Oheneba Poku-Marboah

Mobile Application Development Methods: Comparing Native and Non-Native Applications

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

17 July 2021

Author Title Number of Pages Date	Oheneba Poku-Marboah Mobile Application Development Methods: Comparing Native and Non-Native Applications 34 pages + 0 appendices 17 July 2021
Degree	Bachelor of Engineering
Degree Programme	BSc Information Technology
Professional Major	Mobile Solutions
Instructors	Petri Vesikivi

The topic for this thesis is “Mobile Application Development Methods: Comparing Native and Non-Native Applications”. The purpose of the thesis is to acquaint the reader with factors to consider before commissioning a mobile application. For any decision-makers commissioning a mobile application for a business or other institution, general knowledge of how to reach a broad audience is essential.

Insight into non-native mobile applications’ viability comes from a performance analysis comparing a native mobile application to a non-native one. For this purpose, two identical mobile applications were built representing a native mobile application and a cross-platform mobile application. The applications were built using Kotlin for the native Android application and React Native for the cross-platform application. Because the applications were identical, the CPU load and memory consumption were measured while some tasks were performed. The measurements were then compared graphically to assess the efficiency and capability of the cross-platform application compared to the native method. Thus, it demonstrates that non-native mobile applications can be as capable as the native methods recommended by Google and Apple for their respective operating systems, even if they are not as memory-efficient.

This thesis deliberates the current state of the mobile industry. It expounds on the prominence of mobile applications, their usage, and the predicted trends. Furthermore, as the principal purpose, the different methods of mobile applications for a broad audience is discussed. The thesis’ scope does not include an in-depth look at any of the discussed methods but rather an overview to introduce them to the reader. The scope includes platform-dependent applications and platform-agnostic mobile software applications, i.e., cross-platform mobile applications.

Further research is needed on more varied use cases, for example, heavy usage of animations. In addition, the memory consumption could be further studied with a React Native application that is not built with Expo to see the difference in memory consumption.

Keywords	Android, iOS, Kotlin, mobile application development, mobile testing, React Native

Contents

List of Abbreviations

ADB	Android Debug Bridge
API	Application Programming Interface
APK	Android Package Kit
IDE	Integrated Developer Environment
OS	Operating System
SDK	Software Development Kit
UI	User Interface
WWDC	World Wide Developer Conference

1 Introduction

The thesis aims to acquaint the reader with technological and application development factors before commissioning the development of a mobile application. This thesis deliberates the current state of the mobile industry. It expounds on the prominence of mobile applications, their usage, and the predicted trends. Furthermore, as the principal purpose, the different methods of making mobile applications for a broad audience is discussed. This thesis gives an overview that makes software developers and institutional decision-makers aware of the options before choosing a stack to build their mobile applications.

Smartphone applications are an excellent way to reach consumers. Mobile phones are becoming increasingly ubiquitous. So, a mobile strategy needs to be adopted for organisations to have the most extensive possible reach [1]. What does mobile application development involve in 2021, and why should it be the choice instead of reaching consumers through a web application optimised for mobile devices? This thesis looks at the factors to be considered before choosing between a mobile application and a web application.

The current state of smartphone ownership and mobile application usage is at an all-time high, with consumers downloading a record-breaking 204 billion mobile applications in 2019 [2]. In the countries analysed in an App Annie report, on average, consumers spent 35% more time on their mobile devices in 2019 than 2 years prior [2]. However, not all smartphone users are available on the same mobile platform. There is a duopoly of mobile operating systems. That means that two operating systems capture over 98% of all smartphones [3]. These operating systems are Google’s Android operating system and Apple’s iOS. Consequently, making a mobile application with the maximum reach requires the applications to be deployed or available on both platforms.

Mobile applications have significant advantages in some scenarios over web applications. For example, when an application needs to utilise the in-built hardware of the phone, web applications are not the best solution. It can be unwieldy and a sub-optimal experience for a user to use web pages to access hardware functionality such as camera, GPS, and other hardware-demanding operations. In these cases, a mobile application is the best approach to serve the user on the smartphone.[4]

There are many approaches to building mobile applications. The methods are classified as native, cross-platform and hybrid. Native applications target one platform whilst cross-platform and hybrid target both leading platforms discussed in this thesis. For the reader to grasp what native, web, hybrid, and cross-platform means, the taxonomy of the application types is necessary. This thesis shall expound on the differences between native and non-native mobile application development. The thesis also discusses the features of hybrid and cross-platform applications, thus making them distinguishable from each other.

The thesis focuses on two platforms; they each have their approaches for native development. iOS mobile applications that are native to the iOS operating system are built using Objective-C and, more recently, Swift programming language. Android mobile applications that are native to the Android operating system are made using Java and, more recently, Kotlin programming languages. These native approaches have certain advantages over non-native methods, which are studied further in chapter 3.

There is a multitude of ways to develop a mobile application for both platforms [5]. An approach to consider is building a mobile application with the same languages used to create the native applications of that platform [5]. However, the app is then limited to that platform [5]. An alternative route is to make the mobile platform a hybrid one that deploys to all platforms. This approach is achieved using a native wrapper that wraps HTML content in a full-screen WebView, allowing it to mimic a mobile application [5]. A third option, a cross-platform application, uses one code base for both platforms but adapts elements of the codebase for the target platform.

How viable are these non-native methods of building a mobile application? Two mobile applications are built and compared to answer this question. One is built using a native approach, and the other is built using a cross-platform approach. The platform that the applications are built for in this comparison was the Android operating system. Two identical mobile applications were built, one using Kotlin, a native Android methodology, and the other using React Native, a cross-platform methodology.

Upon building these identical applications, they are tested by performing the same tasks using both applications. Whilst the tasks are performed, measurements are taken of the amount of memory each application uses and the load they both put on the CPU. The tests are done several times, and the average results are calculated. These measurements are done to determine how efficient the cross-platform application is in comparison to the native application. The applications are also compared in terms of the codebase and how many lines of code some components took to create. Finally, the results of the tests are presented and analysed to determine their feasibility as an option for mobile application development.

2 The Current State of Mobile Applications and Their Usage

This section discusses the proliferation of mobile applications and mobile devices. It expounds on the terms necessary to understand this thesis. The section opens up the importance of having a mobile application alongside or instead of a web application.

2.1 Taxonomy – A Brief Explanation of Key Terms

For full utilisation of this thesis, some key terms should be understood. These terms, explained in Table 1, range from ones that may seem self-explanatory, such as cross-platform, to terms borrowed from the real world into the realms of software development, such as native. These terms appear numerous times from here on to the end of this paper.

There appears not to be a standard definition of mobile application terminology such as hybrid and native [6]. Most papers accept them as the basic categories and sometimes add sub-categories and sub-divisions. Several writers, such as Nunkesser [6], mention the difficulty in the terms used to classify them and put forward their suggestions of terminology and classification.

Table 1. Key Terminology

Term	**Definition**
Web Applications	Web applications, also known as web apps or websites, are browser-based applications that are accessed or downloaded from a server. They are based on internet technologies such as HTML5, CSS, and JavaScript and are accessible from any browser regardless of platform. Web applications are not installed on the mobile device. Hence, they require internet access to retrieve information from the server. [7.]
Native Mobile Application	A native mobile application is a mobile application coded or programmed in the programming language recommended, often by the manufacturers or maintainers of that operating system, for that mobile operating system [8]. Native mobile applications are installed directly on the device. A critical distinction between native and non-native is that native applications only operate on the target platform.

Hybrid Mobile Application	Hybrid mobile applications are a combination of native applications and web applications; they are not native. Being able to deploy to multiple platforms is the advantage of hybrid applications that is most relevant to this thesis. Developers of hybrid applications do not require comprehensive knowledge of the target platform. Hybrid applications are installed on the device and get access to the device hardware with APIs developed for that platform. They often have a native look and feel. [7.]
Cross-Platform Application	Cross-platform applications take elements of a mobile application and adapt them to the target platform. They are applications deployable on multiple platforms; in the context of this thesis, iOS and Android operating system. The main difference between hybrid and cross-platform is that hybrid uses web tools. The main similarity between cross-platform and hybrid is that the applications for both platforms can share the same codebase. [9.]

The terms that are used most often in this paper are explained in Table 1. It is essential to comprehend and differentiate them to understand the following sections.

There are some difficulties in defining cross-platform. The first part of the compound word means it can exist on multiple platforms. In that sense, hybrid applications are certainly cross-platform applications. However, in the context of this thesis, it is defined according to Table 1, just like it is in some technical circles [6].

2.2 The Proliferation of Mobile Applications

In 2019, consumers downloaded a record of 204 billion mobile applications [2]. Across the countries analysed in App Annie's annual report, average mobile device usage per person was three hours and forty minutes daily [2]. In the US, between 2017 and 2019, mobile applications on smartphones jumped from 50% to 63% of time spent using digital media, establishing mobile phones as the go-to devices to access the internet [10]. At the same time, the amount of time spent using the desktop decreased from 34% to 23%. These statistics show that mobile platforms are where the average consumer is increasingly choosing to spend their time.

Why are mobile applications so popular? The size, power and connectivity of the smartphone are what has made it so ubiquitous. According to Business Insider [1], almost 80% of smartphone users check their phones within 15 minutes of waking up. 10% of those surveyed even admitted to using it during intercourse. The cost of a megabyte of data has dropped between 2005 and 2015 from $8 to a few cents, whilst smartphones also get increasingly cheaper. The price, coupled with their capability, has made smartphones into genuinely personal computers, thus causing the surge in popularity of mobile applications. [1.]

Thus, it is clear that institutions need to focus on mobile platforms in order to reach the maximum number of users. For the intents and purposes of this thesis, only two mobile operating systems are in consideration, the Android operating system and iOS. That is because Android and iOS operating systems make up over 98% of all mobile devices in use today, as shown in Figure 1 [3]. Software made for one mobile operating system (OS) is not necessarily compatible with software made for the other mobile OS. This duopoly in smartphone platforms [3] means mobile applications need to be developed for both prominent platforms.

Browser engines run JavaScript. Web browsers are ubiquitous in all mobile platforms, and they run JavaScript, the language of the web. In fact, this was such a powerful possibility that the original plan by Apple for iOS was to have third-party applications developed using open web technology. This approach by Apple changed later with the iPhone 2.0 in 2008 [11]. However, using the browser as a wrapper for an application is a viable option that is in use today; applications that use this method are called hybrid applications.

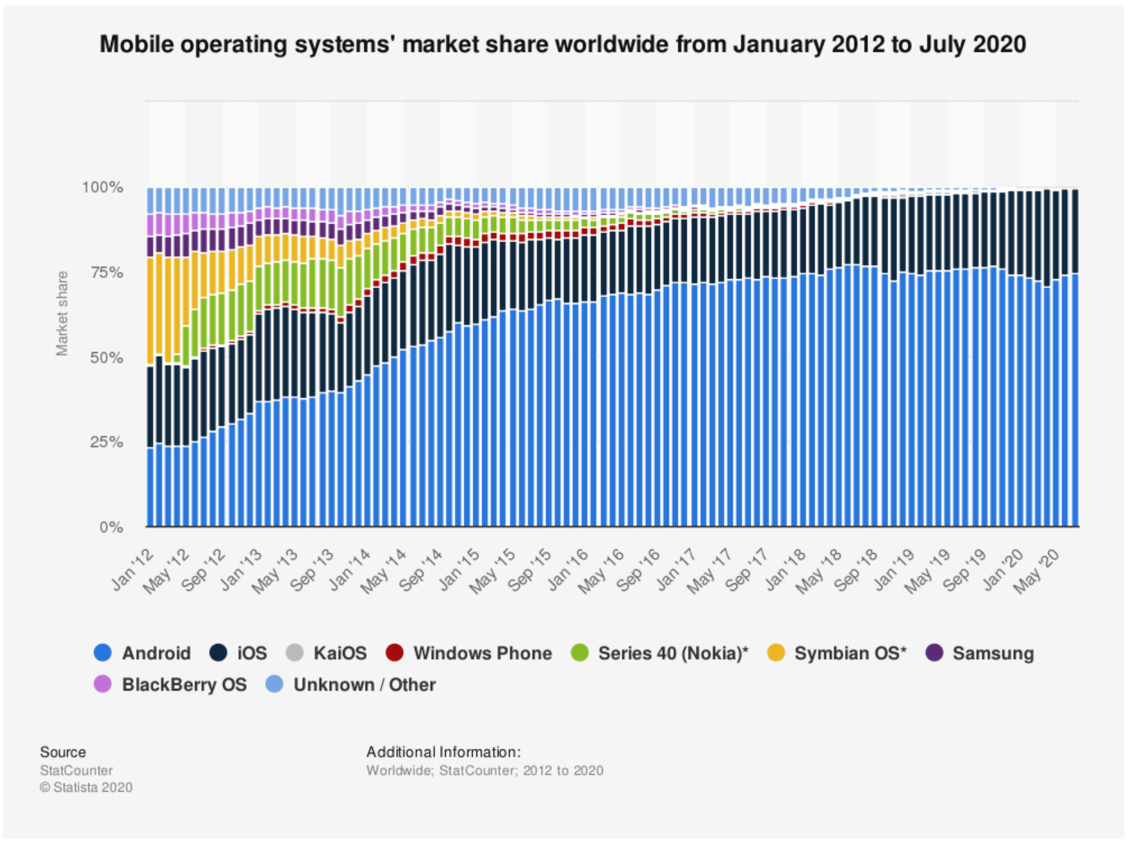


Figure 1. The market share of different mobile operating systems. [3].

As shown in Figure 1, two mobile operating systems have been dominant since 2015 and keep increasing in market share - Android operating system and iOS. They are the two platforms that matter if a mobile application is to target the biggest audience.

2.3 Mobile Application or Web Applications

There has been significant growth in the number of mobile phones in use over the last decade. Many institutions already have websites that are easily accessible from the desktop. With mobile traffic as high as it is and still growing, it is essential to also have a mobile presence in addition to the desktop presence. What kind of mobile presence is warranted? That depends on the mission at hand. Due to the costs involved in building a mobile application [4, p. 3], a website optimised for mobile devices can be a great alternative. There is a case to be made for websites optimised for mobile.

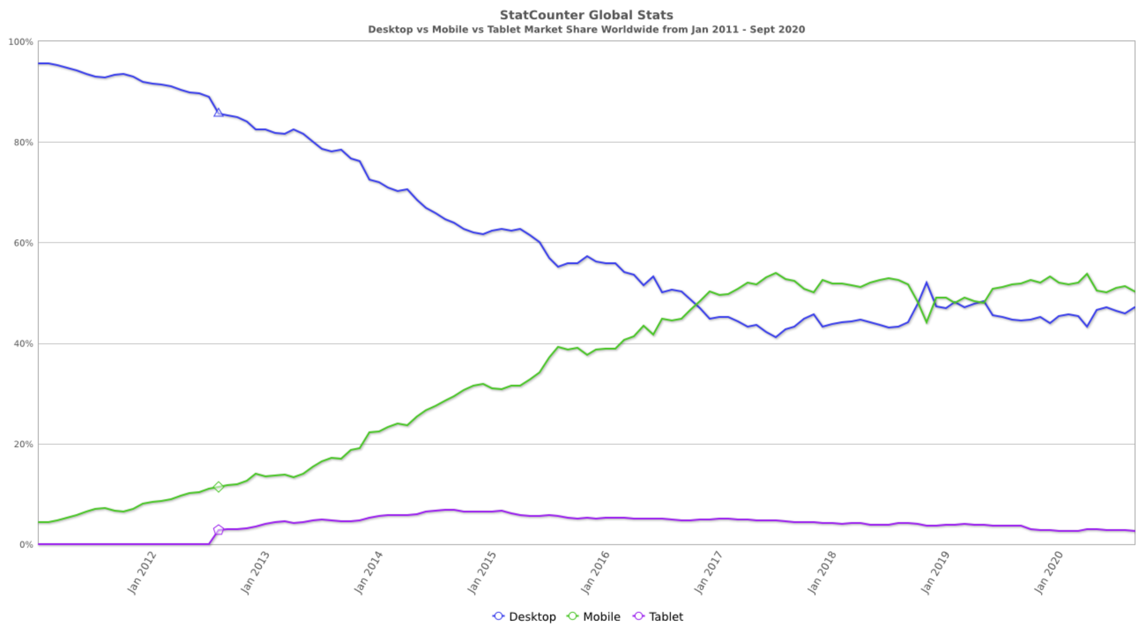


Figure 2. Market share of desktop, mobile and tablets. [12].

As shown in Figure 2, Mobile phones have steadily captured market share over the previous decade. Meanwhile, the number of desktop devices has declined steadily and only plateaued in 2017. As of 2020, there are more mobile devices in use than there are desktop devices.

There are scenarios where a mobile website makes the most sense. Many institutions have websites. However, not all are optimal for a good mobile web experience. Here are some advantages and disadvantages of mobile websites:

Advantages

- Web applications do not require installation on the mobile device before usage [5].
- Because the data comes from the server, it is effortless to upgrade a website. Mobile applications, on the other hand, require a download to up-grade [5].
- Mobile web browsers follow a similar standard, so it is easy to make a site that is accessible from all mobile platforms [5].

- Since a website is accessible from devices other than just mobile, it has a greater potential audience than a mobile application.

- Mobile applications on iOS and Android must pay 30% of the proceeds of in-app commerce to Apple and Google, respectively [13]. Websites are not subject to this fee.

Disadvantages

- A website's performance is subject to the quality of the network at the time of use [5].

- Mobile platforms sandbox web browsers. This sandboxing prevents web applications from freely accessing mobile device hardware such as camera, GPS, and other sensors [5].

- Web applications are severely limited when offline [7].

- Web developers have less control over how different browsers render their content [5].

If the functionality and experience of the required mobile service suffer due to the disadvantages above, then the benefits of a web application or a mobile-optimised website are undeniable. A mobile application is the best solution for a mobile presence in all other scenarios because the advantages outweigh the disadvantages.

2.4 Smartphones Versus Personal Computers: A Tale of Convenience

People are gravitating towards the use of smartphones over personal computers. Smartphones have managed to make an appearance in every facet of our lives. They outsell personal computers four to one [1]. In 2018 about half the adult population had smartphones. The prediction in 2015 was that by 2020, that figure would hover at around 80% [1]. As of 2019, there are a confirmed 3.2 billion users [14]. When asked which digital media they would miss the most, British teenagers chose mobile devices over TV, personal computers, and games consoles [1]. As mentioned earlier, the ubiquity of smartphones is so pervasive that 10% of adults asked admitted having used their smartphones during sex.

It begs the question, why are smartphones so popular? Why are desktop computers and laptops not as popular? Smartphones are intrinsically designed to be small and fit the pockets of users. They are light, with an average smartphone weighing between 100 and 300 grams. The costs of a mobile phone are not as prohibitive as that of a laptop or a desktop. A smartphone can cost as little as forty US dollars [1]. Another advantage of computers, at least desktop computers, is that smartphones come with in-built internet connectivity. Desktops tend to require an ethernet cable or wireless adapter to connect to the internet. [15.]

Personal computers, i.e. laptops and desktops, are far better in some scenarios. Because there is less of a size restriction on computers and laptops, they have a marked advantage in terms of the hardware they can ship. Desktops, for example, can have several storage disks totalling terabytes of storage. In addition, desktops do not have the power consumption constraints of a mobile phone. So, when the software requires significant electrical power to handle its complex computation, a mobile phone is not a suitable option. This restriction exists because mobile phones are battery-powered and thus need to use energy more efficiently. Although mobile phone processors get more powerful each year, they are still no match for desktops and laptops. That is because PCs have fewer size restrictions which allow a broader range of processing solutions. [15.]

3 Mobile Software Development Approaches

This section is about the different types of mobile applications; native, cross-platform and hybrid. It explains what they mean and introduces some of the ways and tools used in building mobile applications of that type. The techniques discussed are the most common techniques that fit most general-purpose applications. However, it must be noted that some other methods exist.

3.1 Native Mobile Application Development

The term native often refers to a program written in the machine language of the hardware platform it is running in. In the case of Mac and Windows applications, that means the x86 machine language [16]. However, since the dawn of mobile operating systems, native's meaning has evolved to mean more than it used to [16]. A native mobile application is coded or programmed in the programming language recommended, often by the manufacturers or maintainers of that operating system, for that mobile operating system [8]. In the case of iOS, the recommended languages for mobile application development are Objective-C and, starting from 2014, Swift [17]. For the Android operating system, the native languages for mobile application development are Java [8] and, beginning in 2019, Kotlin [18].

Sometimes native mobile applications refer to more than just the language it is programmed in. It can refer to the entire toolkit that goes into the development of the application. Another explanation of a native mobile application is an application developed using the SDK and the programming language(s) specific to that mobile operating system [19]. This definition of native is precisely how the word was used by Steve Jobs in 2008 WWDC when he launched the native SDK for the iPhone 2.0. “With the SDK in iPhone 2.0, we’re opening the same native APIs and tools we use internally... that means you as a dev can build apps for the iPhone the same way we do [12].”

A native application is the recommended solution if there are significant performance requirements according to Xanthopoulos and Xinogalos [7], and this will also be demonstrated by the performance evaluation results in chapter 4 of this thesis. A native application has complete access to the underlying hardware of the mobile device. On the other hand, it costs more to develop native applications because of having separate teams building for each platform [5]. However, the superior performance it provides is a considerable factor depending on the purpose of the application. In addition, native applications allow fully custom platform-specific UI implementations in contrast to the generic implementations provided by hybrid applications.

3.1.1 Native iOS Applications

Developing native applications for iOS requires a specific toolset. Apple recommends Swift and Objective-C as the programming languages of mobile applications of its iOS platform [8, 17]. Development for iOS requires either of those two languages or both used in combination [9]. iOS mobile application development happens in Apple’s integrated developer environment, or IDE for short, XCode. XCode comes with an interface builder, error alerting system and complete documentation [9]. These are all a part of Apple’s iOS SDK.

Developing for the iPhone requires owning a Mac or MacBook. XCode is only available in the Apple ecosystem, requiring application developers to have a Mac or MacBook. This obstacle of access to a development environment is a pain point for many naysayers of Apple’s closed ecosystem. There are some workarounds for this. Namely, developing via a virtualisation software like VirtualBox using a Mac image. However, this virtualisation approach still requires an Apple ID to gain access to XCode. That is a considerable cost and factor to account for before commissioning iOS applications.

The popularity of Objective-C has waned since Swift’s release in 2014 [9]. However, despite the popularity of Swift, Objective-C is still integral since most of the iOS SDK is written in it. Consequently, in developing an iOS application, a developer would be in contact with Objective-C code. In 2017, most iOS apps were still written in Objective-C. Swift, however, is more modern and more concise – Lyft rewrote their entire iOS application in Swift, going from 75 thousand lines of code to a third of that [20]. [9.]

3.1.2 Native Android Applications

Native Android applications are built using the Android SDK. The programming language recommended for Android development was Java for a long time until Kotlin’s announcement as the new recommended programming language. Both Java and Kotlin still have support in Android studio. Java and Kotlin code can be used in tandem to build an application.

Kotlin is concise, which means there is less code to type, maintain and test [18]. Java is a prevalent language. It has been around a long time and, as a result, has extensive documentation and a broad talent pool. Kotlin solves some of the inherent problems with Java, such as its verbosity. Google recommends Kotlin over Java because it has greater security and can be combined with Java [21]. The greater security is because Kotlin supports non-nullable types, making it less prone to have NullPointerExceptions [21]. In addition, Kotlin is a Java-based language and can be converted to Java or vice versa very easily. This interoperability allows Kotlin to use the Java frameworks and libraries very easily.

Android Studio is not limited to any platform. Unlike XCode, which requires an Apple computer to access, Android studio is available on Windows, Mac OS, and Linux. Android Studio comes with testing tools and frameworks, a layout editor, and a translations editor [22]. Android Studio also has an emulator that allows developers to interact with a virtual android phone to run their application [22]. This emulator is often used by some hybrid and cross-platform development frameworks to test and run android applications.

3.2 Hybrid Mobile Application Development

Hybrid mobile applications place an HTML5 application inside a thin native wrapper, as illustrated in Figure 3. The wrapper is UIWebView in iOS and WebView in Android. A browser executes the source code, just like with web applications. The browser is a part of the final application and is packaged with the source code, unlike web applications that need to download the source code from the web. The most popular hybrid application development frameworks are Ionic and Apache Cordova/PhoneGap. [7.]

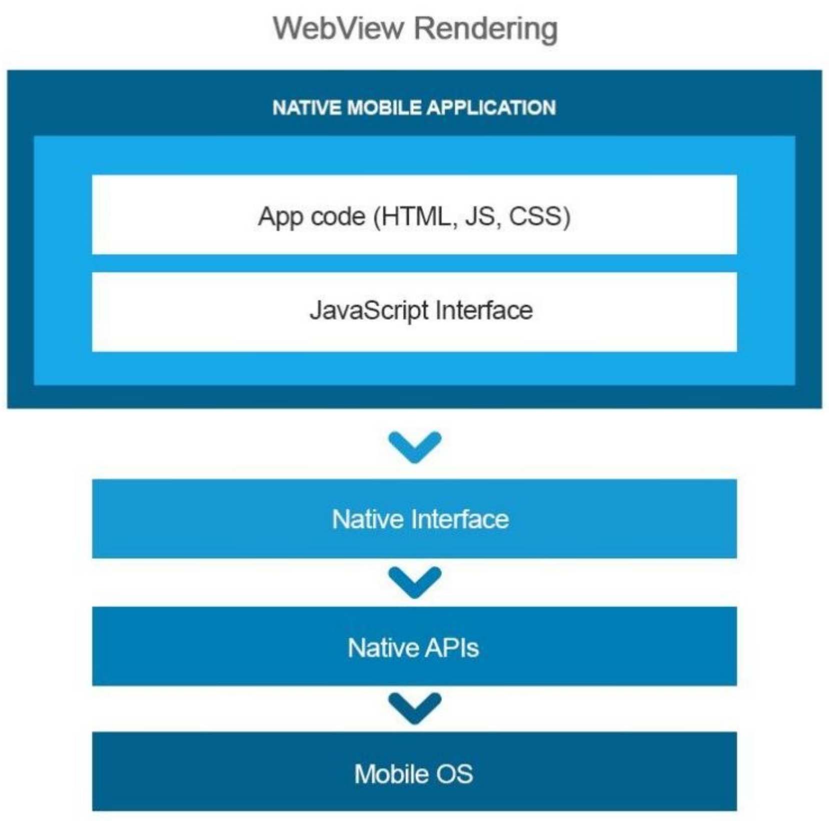


Figure 3. An illustration of hybrid Mobile application architecture. [9.]

As illustrated by Figure 3, hybrid applications are written with HTML, CSS and JavaScript and interface with the Native platform with a JavaScript interface. They are wrapped thinly by a native mobile application. They have access to the mobile operating system via the Native APIs that are exposed by the communication of the JavaScript – Native interface.

Before the iOS/Android duopoly, there were several mobile operating systems. Developing natively for all of them involved many tools, build systems and devices with different capabilities [23]. That also meant having several teams, which added to the cost of development. However, all platforms had browsers and as such, hybrid application development was a desirable option to avoid all the platform-specific hurdles.

3.2.1 Apache Cordova/PhoneGap

Apache Cordova, previously known as PhoneGap, is an open-source mobile development framework. The project was started initially by a company called Nitobi and was bought by Adobe in 2011. Adobe then branded the product as PhoneGap and released an open-source version of PhoneGap called Apache Cordova; some still refer to Apache Cordova as PhoneGap. PhoneGap is Adobe's commercial version of Apache Cordova. However, for various reasons, such as Progressive Web Apps (PWA) improvements and declining PhoneGap usage, Adobe decided to cease developing Adobe PhoneGap in October 2020 [24].

Apache Cordova allows developers to create hybrid mobile applications for both prevalent platforms. It is a runtime environment for web applications to run on native platforms [25]. Apache Cordova can support any kind of web page [25]. These pages are built using HTML5, CSS and JavaScript. Creating HTML and CSS components can be slow and tedious. Consequently, there are frameworks for quicker development, and Apache Cordova/PhoneGap supports them. The main ones are jQuery, AngularJS, and Knockout.js [23].

Apache Cordova/PhoneGap is a great tool in some scenarios. An advantage of using it is that developers do not need to learn a new language from scratch but can develop mobile applications with their existing web development skills. It has access to call native features via JavaScript APIs. It gives access to all the main hardware components such as camera, GPS and more through the JavaScript APIs. The PhoneGap framework has the native code pieces necessary to interact with the phone's operating system. It passes this information to the JavaScript application running in the WebView wrapper. [23.]

3.2.2 Ionic

Ionic is an open-source mobile development framework. It is arguably the most popular hybrid mobile application framework [25]. It is so highly regarded that when Adobe decided to stop funding the commercial PhoneGap product, they collaborated with Ionic as an alternative that developers could use to continue their development [24]. Ionic itself utilises Apache Cordova to deliver hybrid mobile applications [25]. It uses Apache Cordova as the runtime environment and to communicate with native platforms [25].

Ionic mobile application development involves JavaScript, CSS and HTML5. Ionic uses web development frameworks and is framework-agnostic. That means developers can build mobile applications with Ionic using one of the popular web frameworks they already know, such as Angular, React or Vue.

Ionic development is quick since it comes with pre-built components. In addition, all the standard components are available, such as cards, pop-ups and menus. An advantage of using Ionic is that the learning curve is not steep and framework-agnostic. Thus, developers can quickly work on a project without spending extensive time learning a framework. However, the drawback with Ionic, and other hybrid applications, is that the performance is not as good as a native application. Therefore, applications that demand much native functionality should not use with Ionic. These include games and other applications that need complicated hardware functionality.

3.3 Cross-Platform Mobile Application Development

Cross-platform development, like hybrid application development, uses a single codebase and toolset to deliver an application that works on multiple platforms. Cross-platform frameworks have better access to the device hardware than hybrid applications since they do not interface through the browser but instead adapt elements for the target platform. The difference between them and native applications is that they usually need a step or mechanism that compiles the application into native code. Another difference that works in favour of cross-platform applications is that Android and iOS applications can be built from the same codebase [26] without needing two dedicated teams and codebases like native applications. There are different ways of doing this, which sometimes impact the application’s performance. The most popular options for cross-platform development are React Native, Xamarin, and Flutter. This section expounds on these mobile development methods and their tools in sections discussed in sections 3.3.1, 3.3.2 and 3.3.3, respectively.

3.3.1 React Native

React Native is a JavaScript mobile application development framework. Facebook developed a popular web development framework, React, and made it open-source. They then created React Native, a cross-platform mobile application framework based on React, and again made that open-source. React Native renders natively on iOS and Android [27]. It is a young framework, launched in 2015 [27]. Consequently, since many platforms had already declined before its inception, React Native did not need to support them. So, it only supports iOS and Android, unlike Xamarin, which can develop applications for those two platforms and more [9].

React uses JavaScript and XML-like markup, known as JSX, for building applications. It then uses a bridge to invoke the native rendering APIs, Java for Android and Objective-C for iOS. React Native has APIs that interact natively with the device hardware, such as the camera or phone location. React Native also allows developers to combine it and native code for implementation that requires doing so. React Native makes it easy to reuse code across platforms. However, not all the code is usable on both platforms. For example, Facebook’s Ads Manager shares about 87% per cent of its code between Android and iOS [27]

React Native is a popular choice for developers. Its popularity is because the framework is based on the popular React web framework, which many web developers know and love. It allows them to transfer their skills to mobile application development seamlessly. Additionally, the developer community praises the toolset, error handling and hot-reloading functionality of React Native. The downside with React Native is that it is a very young framework and is yet to support all the Android and iOS functionality. Another drawback is that it adds another layer to the application, making debugging a bit more complicated. React Native is a framework you want to use for applications that do not require niche hardware access since it might not have the necessary bridge for the native API required. [27.]

3.3.2 Xamarin

Xamarin allows developers to make applications for many platforms. It can be used to make Android, iOS, Apple watch’s watchOS and Android Wear applications. Most of the mentioned hybrid and cross-platform development frameworks use a language already present in a mobile operating system; JavaScript, in the browser. Xamarin bucks that trend; it uses C# and .Net. It was initially a commercial product until the company was acquired by Microsoft in 2016 and made open-source. Xamarin applications are cross-compiled for the target platform. It is a popular tool for cross-platform application development within the Microsoft ecosystem. [28.]

Xamarin applications are written in C# and .Net framework and is natively compiled for all platforms that it targets. That makes it very fast and produces excellent performance with native characteristics. Developing with Xamarin involves the C# language and the .Net framework. Xamarin development occurs in the Visual Studio IDE. Visual Studio has many flavours, and all of them cost except the community version, which has some limitations. Switching from the native IDEs, Xcode and Android Studio, is very easy since they have the same functionality. C# has been around for a long time and has its known best practices and strong type-safety that allow the code to run predictably. [28.]

There are scenarios where Xamarin is a great solution. For developers familiar with C# and .Net, Xamarin is relatively easy to learn. Otherwise, Xamarin has a steep learning curve which is a significant obstacle if the developers in a team are not familiar with C# and .Net already. Xamarin does very well in interfacing with device hardware and can be used for more complex applications that need hardware resources. Additionally, there is the benefit of code-sharing for mobile platforms. However, the code-sharing is mainly that of logic code and not UI code. That is a drawback for applications with heavy graphics requirements, thus requiring some knowledge of native development. Other factors to consider about Xamarin are that the talent pool is much smaller than other frameworks, and the cost of the IDE for professional and enterprise use is high. [28.]

3.3.3 Flutter

Flutter is a framework by Google used for cross-platform development. At the time of writing, Flutter is growing in popularity. However, it is also in its infancy, having been released in 2017. Flutter uses Dart, which is compiled Ahead of Time (AOT) into native code on the target platforms. That makes Flutter applications perform very well and improves start-up time. It uses widgets in the application for UI rendering. It manipulates widgets in the developed application instead of manipulating the native widgets on the mobile platform like other cross-platform tools such as React Native do. This way, Flutter bypasses using a bridge to the native platform and thus renders very fast and consistently. [29.]

Mobile development with Flutter framework uses Dart Programming language. Dart compiles into native code for multiple platforms. The Flutter development environment recommended by the Flutter team is IntelliJ IDE with Android studio [30]. However, development in Visual Studio Code is also a possibility and has instructions in the Flutter dev documentation [30].

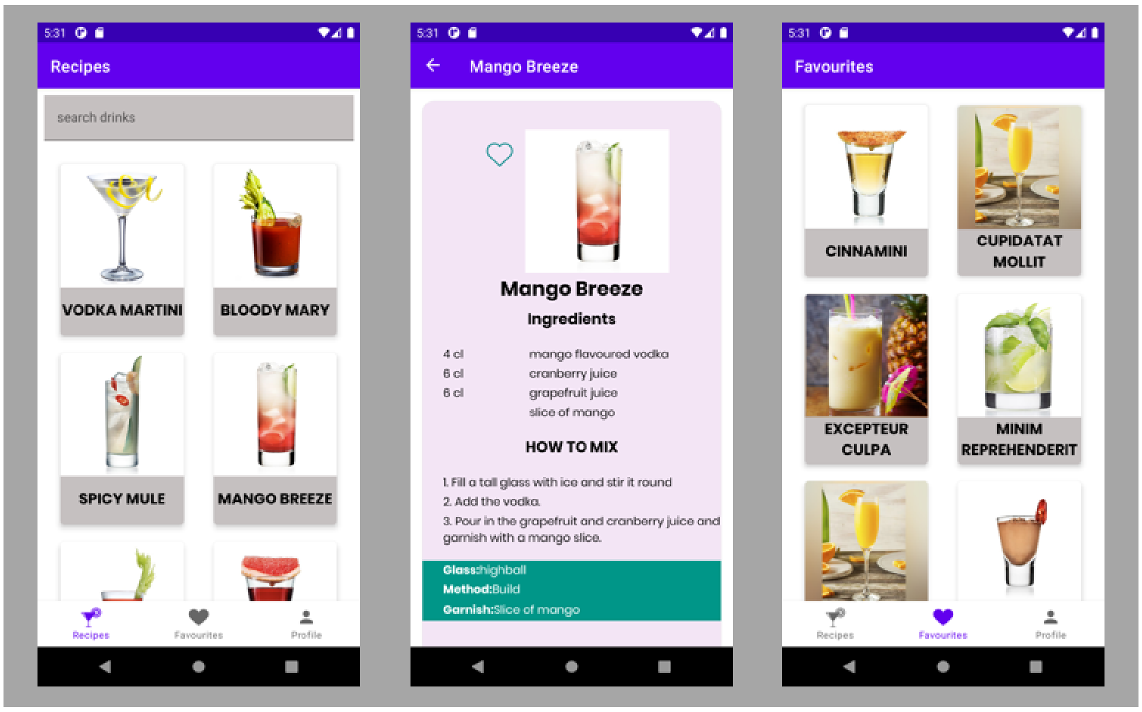
Flutter can be a great solution despite its age. Flutter applications are fast because it compiles to native code. Another reason for its speed is that it renders inside the application instead of on the platform. A drawback of not using the native widget is that developers must wait for new widgets to be added to Flutter or make a widget by themselves to use after platform updates. Since Flutter widgets are in-app, developers can make functionality for older operating systems that would otherwise not natively support that widget. A drawback for Flutter is that it is still new and as a consequence, the best practices are not so well known. Additionally, the talent pool of Flutter, as of 2020, was still relatively small relative to some of the other options. Lastly, another problem with Flutter being new is that there are a limited number of libraries available. [29.]

4 Comparing a Native Android Application to a React Native Application

In this section, there is a comparison between the two applications built for a performance evaluation. One is a cross-platform application, and the other is a native application- native to the Android operating system. The cross-platform application is built using React Native, and the other is built using Kotlin. This section details the work involved in building the two mobile applications involved in the comparison. It also talks about the structure and the methodology of the testing that was done. There is an overview of the technologies and systems that facilitated the mobile applications. Finally, it discusses the differences observed in the two mobile applications regarding the end product, codebase, and performance.

4.1 About the Application - Things Common to Both Applications

Much has been written about the different ways to build mobile applications. For this thesis, two apps were built with native and cross-platform methods to get a complete picture of their performance. Figure 4 shows some screenshots of one of the applications. The application was a cocktail application with a list of cocktails that the user could scroll through and search. The application had a start-up screen that allowed the user to choose if they want to use the browser-based version of the application, i.e. the WebView, or log in to the application.



Mango Breeze

Ingredients

4 cl	mango flavoured vodka
6 cl	cranberry juice
6 cl	grapefruit juice
	slice of mango

HOW TO MIX

1. Fill a tall glass with ice and stir it round
2. Add the vodka.
3. Pour in the grapefruit and cranberry juice and garnish with a mango slice.

Glass:highball
Method: Build
Garnish: Slice of mango

Figure 4. Screenshots of parts of the application

Figure 4 illustrates the parts of the built application. It shows the list of cocktails in the drinks tab, the favourites in the favourites tab and the details of a clicked drink. Since the applications were identical, screenshots from only one is shown. The similarity in appearance is demonstrated in other figures.

After logging in to the application the first time, it detects the subsequent attempts to open the application that the user logged in earlier and automatically moves to the home page. Beyond the login, there are three tabs:

1) Drinks tab - includes all the drinks in the application and the search functionality

2) Favourites tab - includes all the drinks that the user had previously liked

3) Profile tab - includes an option to log out

The drinks and favourites pages have a scrollable list of drinks that the user can click on to view the entire cocktail recipe and other facts. These features were implemented for both applications to behave as similarly as possible, providing an adequately comparable experience.

4.1.1 Backend

The mobile applications require a server component for the drinks and authentication. This server was built using the JavaScript framework, Express.js, and runs on the Node.js JavaScript runtime, Node.js. In addition, the backend database functionality is built with MongoDB and mongoose. Other libraries employed in building the server include bcrypt, jsonwebtoken and Morgan, which are all JavaScript libraries available on Node Package Manager.

One of the deployed backend limitations is that since it is on the free tier of Heroku, the server goes to sleep after 30 minutes of being idle [31]. This idleness affects the mobile apps if used when the server is in sleep mode, making the application seem unresponsive upon the first server query. This limitation is taken into account when testing the application, as further explained in section 4.3.2.

4.1.2 Web Page Option

One of the features of the applications is showing the cocktail service's webpage as an alternative method of using the application if the user does not want to log in. The webpage is served inside of what is known as a WebView. A WebView object allows displaying a website's content as part of a mobile application's content [32]. Adding this feature to the application required building a website that mimics some of the mobile application's functionality. The website had some application features such as viewing the recipe in detail and searching for drink objects.

4.2 Codebase and Development Comparison

This section discusses the codebase and the process of developing some components of the mobile applications. Then, it compares how many lines of code it takes to accomplish them and the appearance of the outcome. It is essential to differentiate between written and non-written code since written code is what the developer of the application writes in contrast to that generated by the React Native or Android frameworks. Finally, diagrams or images of the result are shown to demonstrate the similarity of the outcome.

4.2.1 Implementing a Scrollable List on Both Systems

It is common to have a list view in an application. This list view usually leads to a detailed view detailing the list item that was clicked. How does a scrollable list in React Native compare to that of a scrollable native Android list? The amount of code required for each and the structure of the systems is considerably different. Nevertheless, shown side-by-side in Figure 5, the results are virtually indistinguishable from each other; the exception is the colour deliberately selected for easier differentiation.

For the React Native list, a FlatList was used. A FlatList requires two things; the view to render for every item in the list and the data set of the list [33]. The React Native version took about 148 lines of written JavaScript code to accomplish. However, since the data comes from a network request, that does not contribute to the lines of code required to implement the scrollable list. The resulting list is in Figure 5.

The native Android scrollable list built with Kotlin was made using a RecyclerView. With a native Android list, several different classes work together to build a dynamic list [34]. The RecyclerView is a ViewGroup that holds the data and views corresponding to the list elements [34]. A view holder object defines each element, and the RecyclerView binds the view holder to its data. Binding the view to the data is done through methods in the RecyclerView adapter. Then a layout manager arranges and sets the orientation for the list elements. The RecyclerView itself is then held in a fragment or an activity. The native list required about 190 lines of Kotlin and XML code to create. The resulting list is shown in Figure 5,

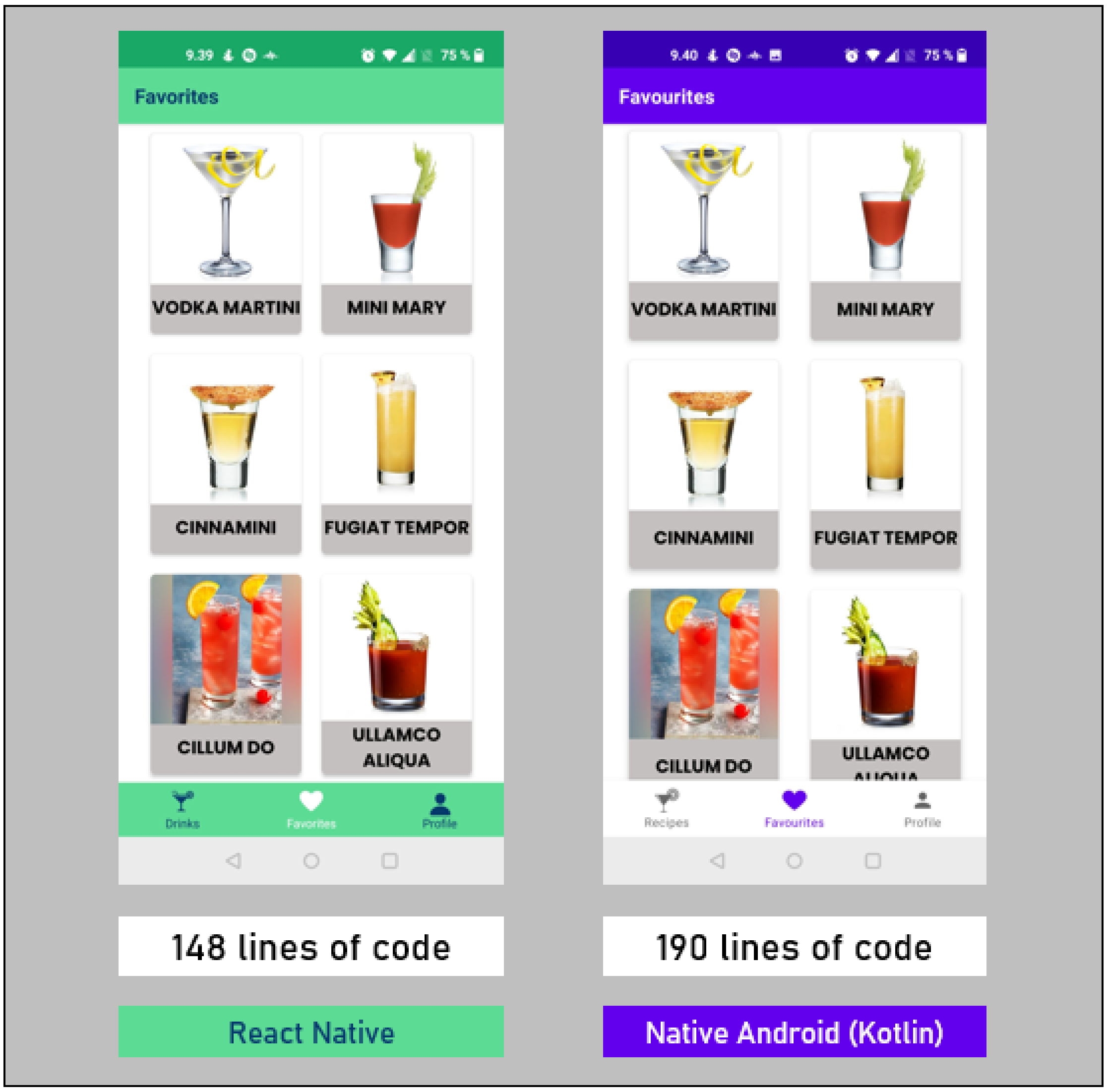


Figure 5. The resulting scrollable list of each platform

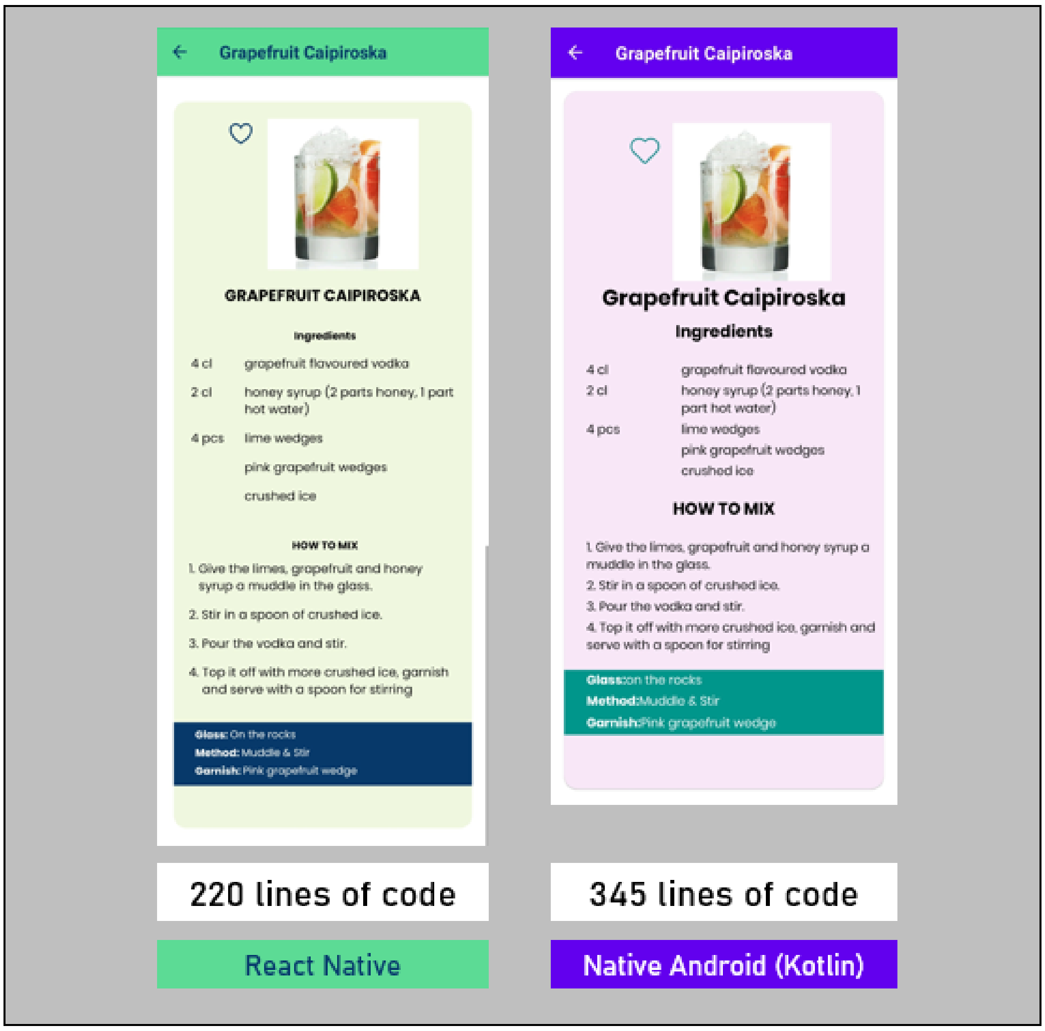
As illustrated in Figure 5, the resulting scrollable lists achieve the same goal. They are virtually indistinguishable, except for the colour, which was a deliberate choice for easier differentiation during the comparison.

The React Native version of the scrollable list took considerably fewer lines of written code to achieve than the native Android version. The React Native version took about 148 lines of written JavaScript code to accomplish, whereas the native one required about 190 lines of written Kotlin and XML code to create. That is the same goal achieved with about 22% fewer lines of code.

4.2.2 Implementing a Detail View on Both Systems

As mentioned in the previous subsection, a detailed view that displays further information about an element in a list is common in mobile applications. The test application had a detailed view of the cocktail recipe steps and other relevant information. Figure 5 demonstrates how similar of an outcome was achieved. What goes into making the detailed view, and how many lines of code does it require on either platform?

The React Native version was achieved with about 220 lines of written JavaScript code. The native Android version needed about 345 lines of written Kotlin and XML code. That means the React Native version required about 36% fewer lines of written code to achieve. As shown in Figure 6, the results are remarkably similar.



GRAPEFRUIT CAIPIROSKA

Ingredients

4 cl	grapefruit flavoured vodka
2 cl	honey syrup (2 parts honey, 1 part hot water)
4 pcs	lime wedges
	pink grapefruit wedges
	crushed ice

HOW TO MIX

1. Give the limes, grapefruit and honey syrup a muddle in the glass.
2. Stir in a spoon of crushed ice.
3. Pour the vodka and stir.
4. Top it off with more crushed ice, garnish and serve with a spoon for stirring

Glass: On the rocks
Method: Muddle & Stir
Garnish: Pink grapefruit wedge

Grapefruit Caipiroska

Ingredients

4 cl	grapefruit flavoured vodka
2 cl	honey syrup (2 parts honey, 1 part hot water)
4 pcs	lime wedges
	pink grapefruit wedges
	crushed ice

HOW TO MIX

1. Give the limes, grapefruit and honey syrup a muddle in the glass.
2. Stir in a spoon of crushed ice.
3. Pour the vodka and stir.
4. Top it off with more crushed ice, garnish and serve with a spoon for stirring

Glass:on the rocks
Method: Muddle & Stir
Garnish: Pink grapefruit wedge

Figure 6. The detail view on both platforms

As shown in Figure 6, the manifestation of the detail view of both applications is remarkably similar. However, it takes a considerably different amount of code to achieve.

4.3 Performance Comparison

The performance comparison was to ascertain the feasibility, efficiency, and capability of cross-platform mobile applications. It was necessary to perform some tasks on both native and cross-platform applications to see how they fared against each other. Since the mobile applications had the same flow of operations and appearance, except for the colour, actions for the test scenarios were the same on both applications. The scenarios used were the following:

1. Scroll from the top to the bottom of the ListView of all 1054 drink objects on the home page.

2. Type “mini” into the search bar on the home page, which updates the results as you type. Next, scroll to the bottom of the list and click on the last element, “minim eu”, to open its detail view.

4.3.1 Theory

Many different parameters can be measured on a smartphone to compare the efficiency and performance of one application to another. These include power consumption, memory usage, frame rate while the application is being used and the load or strain on the Central Processing Unit, CPU. In this thesis, only memory usage and CPU load were compared.

Usage of the Central Processing Unit (CPU) affects the smoothness of the user experience and the consumption of device battery life [35]. The CPU executes all the instructions that are given inside of a computer, from software and hardware. It is a resource that is shared by all systems and applications on a device [36]. As a result, abusing or hoarding the CPU affects the other applications and systems running in the background [36].

The amount of memory or the memory footprint for an application can be measured in several ways. The method used for the conducted tests was the proportional set size, also known as PSS. PSS is used because it takes into account all of the memory available to the device. PSS is the number of memory pages available exclusively to an application and the pages of memory being shared with other applications proportionately. It is a valuable metric for the operating system because adding all the PSS values gives the total amount of memory available on the device. That makes the PSS a good indicator for the actual RAM weight of the processes of an application. [38, 37.]

4.3.2 Method

The measurements of CPU load and memory consumption were done using Apptim. Apptim is a tool and platform for mobile performance testing [39]. It can be used both during the testing and development phases of application development. Apptim can be used to record problems and measurements in 3 primary areas, device resource usage, user experience, and crashes of the application [39]. For the thesis' purposes, it was used only for device resource usage measurements.

The testing device was a OnePlus 8T model KB2003 which, according to the Android Debug Bridge, ADB, had 7702 Mb of memory. Therefore, the memory usage percentage was calculated using this figure as the denominator. The device was running Android version 11. All non-system applications were turned off during the tests to avoid contaminating or affecting the measurements. Since the server uses the Heroku free tier, it goes into sleep mode after 30 minutes of inactivity. Mitigating the effects of this behaviour on the test results meant querying the server from the browser once before any test was conducted.

The tests were done five times for each scenario with both applications, and the average values were plotted in the graphical results provided. This five-time repetition was done to attain a more reliable and accurate representation of the data. Apptim took about 5 seconds to start the application. After the application start, a further 5 seconds of no interaction was allowed as a buffer which means all the data collection starts from the 10-second mark of Apptim's measurements. The 10-second mark is considered the start of the measurements collected for the graphs included in this thesis. Therefore, it is marked as 0 on the time axis of the graphs.

In the first 5 seconds of the collected data, the applications were given no user interaction to measure the memory and CPU load when idle. After that, the actions described in the test scenarios were conducted. After the completed actions, these actions were followed by another 5 seconds of no user interaction to capture the measurements whilst in an idle state.

4.3.3 Results

CPU Load Comparison

The results of the CPU load during both test scenarios are shown in Figure 7 and Figure 8. It is observable that there is a spike in both React Native and native Android CPU load and a subsequent tapering off in both graphs. This pattern is because of the sequence of actions taken during the tests, where the applications were idle for a few seconds before and after the actions had been undertaken. Thus, the peak is during the action, and the troughs are during the idle periods. Figure 7 shows the peak of the CPU strain of the React Native application being noticeably higher than that of the native Android application. However, in Figure 8, the peak of the CPU strain of the React Native application comes out as being slightly less than that of the native Android application.

The CPU usage affects the smoothness of the user experience and the consumption of device battery life [35]. It is important to note that the differences in CPU load, although noticeable on this scale, would be inconsequential or not noticeable if the graph scale went from 0 to 100. Consequently, there was no visible difference in the user experience of the applications in terms of slowness or lag. This similarity in smoothness is understandable because the difference in CPU loads is not significant.

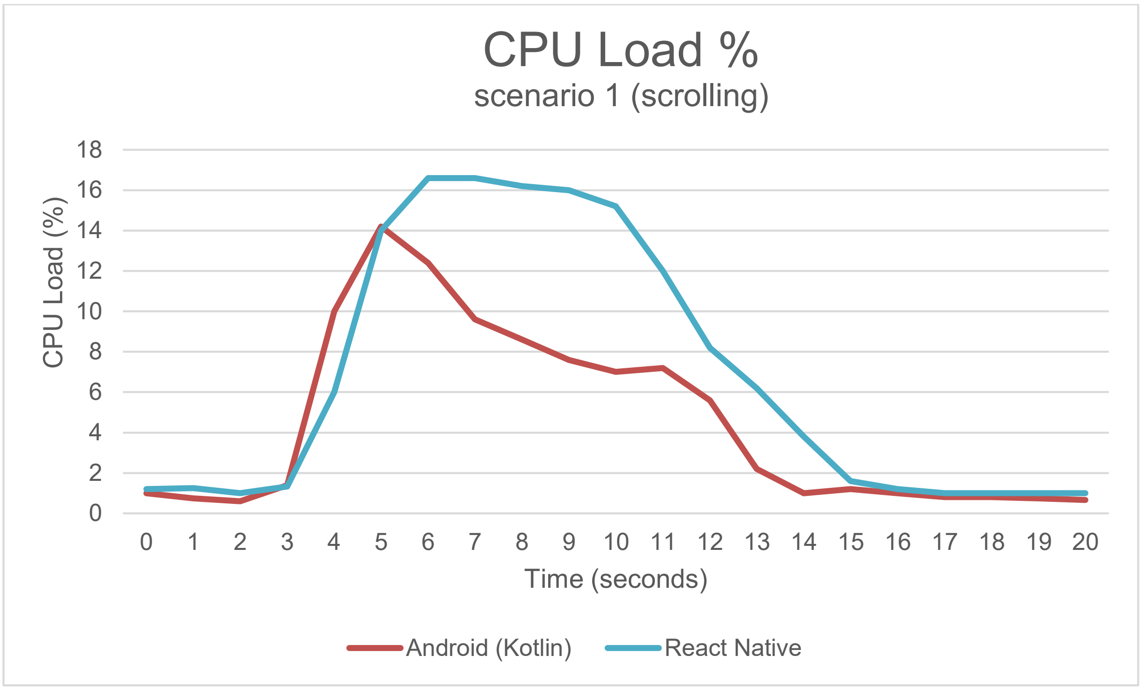


Figure 7. Average CPU load of the device when conducting test scenario 1 for the React Native and native Android applications

As shown in Figure 7, the CPU load of the React Native application is higher than that of the native Android application.

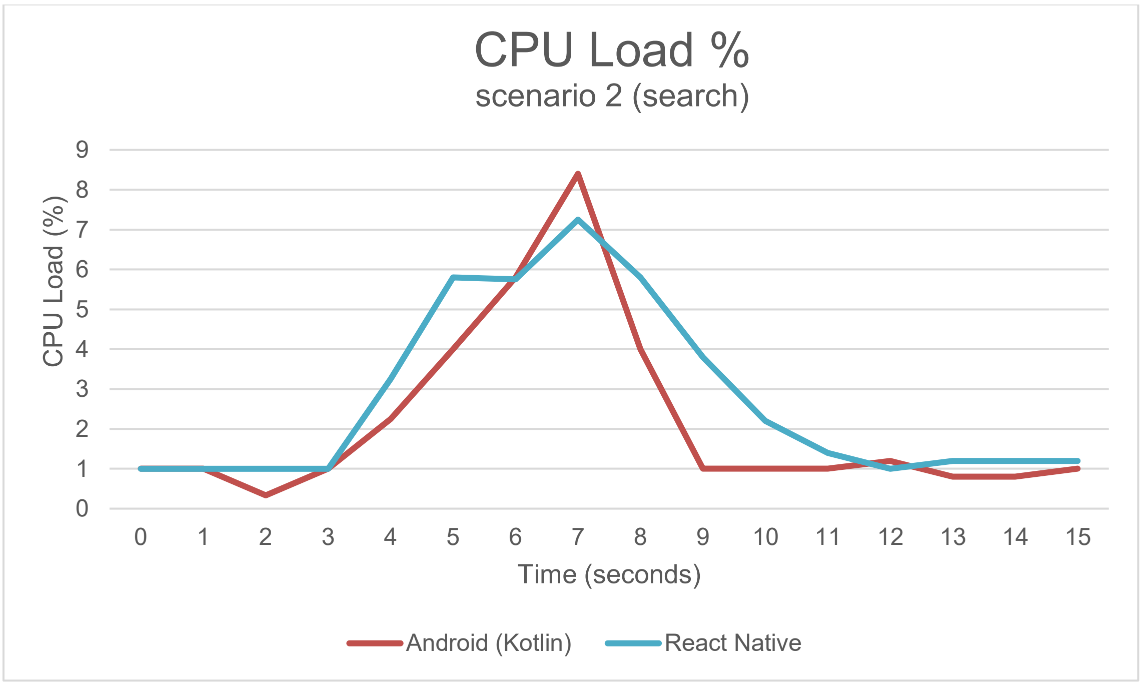


Figure 8. Average CPU load of the device when conducting test scenario 2 for the React Native and native Android applications

As illustrated in Figure 8, the CPU load of the native Android application is slightly higher than that of the React Native application. However, that is contrary to what is observed in Figure 7.

Memory Consumption Comparison

The React Native application needs substantially more memory than the native Android application in both test cases. In both test cases, it takes about twice as much memory as the native Android version. However, the amount of memory they both take, even at their peak, is at or under 4% of the total device memory capacity. Hence this makes the strain on the memory negligible enough not to affect the user experience. Thus, the applications run without any lags, making them both suitable methods for most mobile applications.

Moreover, the significant difference in memory usage exists because Expo tools were utilised in making the React Native application. Expo is a framework and platform for React applications. It provides a set of tools that help developers develop, build, and deploy mobile applications for iOS and Android [40]. Using Expo in building a React Native application leads to an additional 15 to 20 Mb in application package size[41]. This increase in size is because the application comes packaged with the entire Expo SDK, including the parts not used in your application [41].

The size of the application package kit, also known as APK, significantly affects the amount of memory the application requires [42]. The size of the APK directly affects how fast the application loads, how much memory the application uses and how much power it consumes [43]. After developing the applications, an Android Package Kit, APK, is generated. The APK is then installed onto the mobile device. It is worth noting that the size of the native Android APK was only 6.52 Mb, whereas the React Native APK was 53.4 Mb.

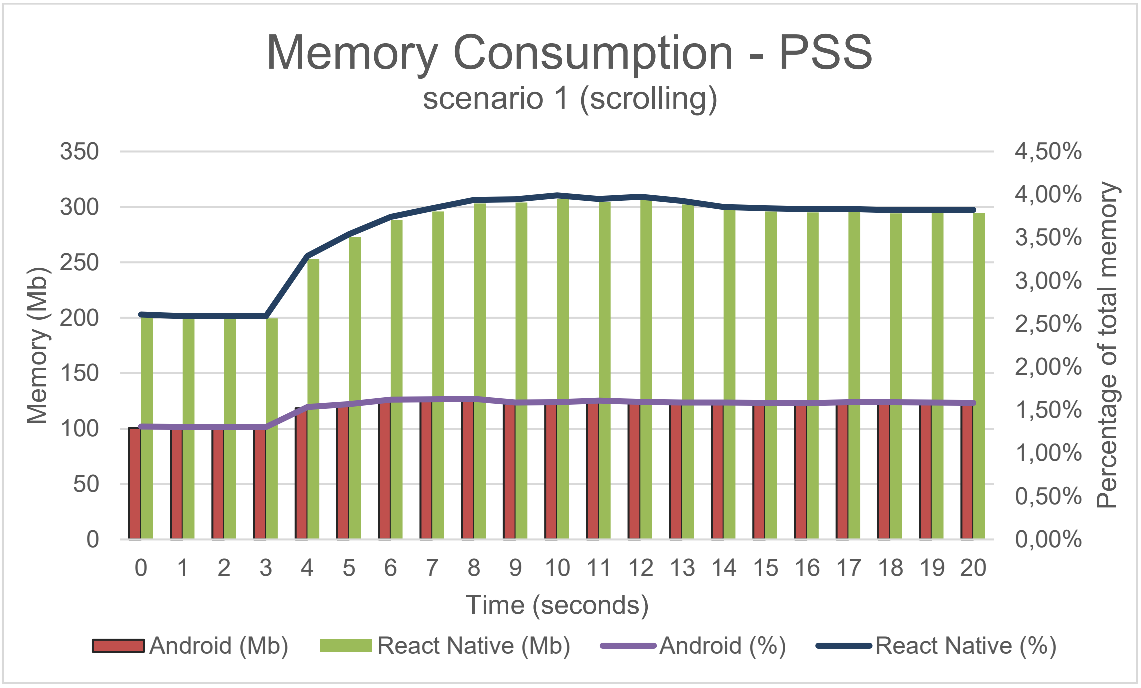


Figure 9. Average Memory Consumption of the device when conducting test scenario 1 for the React Native and native Android applications. Shown as a percentage of total memory and in Megabytes

As illustrated in Figure 9, the React Native application needs about twice as much memory as the native Android application. However, as a percentage of total phone memory, neither application uses a sizable proportion of the device memory.

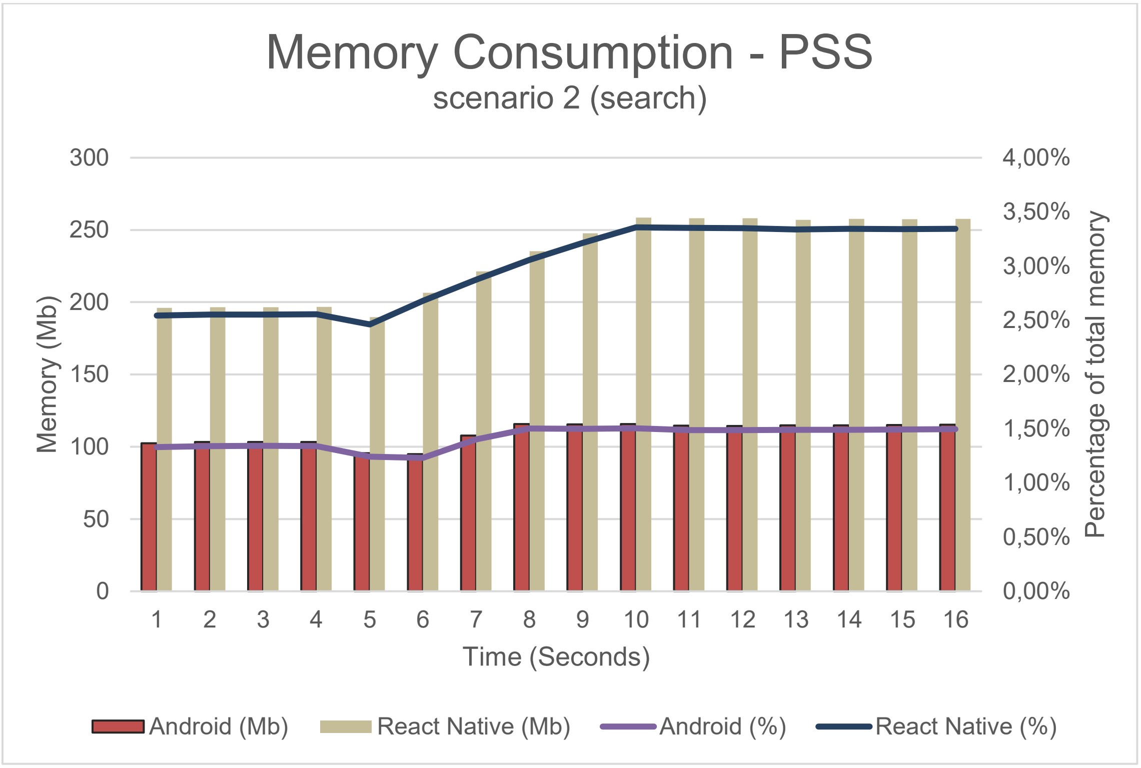


Figure 10. Average Memory Consumption of the device when conducting test scenario 2 for the React Native and native Android applications. Shown as a percentage of total memory and in Megabytes

As illustrated in Figure 10, the React Native application needs more memory than the native Android application. However, as a percentage of total phone memory, neither application uses a sizable proportion of the device memory, thus giving a smooth user experience.

5 Conclusion

The principal goal of this thesis was to inform the reader of the different ways of making a mobile application and the factors to consider when commissioning a mobile application. The factors were presented for the various kinds of mobile application development approaches they fall under; native, hybrid, and cross-platform. Consideration is given to the most common techniques that fit most general-purpose applications. The thesis enlightens the reader on the current state of the mobile application ecosystem and the proliferation of mobile devices. The thesis provides an overview that makes decision-makers aware of the options before choosing an approach to build their mobile applications.

The thesis explains the terms used to describe and categorise the different methods of building mobile applications. It gives the reader an overview of what is meant by native, hybrid and cross-platform mobile applications. It expounds on what a web application is and what makes them so different from a mobile application. There is an option of making a web application that is well-suited to the needs of a mobile device. That possibility is taken into account, explained, and differentiated from a mobile application in section 2.3. Finally, in section 2.4, the thesis sets forth the importance of an application on a mobile phone instead of a desktop-based one. Finally, it expounds on why it is essential to prioritise a mobile application and cases when it is not the best choice.

The mobile application development techniques discussed are the most common techniques that fit most general-purpose applications. The two biggest platforms, iOS and Android, each have their own recommended ways of building mobile applications. In the case of iOS, the recommended languages for mobile application development are Objective-C and, starting from 2014 [17], Swift [8] and for Android operating system, Java [8] and, beginning in 2019, [18] Kotlin. Other methods, cross-platform, and hybrid, utilise different ways of making applications for those systems. Some hybrid methods use a WebView, an existing component in Android and iOS, to create an experience reminiscent of a mobile application but with some shortcomings as they interface with the device through the browser[7]. Cross-platform mobile applications, on the other hand, adapt their elements for the target platforms, which is an extra step but has the advantage of not having many of the shortcomings of hybrid mobile applications. Both cross-platform and hybrid can make mobile applications for the two major mobile operating systems.

The different methods under these three categories are considered in chapter 3, and their advantages and disadvantages are discussed

It was imperative to build two mobile applications that do the same thing in order to demonstrate the capabilities of the different methods. One was built with native methodology, and the other built with React Native, a cross-platform methodology. That allowed for comparing the results to see any noticeable performance discrepancies in a general-purpose application. The appearance of the resulting applications was identical, and neither had any noticeable lags. The application development was compared in terms of the architecture of the system and the number of lines needed to achieve some parts of the application. The results showed that for the things considered, the detail view and the scrollable list, the native Android version required more lines of written code to achieve.

Tests were conducted on the mobile applications built with both methods. Despite the React Native version eating up a lot more memory than the native Android version, the memory footprint of both applications was inconsequential since it was always less than 4% of available device memory. On the contrary, the CPU load of both applications were more evenly matched as the difference between the CPU loads were insignificant, with React Native having a higher CPU load in one test scenario and native Android having a higher load in another scenario. These results demonstrate that a cross-platform application can be as capable as a native mobile application, even if not as memory efficient. Thus, businesses, organisations and decision-makers can choose either a cross-platform or native mobile approach depending on their needs knowing how performant they both are and their advantages and disadvantages.

Further research could be done on some facets of this subject. The memory consumption could be further studied with a React Native application that is not built with Expo to see the difference in memory consumption. The effects of animations on the CPU and memory could also be further analysed with an application that utilises more animations. Finally, the power consumption effects of native and non-native applications need further probing.

6 References

1 Business Insider. Why Smartphones are the Best-Selling Gadgets in History [Internet] 2015 [cited 29 September 2020]. Available at: https://www.businessinsider.com/why-smartphones-are-the-best-selling-gadgets-in-history-2015-2?r=US&IR=T

2 App Annie Inc. State of Mobile in 2020 Report [Internet] 2020 [cited 28 September 2020]. Available (upon request) at https://www.appannie.com/en/go/state-of-mobile-2020/

3 Statista. Mobile operating systems' market share worldwide from January 2012 to July 2020 [Internet]. 2020. Available from: https://www-statista-com.ezproxy.metropolia.fi/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/

4 McWherter J, Gowell S. Professional mobile application development. 1st ed. Indianapolis, Ind.: John Wiley & Sons, Incorporated; 2012.

5 Rahul Raj CP, Tolety SB. A Study on Approaches to Build Cross-Platform Mobile Applications and Criteria to Select Appropriate Approach. IEEE [Internet]. 2012 [Cited 29 September 2020]. Available from https://ieeexplore.ieee.org/document/6420693 DOI: 10.1109/INDCON.2012.6420693.

6 Nunkesser R. Beyond Web/Native/Hybrid: A New Taxonomy for Mobile App Development. 5th International Conference on Mobile Software Engineering and Systems [Internet]. New York, NY, USA: Association for Computing Machinery; 2018 [cited 6 October 2020]. p. 214-218. Available from: https://doi.org/10.1145/3197231.3197260

7 Xanthopoulos S, Xinogalos S. A Comparative Analysis of Cross-platform Development Approaches for Mobile Applications. ACM International Conference Proceeding Series [Internet]. 2013 [Cited 29 September 2020]. Available from: DOI: 10.1145/2490257.2490292.

8 Definition of native mobile app [Internet]. PCMAG. [cited 4 October 2020]. Available from: https://www.pcmag.com/encyclopedia/term/native-mobile-app

9 Key Approaches to Mobile Development Explained [Internet]. AltexSoft. [cited 6 October 2020]. Available from: https://www.altexsoft.com/blog/mobile/key-approaches-to-mobile-development-explained/

10 ComScore Inc. The Global State of Mobile [Internet]. 2019 [cited 28 September 2020] Available (upon request) at https://www.comscore.com/Insights/Presentations-and-Whitepapers/2019/Global-State-of-Mobile

11 Block R. Steve Jobs keynote live from WWDC 2008. Engadget [Internet]. 2008 [cited 5 October 2020];. Available from: https://www.engadget.com/2008-06-09-steve-jobs-keynote-live-from-wwdc-2008.html

12 StatCounter. Desktop vs Mobile vs Tablet Market Share Worldwide [Internet]. 2020 [cited 5 October 2020]. Available from: https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/worldwide/#monthly-201101-202009

13 Wakabayashi D. Google Demands 30% Cut From App Developers in Its Play Store. The New York Times [Internet]. 2020 [cited 5 October 2020];. Available from: https://www.nytimes.com/2020/09/28/technology/google-play-store-30-percent.html

14 Statista. Number of smartphone users worldwide from 2016 to 2021 [Internet]. 2019 [cited 5 October 2020]. Available from: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

15 Computer vs. Smartphone [Internet]. Computerhope.com. [cited 5 October 2020]. Available from: https://www.computerhope.com/issues/ch001398.htm

16 Definition of native application [Internet]. PCMAG. [cited 4 October 2020]. Available from: https://www.pcmag.com/encyclopedia/term/native-application

17 The Verge. The 22 most important things Apple announced at WWDC 2014. [Internet]. 2014 [cited 4 October 2020] Available from: https://www.theverge.com/2014/6/2/5765048/apple-wwdc-2014-os-x-yosemite-ios-8-and-all-the-news-you-need-to-know

18 Lardinois F. Kotlin is now Google's preferred language for Android app development [Internet]. TechCrunch. 2019 [cited 4 October 2020]. Available from:

https://techcrunch.com/2019/05/07/kotlin-is-now-googles-preferred-language-for-android-app-development/

19 Bosnic S, Papp I, Novak S. The development of hybrid mobile applications with Apache Cordova. 24th Telecommunications Forum (TELFOR) [Internet]. 2016 [cited 4 October 2020] pp1-4. Available from: https://ieeexplore.ieee.org/document/7818919/

20 McCracken H. Lyft Goes Swift: How (And Why) It Rewrote Its App From Scratch In Apple's New Language. Fast Company [Internet]. 2015 [cited 6 October 2020];. Available from: https://www.fastcompany.com/3050266/lyft-goes-swift-how-and-why-it-rewrote-its-app-from-scratch-in-apples-new-lang

21 Flauzino M, Veríssimo J, Terra R, Cirilo E, Durelli V, Durelli R. Are you Still Smelling it?. ACM - Proceedings of the VII Brazilian Symposium on Software Components, Architectures, and Reuse [Internet]. 2018 [cited 8 October 2020];:23–32. Available from: https://doi.org/10.1145/3267183.3267186

22 Android Studio Release Notes [Internet]. Android Developers. 2020 [cited 7 October 2020]. Available from: https://developer.android.com/studio/features/

23 Charland A, LeRoux B. Mobile Application Development: Web vs. Native. Queue [Internet]. 2011 [cited 3 October 2020];9(4):20-28. Available from: https://dl.acm.org/doi/10.1145/1966989.1968203

24 Update for Customers Using PhoneGap and PhoneGap Build [Internet]. Adobe Phonegap Blog. 2020 [cited 7 October 2020]. Available from: https://blog.phonegap.com/update-for-customers-using-phonegap-and-phonegap-build-cc701c77502c

25 Cheng F. Build Mobile Apps with Ionic 4 and Firebase. Berkeley, CA: Apress; 2018. Chapter 1

26 Hansson N, Vidhall T. Effects on performance and usability for cross-platform application development using React Native [Masters]. Linköpings universitet; 2021.

27 Eisenman B. Learning React Native. O'Reilly Media; 2015. Chapter 1

28 The Good and The Bad of Xamarin Mobile Development [Internet]. AltexSoft. 2019 [cited 8 October 2020]. Available from: https://www.altexsoft.com/blog/mobile/pros-and-cons-of-xamarin-vs-native/

29 Leler W. What's Revolutionary about Flutter | Hacker Noon [Internet]. Hackernoon.com. 2017 [cited 8 October 2020]. Available from: https://hackernoon.com/whats-revolutionary-about-flutter-946915b09514

30 Flutter Documentation. Android Studio and IntelliJ [Internet]. Flutter.dev. [cited 8 October 2020]. Available from: https://flutter.dev/docs/development/tools/android-studio

31 Build Apps for Free on Heroku [Internet]. Heroku.com. [cited 26 May 2021]. Available from: https://www.heroku.com/free

32 WebView | Android Developers [Internet]. Android Developers. [cited 26 May 2021]. Available from: https://developer.android.com/reference/android/webkit/WebView

33 FlatList | React Native 0.64 [Internet]. Reactnative.dev. [cited 27 May 2021]. Available from: https://reactnative.dev/docs/flatlist

34 Create dynamic lists with RecyclerView [Internet]. Android Developers. [cited 27 May 2021]. Available from: https://developer.android.com/guide/topics/ui/layout/recyclerview

35 Inspect CPU activity with CPU Profiler [Internet]. Android Developers. 2021 [cit-ed 7 June 2021]. Available from: https://developer.android.com/studio/profile/cpu-profiler

36 Better performance through threading [Internet]. Android Developers. 2021 [cited 7 June 2021]. Available from: https://developer.android.com/topic/performance/threads

37 dumpsys [Internet]. Android Developers. 2021 [cited 6 June 2021]. Available from: https://developer.android.com/studio/command-line/dumpsys

38 Memory allocation among processes [Internet]. Android Developers. 2021 [cited 6 June 2021]. Available from: https://developer.android.com/topic/performance/memory-management

39 What is Apptim? [Internet]. Help.apptim.com. 2021 [cited 6 June 2021]. Available from: https://help.apptim.com/en/articles/2965623-what-is-apptim

40 Introduction to Expo - Expo Documentation [Internet]. Expo Documentation. 2021 [cited 7 June 2021]. Available from: https://docs.expo.io

41 Expo - App size in the managed workflow [Internet]. Expo’s GitHub. 2021 [cited 7 June 2021]. Available from: https://github.com/expo/fyi/blob/master/managed-app-size.md

42 Manage your app’s memory [Internet]. Android Developers. 2021 [cited 7 June 2021]. Available from: https://developer.android.com/topic/performance/memory

43 Reduce your app size [Internet]. Android Developers. 2021 [cited 7 June 2021]. Available from: https://developer.android.com/topic/performance/reduce-apk-size

React Native application - https://github.com/pokumars/metropolia-mixer-rn

Native android application - https://github.com/pokumars/metropolia-mixer-kt