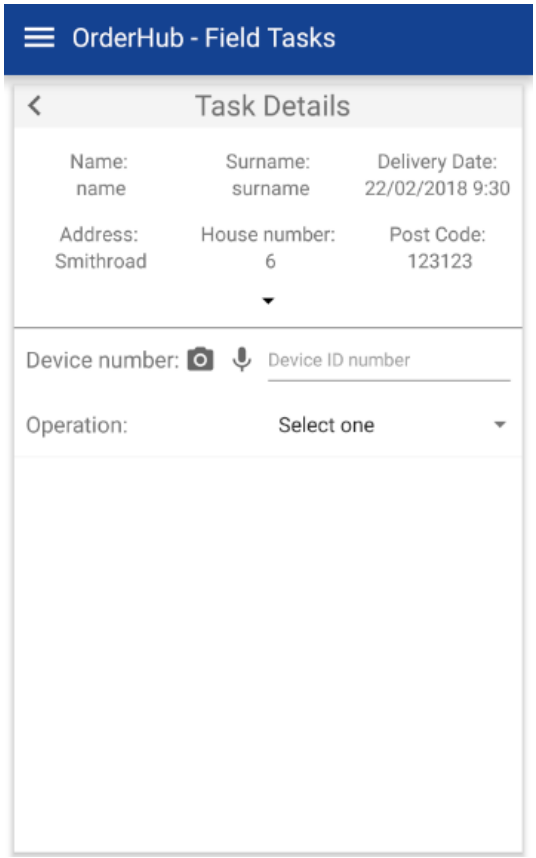

Figure 12. Initial design implementation of the on-field task detail view with task completion form.

Initially, the view was designed to display the required information about the task within a small ecard at the top of the screen, which could be expanded to show additional information, and the rest of the task form situated below the task details card.

To highlight the extra functionality created by a native mobile application the user is encouraged to make use of the different input methods for the installed device identification number. The icons next to the “Device number” label in the initial design both provide the user with additional ways of inputting the device ID. The user may scan a barcode on the device to retrieve the device serial number, or if no barcode is present, an option for vocally inputting the device ID is available too. The user may also just enter the number manually or edit the value once it has been input automatically.

The many available React Native libraries provided a straight forward solution to implementing this functionality. When implementing native mobile functionality, it is required to link the library to the specific platform of the device. This sometimes requires asking

for the device's permissions to access the hardware required. In the case of the barcode scanner, it required the permission to access the device's camera. The libraries mostly contain instructions on how the linking is done within the Android or iOS files within the source code of the project.

The dropdown selection labelled "Operation" is to determine how the task went during the installation. This then will dynamically render a different form depending on the selection of the operation type.

However, this design had flaws which were made apparent during the ongoing meetings with the UX designers. This design was also presented to a few other software developers at Nokia to determine their initial reaction on how to improve the layout of this view.

One of the most obvious design flaws in the initial design was that the icons were not recognized as buttons that could be pressed. This was altered in the next iteration of designs to display the icons within larger buttons, so it would be more clear to the user as to what the buttons would do (See Figure 13).

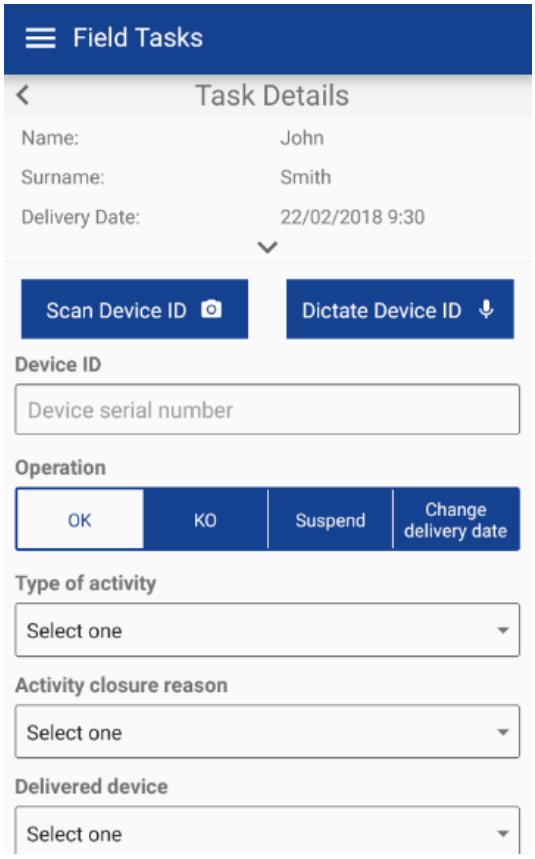

Figure 13. Finalized design of user interface for on-field task form.

Figure 12 displays the final implementation of the task form for the prototype application. The Nokia style guide has been followed when creating the colour scheme and input fields. The operation dropdown was altered to be a radio group component instead. The contrasting colours of the radio buttons highlights the currently selected option within the group, halving the number of clicks required for a selection as well.

If the number of items within the dropdown was 4 of less, it would make sense to use a radio group component instead. However, the contents of the other dropdowns within each operation category were still unknown when the design was finalized, therefore it did not make sense to the developer or the user experience designers to convert them to radio groups. This is something that should be improved in future versions of the prototype once more information regarding the form options is provided.

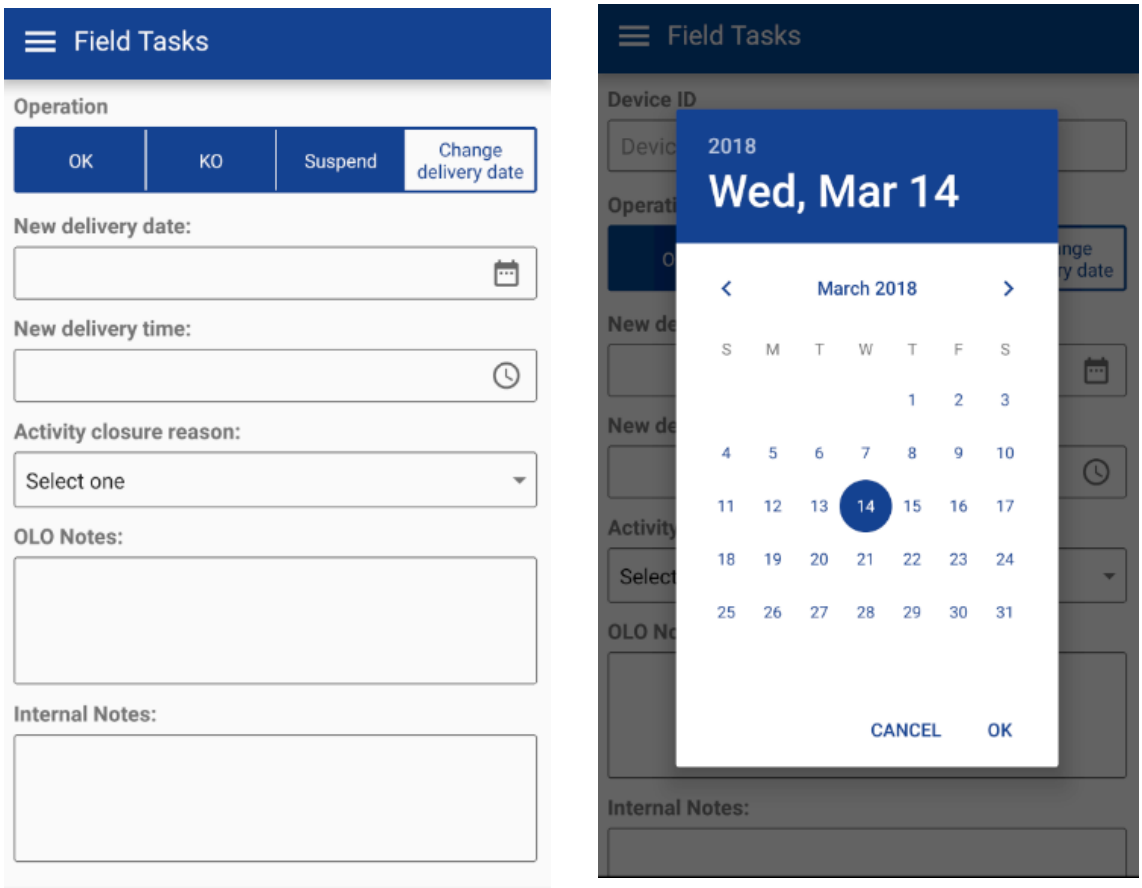

Figure 14. Change delivery date form when corresponding option selected in operation radio group (left). Native Android specific date picker component (right).

The “Change delivery date” option within the operation radio group renders a form that contains two input fields which utilize Android’s native time and date pickers. This native integration demonstrates how React Native can utilize the native features provided by the different device platforms. The code for these form components can be stored within a file with the extension “.android.js” or “.ios.js” and will only be used when the device is recognized as an Android or iOS device respectively.

This provides React Native with a large amount of flexibility when creating a platform--specific user interface and meeting the different user experience requirements that the different platforms have. Apple has set strict guideline rules on how their applications must look and many of the features may not work on Android, however in React Native’s case it does not matter as the features can be separated and used as if they were the same component within the form.

5.4 Application performance

React Native applications can run on mobile devices at 60 frames per second, providing a better user experience and visual feedback than hybrid applications which tend to run slower due to their dependency of the WebView layer of the device. Android Studio’s profiling tool was used to measure the performance of the prototype application (See Figure 15) while running it on an Android emulator.

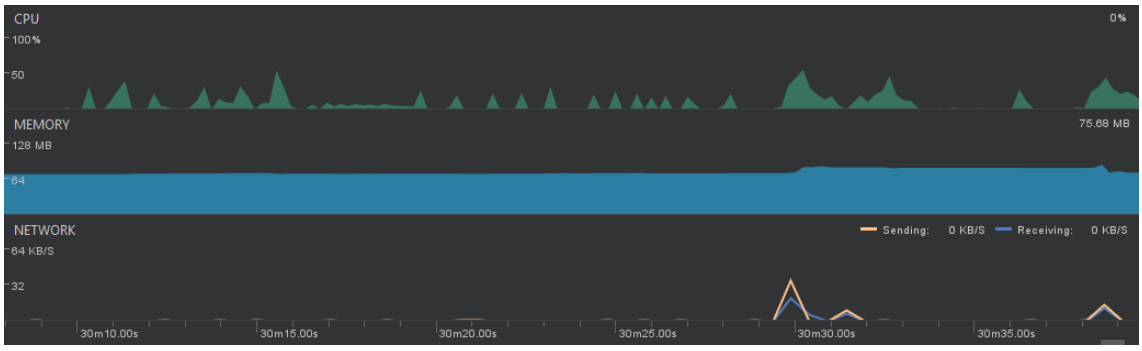

Figure 15. Android Studio’s Android Profiler tool displaying the results after measuring the performance of the prototype application.

The initial larger spikes in the CPU usage towards the left side of Figure 14, occurred when a new view was rendered. The CPU usage was at its highest when navigating and using the MapView component. When the map view screen was opened the only network requests within the profiling occurred because of the required connection to use the

Google Maps API. The raised profile of the memory usage displays the duration the map was kept open during the profiling.

Overall, the performance of the application seems to be acceptable and running well with little to no effect to the user interface. The only time that some noticeable performance issue occurs is when rendering the map. This could somehow be improved further to indicate that the view is not ready yet and should not be displayed until all required components have finished rendering.

5.5 Project summary

The development of the prototype application went well. With the combined contribution of the UX designers, the solution architect and the developer, the resulting application met the requirements that were initially set. The application was able to deliver a cross-platform solution to be used as a work flow client tool by field technicians who perform manual installation tasks. The tool allows the user to manage their daily schedule and fill out all the required information with ease.

The purpose for building this tool with React Native was to demonstrate how a realistic, functional prototype could be built efficiently by a developer who has pre-existing knowledge of React & JavaScript. A large portion of the frontend software developers at Nokia, are required to have this skillset as Nokia has chosen to use React as the tool used to build the majority of their products' user interfaces. This makes React Native a good choice for Nokia to use as their preferred method of implementation for a native mobile version of their existing or future applications.

Further improvements may be made to the prototype as it is taken forward as a proof of concept. One of these improvements may be made to the structure of the on-field task form, to reduce the amount of scrolling required within the form. One of these improvements should include finding a way to create an "Add notes" button and then allow the user to specify which type of note they wish to create. This would remove the need for three separate fields for notes within the task form.

Progressing forwards with the application, the user experience must be taken into consideration when adding more content and functionality to the application as the user interface may begin to feel crowded and overwhelming. Using React Native helps ensure that the user interface provides the user with the functionality that makes the tasks easier and more efficient.

6 Conclusion

React Native is a readily available framework with an efficient setup time that many web developers can feel comfortable with. React Native has shown to be a powerful tool that can compete with native development not only in terms of capabilities but also with its excellent performance on iOS and Android devices. Unlike other JavaScript frameworks, React Native aims to focus on creating reusable components for the UI, which is of the highest importance when developing a mobile application. The user experience of an application is vital to creating a successful application and this is achieved through creating a functional well-designed user interface. React Native can be easily integrated with existing applications and allows for developers to build on them even further by using React's large variety of features.

With the demand for mobile applications ever increasing, it is important for software developers to use their time efficiently. React Native allows the developers to be more efficient by not having to write duplicate code for multiple platforms and creates high reusability to different components that can be manipulated easily, which implies that the scalability of a React Native application is extremely profitable. Start-ups and smaller companies may favour the use of React Native as on average the developer salary is lower for JavaScript developers in comparison with native mobile developers for iOS and Android. Due to the lower salary, there is less risk involved in the project and also a larger number of JavaScript developers available to the smaller companies.

React Native offers a good solution for pre-existing React web applications, such as the ones created by Nokia Solutions & Networks, to be ported to a native mobile application while keeping most of the architectural code structure the same. This was demonstrated during the practical part of the final year project in which a React Native application was created based on a pre-existing application's architecture and design.

Although many hybrid applications tend to suffer from noticeable performance issues, React Native applications do not suffer from these same issues and can be built in a way that is identical in performance to a native application, as React Native applications compile from JavaScript into native code.

Some notable application names that are using React Native for their mobile applications include: Instagram, Netflix, Airbnb and Bloomberg [10]. This demonstrates that larger companies are able to adapt to working with React Native and consider it the best tool for their application's requirements.

With the community every growing and more developers being intrigued by the idea of a truly cross-platform language, React Native can flourish and prove itself to be a worthy consideration for native mobile application development. Whilst coding with React Native, it feels comfortable enough for a web developer to be able to manage the challenges that they may face.

Overall, React Native has proven that it should be considered as a development tool for applications that wish to be done efficiently, within a shorter time frame, when targeting multiple mobile platforms. The user interfaces that React Native can create, are identical to those that are available with native development, but the interface may be customized in a way to suit the requirement of the application. With developers at Facebook and the React Native community continually adding to the framework's development, React Native, seems to have a profitable future and should be a framework that developers consider using for their applications.

References

1 Hunt P. Why did we build React [online]? 5th June 2013. URL: https://reactjs.org/blog/2013/06/05/why-react.html. Accessed: 3rd November 2017

2 Korolova I. What are the real benefits of using React and React Native [online]? 9th February 2017. URL: https://aog.jobs/blog/what-are-the-real-benefits-of-using-react-and-react-native. Accessed: 3rd November 2017

3 Kociecki D. React Native can slash your mobile development costs by 30 percent [online]. 16th June 2017. URL: https://www.netguru.co/blog/react-native-can-slash-your-mobile-development-costs-by-30-percent. Accessed: 6th March 2018

4 Apple Vs Android - A comparative study 2017 [online]. 1st March 2017. URL: https://android.jlelse.eu/apple-vs-android-a-comparative-study-2017-c5799a0a1683. Accessed: 19th February 2018

5 Narayan A. React Native vs Native iOS/Android [online]. 15th November 2017. URL: https://www.coursereport.com/blog/so-you-want-to-build-a-mobile-app-react-native-vs-native-mobile. Accessed: 19th February 2018

6 Canavesi B. Benefits and disadvantages of hybrid mobile applications [online]. 15th May 2016. URL: http://brookscanavesi.com/uncategorized/benefits-disadvantages-hybrid-mobile-applications. Accessed: 3rd November 2017

7 Bristowe J. What is a hybrid mobile app [online]? 25th March 2015 URL: https://developer.telerik.com/featured/what-is-a-hybrid-mobile-app. Accessed: 3rd November 2017

8 Flux, application architecture for building user interfaces [online]. URL: https://facebook.github.io/flux/docs/in-depth-overview Accessed: 16th March 2018

9 Holmes E. Bray T. Getting Started with React native. Packt Publishing 2015.

10 Facebook React Native [online]. URL: https://facebook.github.io/react-native. Accessed: 3rd November 2017

11 Terpil I. Redux. From twitter hype to production [online]. URL: http://slides.com/jenyaterpil/redux-from-twitter-hype-to-production#/1 Accessed: 14th March 2018

12 Calderaio J. Comparing the performance between native iOS (Swift) and React-Native. [online]. Medium; 22nd February 2017. URL: https://medium.com/the-react-native-log/comparing-the-performance-between-native-ios-swift-and-react-native-7b5490d363e2. Accessed: 31st October 2017

13 Altexsoft. The Good and the Bad of ReactJS and React Native [online]. 20th May 2017 URL: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-reactjs-and-react-native. Accessed: 19th November 2017