Tampere University

Mika Kuitunen

CROSS-PLATFORM MOBILE APPLICATION DEVELOPMENT WITH REACT NATIVE

Faculty of Information Technology
and Communication Sciences
Bachelor's Thesis
February 2019

ABSTRACT

MIKA KUITUNEN: Cross-Platform Mobile Application Development with React Native
Tampere University of Technology
Bachelor's Thesis, 26 pages, 3 Appendix pages
December 2018
Bachelor's Degree Programme in Information Technology
Major: Software Engineering
Examiner: Tiina Schafeitel-Tähtinen
Keywords: cross-platform, multi-platform, mobile application, development framework, React Native

Developing and maintaining mobile applications for multiple platforms at the same time can be time consuming. Since the applications have to be developed with each platform's native development technologies, the software designers have to develop and maintain multiple separate source codes for one application. This has pushed many companies and communities to creating new development frameworks and tools for mobile applications which allow the developers to write and maintain a single codebase. The codebase is then compiled to native applications for each platform, and the applications can be published through the platforms' application marketplaces.

This thesis focuses on cross-platform mobile development technologies, specifically on the ones that utilize each platform's native functionality to deliver a familiar user experience to the end users, and React Native is then picked for a closer look. The purpose of this research is to find out if cross-platform technologies are a viable option for modern mobile application development in terms of both development experience as well as user experience.

The primary finding of the research was that while React Native and other cross-platform technologies may be a practical option for mobile application development, it is unclear when cross-platform technologies have a significant advantage over native development. In small applications the difference between the two is not pronounced and in more complex applications cross-platform technologies do not necessarily have many benefits compared to native technologies.

TIIVISTELMÄ

MIKA KUITUNEN: Alustariippumaton mobiilisovellusten kehitys React Nativella
Tampereen teknillinen yliopisto
Kandidaatintyö, 26 sivua, 3 liitesivua
Joulukuu 2018
Tieto- ja sähkötekniikan TkK koulutusohjelma
Pääaine: Ohjelmistotekniikka
Tarkastaja: Tiina Schafeitel-Tähtinen
Avainsanat: cross-platform, alustariippumaton, mobiilisovellus, kehityskehys, React Native

Yhden mobiilisovelluksen kehittäminen ja ylläpitäminen usealle eri alustalle voi olla hyvin aikaa vievää. Ohjelmistojen kehitys alustojen omilla kehitysteknologioilla pakottaa ohjelmistosuunnittelijan kehittämään ja ylläpitämään useita täysin erillisiä lähdekoodeja yhtä sovellusta varten. Tämä on saanut useat yhtiöt ja yhteisöt luomaan uusia kehitystyökaluja ja sovelluskehyksiä, jotka mahdollistavat sovelluksen kehittämisen ja ylläpitämisen usealle alustalle samasta lähdekoodista. Lähdekoodista käännetään jokaiselle alustalle natiivi mobiilisovellus, jonka voi julkaista alustojen omissa sovelluskaupoissa.

Tämä tutkimus keskittyy mobiilisovellusten kehitystä varten luotuihin alustariippumattomiin sovelluskehyksiin. Työ rajataan erityisesti sovelluskehyksiin, jotka hyödyntävät mahdollisimman paljon alustojen natiiveja ominaisuuksia käyttäjälle tutun käyttäjäkokemuksen luomiseksi. React Native valitaan teknologioista tarkempaa tutkimusta varten. Tutkimuksen tavoitteena on selvittää, onko alustariippumaton mobiilisovellusten kehitys järkevä vaihtoehto alustakohtaisille teknologioille modernissa sovelluskehityksessä.

Tutkimuksen tärkein löydös oli se, että React Native ja muut alustariippumattomat kehitysteknologiat voivat olla käytännöllinen vaihtoehto natiiville sovelluskehitykselle. Ei ole kuitenkaan selvää, milloin alustariippumaton teknologia tarjoaisi merkittävää etua natiiviin kehitykseen verrattuna. Yksinkertaisissa sovelluksissa näiden kahden välinen ero ei ole suuri, ja monimutkaisissa sovelluksissa alustariippumaton teknologia ei välttämättä tarjoa etua verrattuna natiiviin kehitykseen.

CONTENTS

LIST OF ABBREVIATIONS AND SYMBOLS

API	Application Programming Interface, a specification and implementation used to interface new software with other existing software
APK	Android Application Package, a package file used by Google to distribute Android applications
CSS	Cascading Style Sheets, a language for describing the style of a document written in HTML
GUI	Graphical User Interface, a category of user interfaces though the terms are sometimes used interchangeably
HTML	HyperText Markup Language, a markup language for creating websites and web applications
IPA	iOS App Store Package, a package file used by Apple to distribute iOS applications
JS	JavaScript, an interpreted programming language commonly used in web development
MVC	Model-View-Controller software design architecture
MVP	Minimum Viable Product, A product with the minimum amount of features to satisfy the requirements
MVVM	Model-View-ViewModel software design architecture
npm	Node Package Manager, a service for managing JavaScript libraries
SDK	Software Development Kit, a set of tools that allows the development of applications for a certain platform
UI	User Interface

1. INTRODUCTION

Mobile application market has become one of the largest branches of the software industry since its inception. According to reports from We Are Social and Hootsuite, in 2017, 3.7 billion unique users owned smartphones globally[1]. In 2017 52% of all web traffic came from smartphones[1] and another report by App Annie states that in 2017 the total number of mobile application downloads was over 175 billion[2]. The need for mobile application development continues to grow but, at the same time, the unique use-case for these applications poses several problems to the developers because each smartphone operating system is its own platform, independent from the others in terms of the application ecosystem, the intended user experience and the development technologies used. Since the potential user-base for most mobile applications covers multiple platforms, the developers have to develop separate applications for each platform, following the platforms' design guidelines and using their development technologies.

This requirement for expertise in multiple native development technologies and application architectures has created a need for development tools that simplify the development process of mobile applications. Since the main hindrance is having to develop a complete mobile application separately for each platform, new technologies have emerged that allow developers to share parts of the codebase between platforms. These so-called cross-platform technologies promise to solve the issues of developing native mobile applications, but currently there is no clear consensus on whether they fulfil that promise.

The purpose of this research is to determine whether React Native, and cross-platform mobile development frameworks in general, are a viable option for platform native development technologies. The mobile platforms included in this research are Google's Android and Apple's iOS. Other platforms do exist, but these two cover the vast majority of end users today[3]. Different approaches to mobile development will be introduced and compared through means of a literature review, and after that a small mobile application will be designed and developed with React Native. The development experience and the final application will be evaluated against similar situations in native development to see if any significant differences arise.

Prior research in this area has been done since 2011, but many past studies have focused on technologies that have very little in common with modern cross-platform development. More recent studies exist and are applicable to modern mobile development, but the development frameworks themselves change at a fast rate, so some of the results of past research have become outdated. Most notable studies from the past few years include Niclas Hansson and Tomas Vidhall's study on React Native[4], and Matias Martinez and Sylvain Lecomte's study on the quality of cross-platform applications[5].

Chapter 2 introduces mobile application development in general and Chapter 3 describes native mobile development in more detail. In Chapter 4 different options for cross-platform mobile development are introduced and compared with each other. Chapter 5 takes a closer look into React Native, a cross-platform framework, to better understand how the framework attempts to achieve native user experience regardless of the platform. In the following chapter React Native is first compared directly against native mobile application development through means of a literary review, and secondly through a small cross-platform application that will be developed. The comparison will be divided into two categories: development experience and user experience. Finally, Chapter 7 will end the study with conclusions from the development, the finished application and thoughts on the research question.

2. MOBILE APPLICATION DEVELOPMENT

Mobile development is a branch of software development that focuses on applications created for mobile devices. The importance of some features is more pronounced than in the desktop or web software development. Namely the power efficiency and the ability to work within the smartphones' hardware limitations, especially the small screen size, are important, but most of all the user experience becomes crucial.

Usability is, of course, important whenever developing software that interfaces with a human in some way, but with mobile applications the "look and feel" of the app is one of the biggest concerns in every project. Each platform has a design guide [6, 7] and mobile developers are encouraged to design the user experience of their apps following these guides as closely as possible. This results in applications that the user does not have to learn to use, but instead feel intuitive and familiar because the user has already learned the design of the platform while using the operating system and the pre-installed applications.

User experience is also one of the biggest differences between a mobile application and a modern web application. With the evolution of responsive web applications that fit into a screen of any size, the line between a mobile app and a web app has become less clear. Sometimes it can be a better option to serve the end user a web application than getting them to download an app from the platform's application marketplace. In some cases though, a mobile application has advantages over the web counterpart, and if the application requires the usage of hardware features such as camera or accelerometer, the support for those in web applications is not mature enough at this point.

The similarity between web and mobile applications has also led to some misconception about mobile development. Those not familiar with native mobile development may think that a mobile app only needs to be written once, and can then be released for all platforms. This is traditionally not true for mobile applications and even in the case of cross-platform development, not everything can be shared.

The spectrum of software, that you can use on a smartphone goes from fully custom-built and polished native applications all the way to generic responsive web applications. Cross-platform technologies are not all the same but most fall somewhere between the two extremes.

3. MOBILE DEVELOPMENT USING NATIVE TECHNOLOGIES

Native mobile application development refers to the traditional way of developing applications, using the platforms' native programming languages and software architectures to write codebases that are then compiled to truly native applications. While this method has been used since the inception of smartphones, the tools and technologies have evolved over the years. For Android the native programming language used to be Java or C++[6] but since 2017 Android has also supported the much younger Kotlin[8]. Objective-C was the only language for iOS development[7] until 2014, when Apple introduced their programming language Swift to the public[9].

Native app development forces developers not only to learn and use the native programming languages, but it also steers the developers to use the recommended software architecture of each platform. For iOS the recommended architecture is Model-View-Controller (MVC), while Android applications generally use Model-View-ViewModel (MVVM) architecture.[6, 7] As a consequence, the native codebases have very little in common with each other, and often knowing one codebase offers very little help in understanding the other.

The strongest argument for the power of native mobile development is user experience. Since developers are writing native code, they can utilize the native APIs (Application Programming Interfaces) and support libraries of the platforms, as well as create their user interface (UI) layouts using the native UI components provided by the platform. As a result, the sought-after native look and feel is relatively easy to achieve when using native development technologies.

The biggest issue with native application development is, of course, the separate codebases. Developing the same application for multiple platforms requires knowledge of the platform-specific development methods, design guides and technologies. Developing and maintaining native mobile applications for multiple platforms is comparable to running multiple individual software projects at the same time, so often more developers and other project personnel are required compared to a single

application project. This increases both the cost of the development as well as the maintenance as the number of supported platforms increases.

4. CROSS-PLATFORM MOBILE DEVELOPMENT TECHNOLOGIES

Cross-platform development in general refers to the act of developing platform-independent software. This kind of software can be run on multiple devices and operating systems and should function the same way regardless of the environment it is being run in.

Cross-platform development in a mobile context has existed for almost as long as smartphones themselves. The original iPhone was released in 2007[10] and the first cross-platform frameworks for mobile development were released in the early 2008[11]. However, most of the early cross-platform development technologies for mobile applications were different kinds of wrappers for web applications, allowing software that is written in HTML, CSS and JavaScript to be run as an individual application.[11] This approach has many disadvantages which will be discussed in more detail in Section 4.1.

Later on more modern development technologies started to emerge, allowing developers to create cross-platform mobile applications which resembled their native counterparts much better. The need for a native user experience is just as strong in cross-platform applications as native applications, so the developers still need to be familiar with the platforms they are working with. An argument could be made that cross-platform developers do not need such a deep knowledge of each platform as native developers, but some platform-specific development has to be done if the goal is to obtain a native user experience.

Cross-platform applications can be categorized in different ways based on how they were developed. One study on the approaches of cross-platform development, published in 2012, divides cross-platform mobile applications into four primary categories: mobile web applications, hybrid applications, cross-compiled applications, and interpreted applications[12]. These categories were further refined in the thesis by Niclas Hansson and Tomas Vidhall in 2016[4]. Mobile web applications were already discussed in Chapter 2 and will not be included in this chapter. Developent frameworks using the latter three approaches will be examined in more detail. It is

worth noting that native scripting frameworks is the modern term for interpreted frameworks as the term is more descriptive[4].

4.1 Hybrid Development Frameworks

Hybrid frameworks are a collection of web-based development technologies. The major difference between a pure web application and a hybrid application is that while a web application will always be used via a web browser, the hybrid frameworks produce applications which are always installed on the smartphone.[12]

The features and user interface of a hybrid application are developed with HTML, CSS and JavaScript, much like any web application. The hybrid framework then wraps the web application into a container that creates an installable application for the different mobile platforms. Some hybrid frameworks act as a very thin wrapper for the web application, offering no access to hardware features such as camera or geolocation. This was especially true for the earliest cross-platform frameworks[11] but most of the modern options allow the developer to utilize the devices' hardware features as well.

Almost all hybrid frameworks, such as PhoneGap, Apache Cordova and Ionic, have the same drawback: the user experience. A mobile application is supposed to look and feel native to the platform but this goal is extremely difficult to achieve when ultimately you are developing a web application. Separate user interfaces could certainly be developed for different platforms but this completely defeats the purpose of cross-platform development. Code reuse becomes marginal and the performance of the application will always be worse than a native application.[12]

Hybrid frameworks have not changed much throughout the years and still suffer from the same disadvantages as they did in 2012. Since the user experience of the application does not match the expectation set by the distribution channel, applications that are developed with hybrid frameworks are often considered inferior to pure web applications or more native approaches.[12, 4]

4.2 Cross-Compiled Native Frameworks

Cross-compiled native frameworks allow developers to write source code in a common programming language, which is then compiled to native binaries for each platform using a cross-compiler. The cross-compiler's responsibility is to depict which native user interface components and native platform features need to be used.[12] An

application developed with a cross-compiled framework can be very close to a fully native mobile application, but exceptions do exist.

The main advantages of cross-compiled frameworks are performance and user experience. The user interface is rendered on each device using native components, so the application will look native as long as the developers take platform design guides into account. While no cross-platform framework can quite match the performance of fully native applications, performance tests show that cross-compiled frameworks achieve adequate performance in most situations and the application will feel responsive.[13, 14]

Examples of cross-compiled frameworks are Microsoft's Xamarin and Xamarin.Forms and Google's Flutter[15, 16]. Xamarin and Xamarin.Forms applications are written in C# and a combination of C# and XAML respectively. Flutter applications are written in Dart. Flutter is also an exception to other cross-compiled frameworks in terms of the implementation of the user interface.

Flutter is a relatively new cross-platform framework with its initial release in 2017, and it has just reached the first stable release of Flutter 1.0[17, 18]. The notable difference between it and other cross-compiled frameworks is that Flutter does not use any native user interface components to render its views. Instead it has a library of widgets implemented by the Flutter team[16]. This allows Flutter to abstract the components in the same way regardless of the platform since it is not relying on platform-specific implementation and architecture. However, the drawback to this approach is that the developers have to rely on the maintainers of Flutter to keep all of their UI widgets up to date with the native counterparts.

Cross-compiled frameworks have matured over the years and some of the issues noted in the studies made around 2012 no longer exist or have been mitigated to a sufficient degree. The level of user interface abstraction has increased, and as such, code reuse in user interfaces has become easier. Some native platform features have been abstracted enough to allow code reuse as well, but more intricate features still require platform-specific implementation.

4.3 Native Scripting Frameworks

Native scripting frameworks utilize an interpreter to execute written code during runtime. Any scripting language can be used but most commonly modern frameworks use a variant of JavaScript, the most popular language for developing web applications.[4] Some examples of modern native scripting frameworks are Native-

Script and Facebook's React Native, which this study focuses on.

Native scripting applications use the scripting language for their business logic and user interface design but utilize the native platform user interface components for rendering the application, and call native platform APIs for all possible features. The advantages of this approach are similar to the advantages of cross-compiled frameworks: the user interface will look native to the platform if designed properly, and the performance of the application is very close to a native application. This results in a user experience very close to that of native applications without the need for a deep understanding of the platforms themselves.

The biggest drawbacks of native scripting frameworks are the scripting language used and the implementation of native features. JavaScript and many other scripting languages are not ideally suited for all use-cases due to their inherent features as scripting languages, namely the dynamic characteristics and weak type definitions. This will often lead to mistakes in code only being noticed at runtime, though some of these issues can be solved using additional support libraries created for the languages.

5. REACT NATIVE

React Native is a native scripting framework for cross-platform mobile development. The framework was originally developed by Facebook and released in 2015 for iOS only, but an active community has since added Android support and done many other contributions to the open source project.[19]

React Native applications are developed using JavaScript, more precisely EcmaScript 2015 (ES2015 or ES6)[20]. EcmaScript is a superset of JavaScript, adding numerous features to JavaScript and fixing some of the mistakes originally made in the JavaScript specification. EcmaScript has also replaced pure JavaScript in modern web applications, and all modern browsers support some version of it.[21] In addition to ES2015, React Native implements parts of the next version of EcmaScript, ES2016 or ES7[20].

5.1 Architecture

The React Native application architecture consists of a JavaScript virtual machine, the React Native bridge and native modules. This architecture is visualized in Figure 5.1. The application code runs on the JS virtual machine along with any third party libraries used. The native module calls are routed through React Native's bridge to the native APIs and the third party libraries, and the results are passed back through the bridge if needed. This allows the use of JavaScript to develop the application while still utilizing the native UI components and features of the platforms.[19]

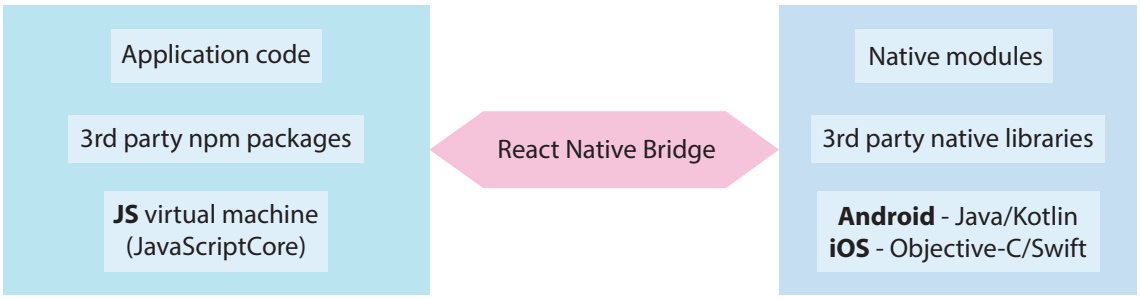


Figure 5.1 *React Native application architecture*

It is worth noting that React Native is undergoing an architecture change at the time of writing this thesis since React Native's new Fabric architecture is under development and will likely be released in 2019.[22]

The JavaScript application architecture is based on Facebook's React, a component-based framework for building interfaces for web applications. The basic element of a React application is a component, which is an individual JavaScript class that always renders a part of the user interface. The component may also receive properties (known as props in React) and have an internal state. Multiple components are then composed to create more complex user interfaces and data can be passed down the component chain as props, from parent to child.[19]

Mutating data in a React application can be quite challenging, as the data flow is unidirectional and the data needs to be mutated only at the highest level state where it exists. The updated data then needs to be passed down as props to children components. In pure React applications this is achieved through passing callback functions to the children component as props. Then the child component can call a callback function to mutate the data. As the parent component that has the highest level state receives the callback, it will mutate the data and the updated data is passed down to the children components automatically.[23] Other solutions exist in the form of Node Modules provided by the Node Package Manager (npm). One of the most popular Node Modules for handling application state and data mutation is Redux, which provides a centralized state that any component can access and call mutating functions on[24].

Neither React or React Native offer any solution to navigating between different views in the application. Again, multiple Node Modules exist to solve navigation in different ways but one of the most common is React Navigation[25]. React Navigation offers different types of navigation, such as stack navigation or tab navigation, for different situations.

5.2 Native Features

Most of the common native features are already implemented in React Native and can be called from the JavaScript code directly. Sometimes though the developers need to implement a more complex feature than the React Native APIs can provide, such as accelerometer tracking in a background process. In these situations React Native allows developers to implement their own native functionality in much the same way as React Native's own implementation. A module is created with a JavaScript interface that can be called within the application code. Platform specific code is written for the module in the native programming language of each platform, and React Native bridges the calls from JavaScript to the native modules.

6. COMPARING REACT NATIVE TO NATIVE DEVELOPMENT

Due to the limited scope of this thesis a fully fledged comparison between React Native and native development could not be made. Instead a small mobile application was designed and developed with React Native. The development experience of React Native and possible issues that occurred during the development is compared to how those situations would have been solved in a native mobile application, or whether the same issues would have existed at all.

The user experience of a React Native application is assessed both through the test application as well as prior studies. The test application developed for this thesis is fairly simple so no dramatic differences in the usability or performance would be found compared to a native application.

6.1 Designing and Developing the Test Application

The concept for the test application was an application that lets users scan barcodes from books, searches an online database for information about the book, converts the information to a formatted citation, and lets the user share the citation as text through whatever channel they find most convenient.

The basic requirements for the application were simple. When a user opens the application, on the first startup the user is asked for permission to use the device's camera. The user can then navigate to the scanning view where a barcode can be placed in the camera's view using the camera viewfinder. Upon scanning a barcode, the application lets the user confirm that the barcode is correct. If it is, the application navigates to a searching view and starts searching the online database for results matching the barcode. The searching view shows the status of the search and the eventual results, allowing the user to choose which of the results was correct. Once the user chooses one of the results, the application navigates to a sharing view where it creates a formatted citation and allows the user to share it via the Share API of the platform. The full navigation flow of the application can

be seen in Appendix 1.

Some restrictions were set for the application before its development:

- The citation is formatted in BibTeX format which is supported by most reference management applications.
- The user is not allowed to edit the citation in the application because either a very complex editing form would have to be developed, or the application would run the risk of letting the user break the citation formatting.
- The online database used is Google Books, used via the Google Books API, because it is the most comprehensive free book database that has, what appeared to be, a suitable API.

The main technology used for developing the application is React Native, but most often developers add multiple third party libraries and tools to compliment React Native and make the development easier. In this application the technology stack was as follows:

- React Native for most of the application
- Redux library for controlling the state and passing the data and actions as props to the components
- React Navigation library for creating the navigation flow
- Expo barcode scanner library
- Expo development tools for testing in a development environment
- TypeScript language and compiler for adding static type checking to JavaScript code

The source code developed with this technology stack is far from pure React Native but is also more indicative of a real world example of development with React Native. Especially once the data that the application handles becomes non-trivial, many of these libraries become crucial to avoiding mistakes in development.

The application development itself was done using agile methodology: small modules of the application were developed one at a time, first to a Minimum Viable

Product (MVP), and then expanded as other modules of the application required more features from them. As React Native facilitates very fast development and test iterations, the prototyping was continuous, but the development process can be divided into five larger milestones, and into corresponding prototypes which build on the previous versions, listed below:

1. The main view with camera permission checking, navigating to the scanner view and scanning a barcode
2. Scanning a barcode and letting the user confirm it, searching the Google Books API
3. Searching view and navigating to it from the scanner view, searching the Google Books API for the barcode, letting the user choose which result to use, and creating the citation
4. Citation view and navigating to it, implementing the Share API, styling the user interface for Android, and the first release build for Android
5. Customizing some styles for iOS, adding some graphics and finalizing the application as far as this thesis is concerned, and a second release build for Android. Release 1.0.

Android versions of the application were tested on a device and APK (Android Application Package) packages were built to install the application on multiple devices. The iOS versions were tested in emulators but no test device was available. No IPA (iOS App Store Package) package was built either because Apple requires a paid subscription to their developer program to be able to build and sign IPA packages[26].

The Google Books API turned out to be more difficult to work with than what it initially seemed like. The major drawback of the API is that it does not guarantee the existence of any of the information vital to the mobile application's functionality. For example, the author information of some books did not exist in the data returned by the API even though the corresponding Google Books web page clearly showed it.[27] This meant the information from the API could not be relied on for creating the citations, and instead the citations had to be fetched from a separate API that Google uses in their own Books service for exporting citations. All issues with Google Books could have been avoided by switching to a more suitable API like the one provided by ISBNdb[28], but unfortunately no freely available APIs seemed any better.

The source code of the application is published as an open source GitHub repository[29] and the state of the code at the end of this thesis is marked by the 1.0 release tag[30].

6.2 Development Experience

It is very easy to set up the development environment using Expo's tools, and it is equally easy to get the development version of the application to a test device or an emulator. Getting the initial "Hello world" application running is a quick process but installing all of the supporting npm libraries and getting them to work in the application takes much more effort.

Developing an application with React Native is characterized by fast prototyping. It seems like many of the advantages of the framework are centered around making the testing of an updated application easier and faster. One of the major selling points of React Native and many other native scripting frameworks is live reloading, also known as hot reloading. During the development of the application, a device or an emulator is connected to a development server running the application. Whenever one of the JavaScript source code files is changed, the development server will rebundle the JavaScript code and send it to the device, and the device reloads the application with the updated code immediately. This reduces the amount of time spent on waiting for a build to complete in order to test a new change, but it also encourages the developer towards using a trial-and-error development style where the application is rebuilt after every small change. This can be a good or a bad effect of the feature.

Facebook and many other cross-platform technology developers market their frameworks with a slogan that can be summarized as "learn once, write everywhere". This means the developer should only need the knowledge of one programming language to write a mobile application. This claim seems to be true at least in the case of the test application and other applications of similar complexity. Unless native components need to be developed, React Native applications can be written purely in JavaScript which may make this technology appealing to web developers who are looking to get into mobile development.

Another major question, and one that has even more significance to the financial viability of cross-platform mobile development, is code reuse. Facebook claims that about 75% of React Native code can be shared across platforms even when developing mobile applications that are designed to perform as natively as possible[20]. In the test application the code reuse is excellent because the platform-specific code

had to be written for only one out of five UI components, and the business logic of the application, essentially all logic and data processing, is fully shared. Niclas Hansson and Tomas Vidhall found in their research that this claim can be considered true or false depending on how we look at the phrasing. In their entire project 62% of the code was shared but if we look at the Android and iOS versions as separate applications, over 75% of the code used for each platform was shared across platforms.[4] If a similar web application was to be developed using the Redux library for state management, the business logic could be shared across projects since it is in no way bound to the user interface layer, which can be regarded as a bonus

The most interesting part of the project and one that can tell much more about React Native in general, was implementing the barcode scanner. Initially this was thought to be the biggest challenge of the project but doing more research on the subject, it seemed like Expo already has a working library that implements exactly what was needed. The Expo library was used and in the end the barcode scanner turned out to be one of the easiest features of the application in terms of implementation.

The barcode scanner also reveals what can be considered a strength or a flaw in React Native's design, namely the reliance third party libraries. The barcode scanner was easy to implement because Expo had already developed a well-working library for this purpose and published it as open source. If this was not the case, native modules would have had to be developed for both iOS and Android to be able to use the native features and as such the project would have required the knowledge of at least three different programming languages: JavaScript for React Native, Swift or Objective-C for iOS, and Java or Kotlin for Android. The difficulty is further increased because knowledge of each platform's preferred method for barcode scanning would also be needed. Not only that, creating a module like this would also require knowledge of React Native's architecture. This would immediately negate one of the largest strengths of the cross-platform technologies.

Other issues that come up when using third party libraries are maintainability and lifespan. Apple and Google always keep their Source Development Kits (SDKs) up to date with the latest operating system versions, and they release developer previews before a major operating system update so that the native application developers are able to keep their software up to date. With cross-platform technologies we do not only expect Apple and Google to do this, but we also trust the cross-platform authors to keep their frameworks up to date, and we trust the third parties to keep their libraries up to date. Sometimes this does not happen, and when a major update hits the consumers, the applications will stop working until somebody else has fixed the issues within their library or framework. This happened with React

Native and Android 8.0 (Oreo) in 2017 so it is not just a hypothetical threat[31]. Matias Martinez and Sylvain Lecomte also discovered these issues in their research on the quality of cross-platform applications[5].

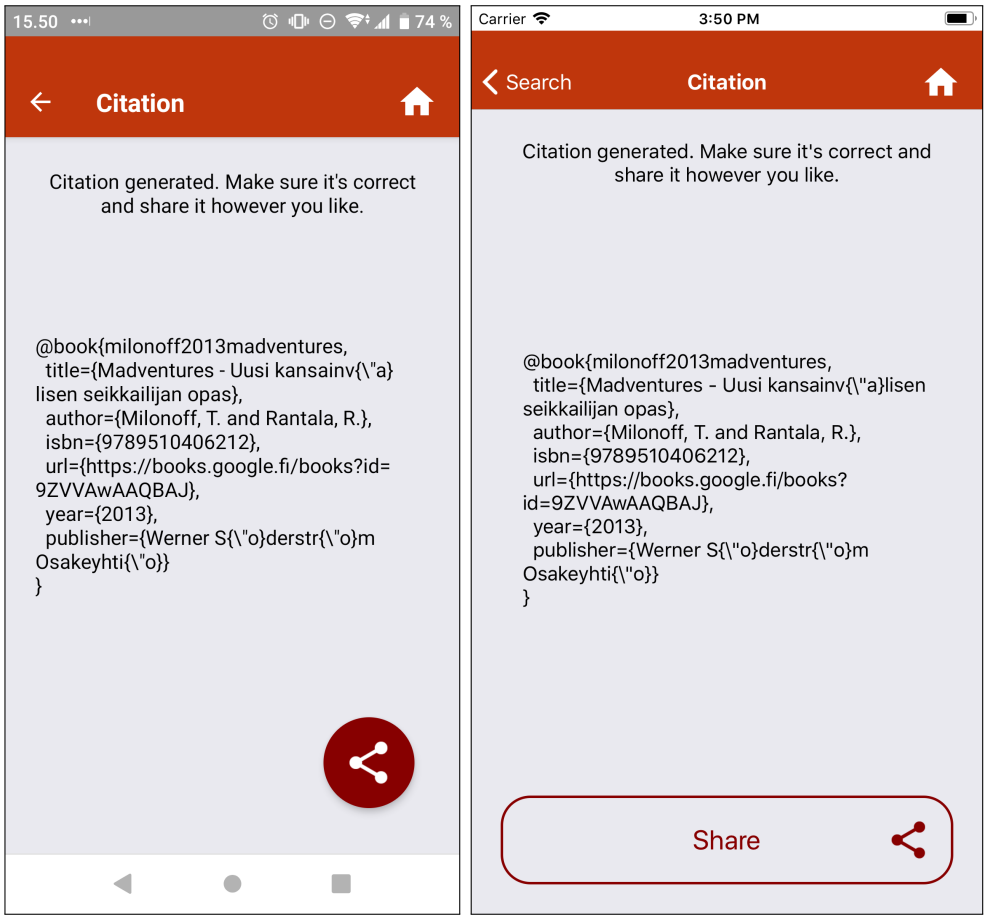
Third party libraries may also be using deprecated features to achieve what they want, such as barcode scanning. Expo has done well in this regard, and their BarCodeScanner library uses the newest Google Machine Vision API and Apple AVFoundation to do their scanning[32], but developers can only hope that Expo continues the maintenance and implementation of new APIs in the future, too. Third party libraries using deprecated features can become a major issue during an operating system update, since these features may be removed.

6.3 User Experience

It is not that difficult to achieve a native look and feel in a React Native application. One important factor in this is that the JavaScript code usually only describes the layout of the view, and React Native uses the native UI components to render the application. It is also easy to add platform-specific styling when it is needed. The largest potential difficulty comes from components where the platform-specific implementations differ greatly, even with the abstraction provided by React Native. The Picker and DatePicker are examples of such components which require the parent component to use a different layout depending on the platform[20].

6.3.1 User Interface

The user experience of the test application is on par with natively developed applications. The user interface feels responsive and fits to the overall style of the platform. The layouts of the views follow each platform's design guides relatively well and the smaller components of the application, such as buttons or lists, also look and feel native.



(a) (b)

Figure 6.1 *The citation view of the test application for Android (a) and iOS (b)*

In Figure 6.1 one can see the slight differences in the user interface for Android and iOS. The biggest difference is the share-button, which does have platform-specific styling, but it is internal to the custom button component. No other part of the application needs to be aware of the platform in order for the button to render properly.

Hansson and Vidhall also found that the React Native application and the native applications which they developed for their study had very similar user experiences and no significant differences were observed[4]. This is an expected result as long as we pay sufficient attention to the user interface and platform style during the development. As with the native development technologies, React Native does not guarantee a good user experience and it is still up to the designers and developers to create an application that feels like a natural part of the platform.

6.3.2 Performance

From the user's perspective the test application is slightly slow to start, but once the application is running, all actions feel snappy and quick on a modern smartphone. No noticeable dips in performance were observed during the testing but the test application is not complex, and a larger application might reveal more issues with the performance.

The final APK file was 20,4 megabytes which is fairly large for an application of this scale. Initially this was thought to be caused by the added graphics but this was not the case; earlier APK builds without the graphics were almost of the same size. This is an area where the native application definitely surpasses all the others, because React Native applications have to contain all of the used JavaScript code, libraries included, as well as a JavaScript runtime.

More accurate testing needs to be done to determine how the performance of React Native compares against native applications, but due to the limited scope of this thesis, performance testing was not included. Hansson and Vidhall tested their React Native application and identical native applications for performance, and the findings of this study largely correlates with what they discovered. CPU usage is consistently higher in the React Native application but memory usage and response time are almost equal to the native applications, sometimes even slightly better.[4]

7. CONCLUSIONS

Overall the results of the study look promising for React Native. All of the main goals for the developed application were achieved and, the final result had the look and feel of a native application. However, is seems that further research still needs to be done before solid conclusions can be drawn for cross-platform mobile development as a whole. The development technologies continue to evolve at a rapid pace as well, so much of the past research becomes outdated with time.

Many facets of application development were not discussed at all within this thesis but they are still significant factors for successful development projects. These include testing in the form of unit testing and automated UI testing, publishing a cross-platform application, and automating the building, testing and publishing processes.

Based on the results React Native, and cross-platform mobile technologies in general, can be a viable option to developing native mobile applications. Certain requirements do not change, and it is still the designers' and developers' responsibility to create the native user experience for each platform, and as such, mobile development still requires knowledge of the specific user experience and design guidelines for each platform. Developing the application by using cross-platform technologies can make the development process easier and less time consuming in some ways, and the maintainability of the application should also increase due to code reuse.

Even though cross-platform technologies may be a viable option for mobile development, based on this study alone it is unclear when they have a clear advantage over native development. In simple applications the differences in development time and maintainability are not significant, and in more complex applications the increased abstraction might not offer any benefits to the developers. In fact, in some cases, when we need to implement a lot of platform native features to the application, cross-platform technologies might only act as a hindrance to the progress of the application.

Surprisingly it seems like the largest potential issue for React Native and similar

technologies comes from maintainability. In a React Native project one not only relies on Google and Apple to keep their own SDKs up to date, one also relies on the whole group of third party organizations and people to keep their frameworks, libraries and tools up to date as well. This is not a problem when everything works, and the open source community has definite strength in that regard. The real issues appear when even one of the third party frameworks, libraries or tools is dropped from the support and maintenance.

BIBLIOGRAPHY

[1] N. McDonald, We Are Social: Digital in 2018, Jan. 2018, Available (accessed 17.11.2018): https://wearesocial.com/us/blog/2018/01/global-digital-report-2018.

[2] L. Sydow, S. Cheney, App Annie 2017 Retrospective, Jan. 2018, Available (accessed 17.11.2018): https://www.appannie.com/en/insights/market-data/app-annie-2017-retrospective/.

[3] IDC, Smartphone OS Market Share, 2018, Available (accessed 15.11.2018): https://www.idc.com/promo/smartphone-market-share/os.

[4] N. Hansson, T. Vidhall, Effects on performance and usability for cross-platform application development using React Native, PhD thesis, 2016.

[5] M. Martinez, S. Lecomte, Towards the Quality Improvement of Cross-Platform Mobile Applications, 2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft), May 2017, pp. 184–188.

[6] Google, Android Developer Documentation, Available (accessed 3.11.2018): https://developer.android.com/index.html.

[7] Apple, iOS Developer Documentation, Available (accessed 3.11.2018): https://developer.apple.com/.

[8] J. Titus, Google I/O 2017: Empowering developers to build the best experiences across platforms, May 2017, Available (accessed 4.11.2018): https://android-developers.googleblog.com/2017/05/google-io-2017-empowering-developers-to.html.

[9] Apple, Swift Has Reached 1.0, Sept. 2014, Available (accessed 4.11.2018): https://developer.apple.com/swift/blog/?id=14.

[10] Apple, Apple Reinvents the Phone with iPhone, Jan. 2007, Available (accessed 16.11.2018): https://www.apple.com/newsroom/2007/01/09Apple-Reinvents-the-Phone-with-iPhone/.

[11] G. Hartmann, G. Stead, A. DeGani, Cross-platform mobile development, Mobile Learning Environment, Cambridge, Vol. 16, Iss. 9, Mar. 2011, pp. 158–171.

[12] C. P. R. Raj, S. B. Tolety, A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach, 2012 Annual IEEE India Conference (INDICON), ID: 1, 2012, pp. 625–629.

[13] Altexsoft, Performance Comparison: Xamarin.Forms, Xamarin.iOS, Xamarin.Android vs Android and iOS Native Applications, Aug. 2017, Available (accessed 18.11.2018): https://www.altexsoft.com/blog/engineering/performance-comparison-xamarin-forms-xamarin-ios-xamarin-android-vs-android-and-ios-native-applications/.

[14] K. Bircan, Cross-Platform Mobile App Development with Flutter — Xamarin — React Native: A Performance Focused Comparison, Nov. 2017, Available (accessed 18.11.2018): https://medium.com/@korhanbircan/cross-platform-mobile-app-development-with-flutter-xamarin-react-native-a-performance-focused-a4457bcbdacc.

[15] J. Peppers, G. Taskos, C. Bilgin, Xamarin: Cross-Platform Mobile Application Development, Packt Publishing, Birmingham, UNITED KINGDOM, 2016.

[16] Google, Flutter documentation, Available (accessed 18.11.2018): https://flutter.io/.

[17] S. Ladd, Announcing Flutter beta 1: Build beautiful native apps, Feb. 2018, Available (accessed 18.11.2018): https://medium.com/flutter-io/announcing-flutter-beta-1-build-beautiful-native-apps-dc142aea74c0.

[18] T. Sneath, Flutter 1.0: Google's Portable UI Toolkit, Dec. 2018, Available (accessed 21.12.2018): https://developers.googleblog.com/2018/12/flutter-10-googles-portable-ui-toolkit.html.

[19] V. Novick, React Native - Building Mobile Apps with JavaScript, Packt Publishing, Birmingham, UK, 2017.

[20] Facebook, React Native Documentation, Available (accessed 3.11.2018): https://facebook.github.io/react-native/index.html.

[21] E. International, ECMAScript® 2018 Language Specification, June 2018, Available (accessed 21.12.2018): https://www.ecma-international.org/ecma-262/9.0/index.html.

[22] N. Parashuram, Chain React 2018 - React Native Fabric Architecture, Available (accessed 27.11.2018): http://blog.nparashuram.com/2018/07/chain-react-2018-react-native-fabric.html.

[23] Facebook, React Documentation, Available (accessed 21.12.2018): https://reactjs.org/docs.

[24] Redux Documentation, Available (accessed 21.12.2018): https://redux.js.org.

[25] React Navigation documentation, Available (accessed 23.11.2018): https://reactnavigation.org.

[26] Apple, Code Signing, Available (accessed 23.11.2018): `https://developer.apple.com/support/code-signing/`.

[27] Google Books API documentation, Available (accessed 23.11.2018): `https://developers.google.com/books/`.

[28] ISBNdb API documentation, Available (accessed 21.12.2018): `https://isbndb.com/apidocs`.

[29] M. Kuitunen, GitHub: refgen-app, Available (accessed 20.11.2018): `https://github.com/kulmajaba/refgen-app`.

[30] M. Kuitunen, GitHub: refgen-app release 1.0, Nov. 2018, Available (accessed 20.11.2018): `https://github.com/kulmajaba/refgen-app/releases/tag/1.0`.

[31] A. Jack, React Native pull request #15601, Aug. 2017, Available (accessed 23.11.2018): `https://github.com/facebook/react-native/pull/15601`.

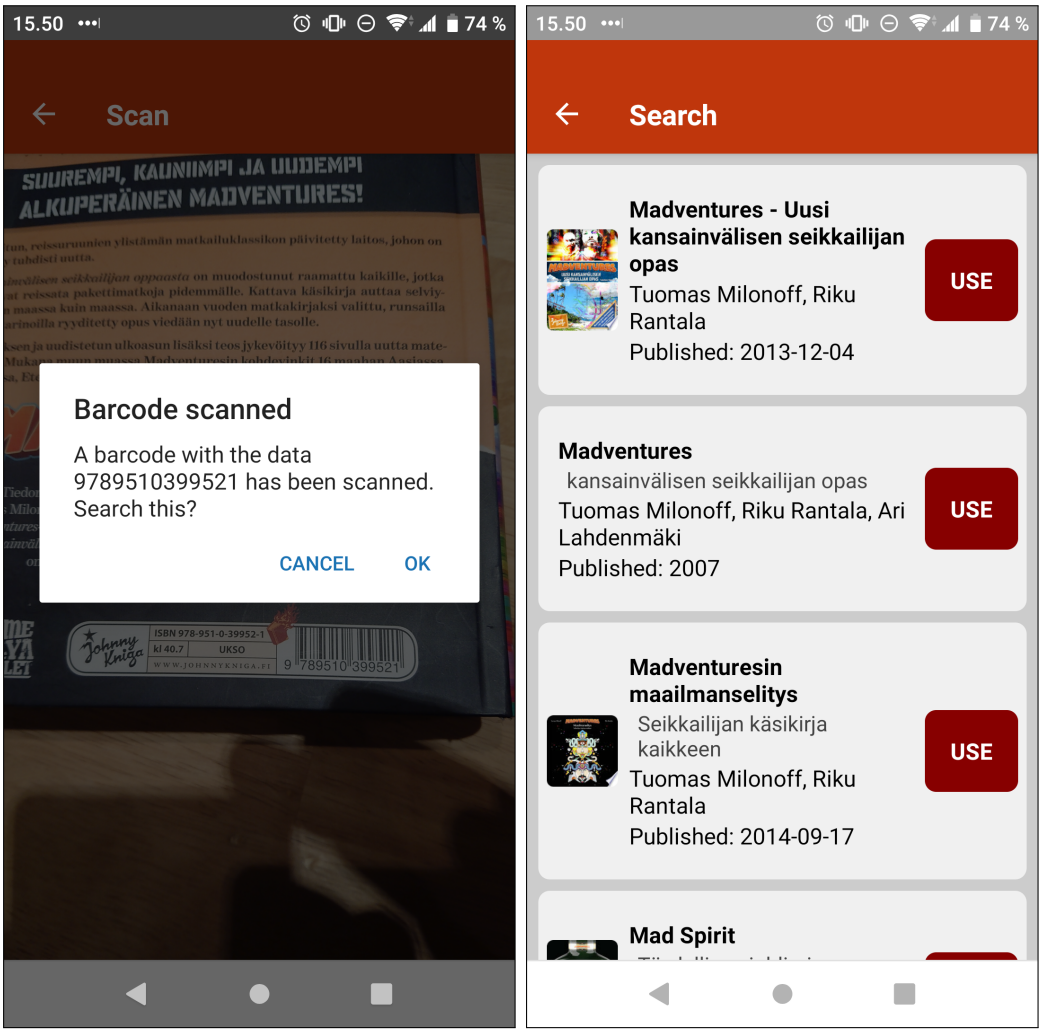
[32] Expo v30.0.0 documentation, Available (accessed 23.11.2018): `https://docs.expo.io/versions/v30.0.0/`.

APPENDIX 1. NAVIGATION FLOW OF THE TEST APPLICATION

(a) (b)

Figure 1 *(a) Splash screen (b) Home view*

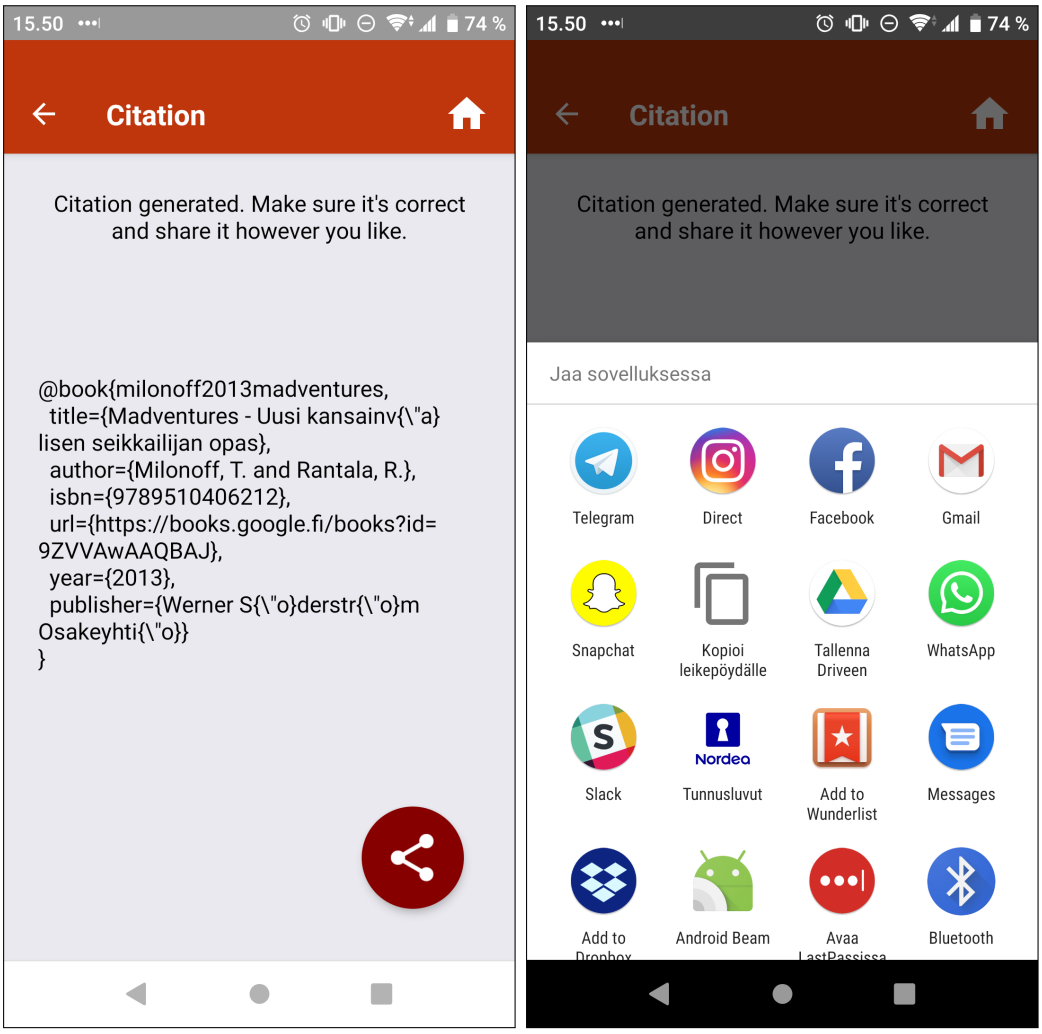
Appendix 1. Navigation Flow of the Test Application



(a) (b)

Figure 2 *(a) Scanning view, (b) Search view*

Appendix 1. Navigation Flow of the Test Application



(a) (b)

Figure 3 *(a) Citation view, (b) Share feature*