DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2018

Emulating a Native Mobile Experience with Cross-platform Applications

RASMUS FREDRIKSON

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

KTH Computer Science
and Communication

Emulating a Native Mobile Experience with Cross-platform Applications

Emulering av en naturlig mobil-upplevelse med plattformsoberoende applikationer

RASMUS FREDRIKSON

Master in Computer Science, DA222X
Supervisor: Pedro Afonso Nunes Sanches
Examiner: Eva-Lotta Sallnäs Pysander

EECS, KTH. Stockholm, Sweden. June 05, 2018.

Abstract

This thesis compares a native Android application with two different classes of cross-platform applications, an interpreted application developed in React Native and a Progressive Web-App (PWA). The main contribution of the thesis is a comparison table, placing application features on the y-axis and the evaluated frameworks on the x-axis. This table in conjunction with the cost analysis provide clear framework selection guidance. The evaluated applications were created to evaluate the frameworks' fulfillment of the enumerated features. A user study was performed to learn if there was any discernible difference between the evaluated applications. Qualitative data obtained from a think-aloud in the user study, indicates a slight preference for the PWA, despite a smaller feature set. However, quantitative analysis from a User Experience Questionnaire (UEQ) failed to show any systematic UX performance difference over the evaluated applications. Therefore, it is concluded that cross-platform applications are able to both emulate and even outperform a native experience in some regards, with the added advantage of significantly offsetting both development and maintenance costs. Finally, we recommend using a cross-platform mobile application framework if it provides all the features required by the intended application.

Sammanfattning

Denna rapport jämför en “Native Android”-applikation med två olika klasser av plattformsoberoende applikationer, en “Interpreted”-applikation utvecklad i React Native och en “Progressive Web”-Applikation (PWA). Det största bidraget från denna rapport är en jämförelsetabell, där applikationsfunktioner placeras på y-axeln och de utvärderade ramverken på x-axeln. Denna tabell, i konjunktion med en kostnadsanalys, förser läsaren med en tydlig guide vid val av ramverk. De utvärderade applikationerna skapades för att utvärdera ramverkens tillgänglighet till de uppräknade funktionerna. En användarstudie utfördes för att utreda huruvida det existerade någon märkbar skillnad mellan de utvärderade applikationerna. De kvalitativa data som erhölls från en “think-aloud” i användarstudien indikerar en liten preferens för PWA:n, trots att den har tillgång till färre funktioner. Den kvantitativa analysen från ett “User Experience Questionnaire” (UEQ) misslyckades med att visa någon systematisk skillnad i UX mellan de utvärderade applikationerna. Slutsatsen är därför att plattformsoberoende applikationer både kan emulera, och till och med överträffa, en naturlig upplevelse i vissa avseenden, med en ytterligare fördel av att både utvecklings- och underhållskostnader väsentligt minskar. Slutligen rekommenderar vi användandet av ett plattformsoberoende applikationsramverk förutsatt att det har tillgång till alla funktioner som krävs för den avsedda applikationen.

Emulating a Native Mobile Experience with Cross-platform Applications

Rasmus Fredrikson
KTH - Royal Institute of Technology
Stockholm, Sweden
rasmuf@kth.se

ABSTRACT

This thesis compares a native Android application with two different classes of cross-platform applications, an interpreted application developed in React Native and a Progressive Web-Application (PWA). The main contribution of the thesis is a comparison table, placing application features on the y-axis and the evaluated frameworks on the x-axis. This table in conjunction with the cost analysis provide clear framework selection guidance. The evaluated applications were created to evaluate the frameworks' fulfillment of the enumerated features. A user study was performed to learn if there was any discernible difference between the evaluated applications. Qualitative data obtained from a think-aloud in the user study, indicates a slight preference for the PWA, despite a smaller feature set. However, quantitative analysis from a User Experience Questionnaire (UEQ) failed to show any systematic UX performance difference over the evaluated applications. Therefore, it is concluded that cross-platform applications are able to both emulate and even outperform a native experience in some regards, with the added advantage of significantly offsetting both development and maintenance costs. Finally, we recommend using a cross-platform mobile application framework if it provides all the features required by the intended application.

ACM Classification Keywords

H.5.m. Information Interfaces and Presentation (e.g. HCI): Miscellaneous; See http://acm.org/about/class/1998/ for the full list of ACM classifiers. This section is required.

Author Keywords

Cross-platform application; React Native; Progressive Web Application; UEQ; Android.

INTRODUCTION

Today many companies are struggling with the maintenance of two codebases for their mobile applications, iOS and Android. Due to this problem, cross-platform development is increasingly used at companies looking for ways to reuse their codebase to lower both development costs as well as the total cost of ownership drastically. These companies will be interested by the results of examining and comparing different cross-platform techniques. The gains of only having one shared codebase is reduced maintenance time while making it easier to find and fix bugs, since they only need to be fixed once. Furthermore, only one programming language and framework need to be used, which makes it easier to find and hire developers while keeping down the amount of teams at the company.

Xanthopoulos and Xinogalos [20] compare four different classes of cross-platform mobile application frameworks: web, hybrid, interpreted and cross-compiled. Since their publication, the frameworks evaluated have been improved and new ones have been added to the aforementioned classes. Therefore, a new comparison is relevant where this study will expand on their study by taking new frameworks into consideration. In order to conclude this study we will attempt to answer the following research questions:

- How close can a cross-platform application emulate a native experience?
- What set of features can effectively describe the capabilities needed for a mobile application framework to be able to facilitate a native experience?
- Out of the evaluated frameworks, how do they compare as to the fulfillment of above features?

THEORY AND RELATED RESEARCH

The four different framework classes mentioned in the previous section are explained in detail in this section. Other user-studies carried out prior to this study which compare different cross-platform applications are also described.

Native applications

A native application is an application developed directly for a certain mobile Operating System (OS) e.g. iOS or Android. When developing for iOS the code is written in Objective C or Swift while Android is written in Java or Kotlin [4]. Since the applications are written in two different programming languages, two completely separate applications need to be developed. The applications however are thereby optimized

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

to their respective OS and can access all of the native features described in the corresponding native API, presuming the user allows it.

Web applications

Web applications use the mobile phone's web browser as its runtime environment and are implemented using JavaScript (JS), Cascading Style Sheets (CSS) and Hypertext Markup Language (HTML). Due to this web applications are never installed on the device and therefore no updates are needed to be shipped to the device. However, Internet access is needed to use such applications and due to the visible browser artifacts, such as the address bar, they lack a native look and feel. Other drawbacks with web applications are their restricted access to the mobile device's native features and their slow rendering of components compared to native applications [10].

Progressive Web-Applications (PWAs)

PWA is a new concept which expand on the web applications and effectively deal with their drawbacks. Thanks to PWAs and new concepts and requirements advocated by Google [14] regular websites can now act, look and feel more like an installed application, especially on Android devices. With the help of web-app manifests, web-sites in PWA supported browsers can be added to the user's device's home screen, where the browser artifacts have been stripped from the application. By implementing a background JS-script (Service Worker) logic and background tasks can be cached [14], thereby allowing the application to work offline. Since the application's data is cached offline, the application can render its GUI faster than an ordinary web application, which needs to fetch its data from the Internet first [14].

Not much research has been done on PWAs so far, due to their short time on the market [14]. Biørn-Hansen, Majchrzak and Grønli [5] compare a PWA implemented in ReactJS with an interpreted application in React Native and a hybrid application in Ionic. They compare each application's installation size, start-up time and first activity launch-time. The PWA had the smallest installation size, the shortest launch-time and second lowest start-up-time of the three applications. The authors reckon that PWAs could be a potential solution for web-native development without using cross-platform frameworks.

Fransson and Driaguine [7] created a PWA in ReactJS and compared it with a native Android application. Since a common problem with web applications before was their inability to access the mobile device's hardware, they wanted to investigate whether this was still a problem with the introduction of PWAs. They realized that this was no longer an issue and decided to compare the PWA's performance of the camera and geolocation with the Android application. It showed that the PWA's geolocation was faster than the Android application's geolocation, whereas the opposite was true regarding the camera.

Several researchers [5, 7, 14] recommend that further research should be done due to the lack of scientific research within the area. Furthermore no User Experience (UX) research has been done concerning PWAs as of now and should therefore be an interesting contribution to the field.

Hybrid applications

Hybrid applications are a mix between native and web applications and are packaged and installed on the mobile device like a normal native application. However instead of being developed in a native language they are built using web technologies, where the application creates web-views employed on the standard OS's design, to make it look and feel native. The views are packaged into the native binary along with the hybrid framework, where the framework then renders the views into a web-view at runtime.

The pros and cons are similar to the web applications, however an Internet connection is no longer needed for the application to work. The rendering of hybrid applications' web-views are faster than web applications, but still slower than native applications' component rendering [10]. Examples of hybrid frameworks are PhoneGap and Ionic [18].

Much research has already been done regarding hybrid applications and most new articles evaluates Ionic [5, 13, 18, 19]. Ionic uses Cordova plugins to gain access to native features like notifications and file storage and are more focused on performance than other hybrid solutions [10].

Interpreted applications

Interpreted applications use a technique which renders the native components directly instead of creating a web-view like the hybrid applications. Interpreted applications are developed with a common language, such as JS and uses an abstract layer which translates the common language code at runtime to access the native features of the mobile device. By rendering native components these applications create a native user-interface, however they rely completely on the framework and if new native features are created the framework's features need to be updated as well [10]. React Native, NativeScript and Titanium are examples of interpreted frameworks. React Native is as of now the most popular interpreted framework on GitHub with over 64 000 stars and more than 14 000 forks[1].

There are several scientific articles evaluating interpreted applications using React Native [3, 5, 8, 9, 17], but only two of those [3, 8] evaluate them from a UX perspective. Both of these articles consider React Native as a potential game-changer within cross-platform frameworks since it both looks and feels similar to native applications. However as both Hansson and Vidhall [8] and Axelsson and Carlström [3] state in their discussion, the framework was still young when the articles were published and has since then updated frequently. It is therefore possible that the framework has gone through such changes that the articles' results may no longer be of relevance. Furthermore, Hansson and Vidhall [8] recommend a more detailed user-study with more test subjects together with a think-aloud or a longitudinal study since their study could not statistically confirm their UX-results.

Cross-compiled applications

Cross-compiled applications are written in a common, usually statically typed and compiled application programming language and then transformed into targeted native code by

[1] https://github.com/facebook/react-native

a cross compiler [10]. Since the code is retargeted to use the native UI toolkit the performance of the application is equal to that of a native application as well as the look and feel. It also has access to most of the native features except for platform dependent features such as the camera and geolocation since the access differs between platforms [10]. A downside with this approach is that it is focusing on mobile development and cannot easily be transformed into a website. This approach is more costly for the many companies following a web-first approach when developing, due to the need of developing two separate codebases, one for the web and one for the mobile application. Xamarin is a popular framework for cross-compiled applications [10].

Delimitations

Due to time constraints not all of the previous mentioned application classes were implemented. This work aims to compare applications able to share code between themselves as well as the web. Therefore cross-compiled applications were not included in this study, since they have no connection to web development. Since a lot of UX research has already been done regarding hybrid applications, such applications will not be included in this study either.

Due to lack of recent UX-research regarding PWAs and interpreted applications this project aimed to compare a PWA's UX with an interpreted and native application. The frameworks' accessibility to a mobile device's hardware was also investigated. A native application was developed as a baseline when comparing the cross-platform applications. Android was chosen as the native application, since at the start of this project the iOS platform did not support PWAs. The interpreted application was developed in React Native due to its wide popularity. To make the comparison easier between the PWA and the React Native application, the PWA was developed in ReactJS to make sure the programming paradigm and execution model did not affect the comparison between the two cross-platform applications.

Related work

The comparison of cross-platform applications' UX have been approached rather differently during the years. Some have conducted longitudinal studies and some used questionnaires.

Longitudinal studies

Angulo and Ferre [2] performed a laboratory study where they compared a hybrid application created in Titanium with a native application. They created both an iOS and an Android version of the application where the users tried either version. After the laboratory study they carried out a longitudinal study where half of the users started with the native application and the other half the cross-platform application and then switched after five days. They concluded that a good level of UX could be obtained by using a cross-platform application if the development framework is chosen carefully.

Similarly, Andrade et al. [1] carried out a longitudinal study where they compared a native application with a hybrid application developed in Phonegap. A number of their test subjects used the native application and the rest tested the Phonegap application. After a few days they told their test subjects that they had updated their application and that they should try them for a few more days. The updated version was a switch between the two applications depending on which application the user had used from the beginning. However, the authors only installed the other application on some of the devices, thereby letting some of the test subjects keep the same application without knowing it. This was to avoid a tainted test result since the test subjects now thought they had got a new version. The result was that only eight out of sixty noticed a difference between the applications.

Questionnaire studies

Hansson and Vidhall [8] used questionnaires to compare an interpreted application in React Native with a native application. Their test subjects carried out their assigned tasks and then completed a User Experience Questionnaire (UEQ) afterwards. They discussed the use of System Usability Scale (SUS) but chose to use a UEQ since it was easier to use when comparing two products. Their users tried either the React Native application or the native one. Their study showed that React Native had similar user experience compared to its native correspondent.

Axelsson and Carlström [3] performed a similar study, however they made use of a cognitive study while performing their user-study and then let the user complete a Single Ease Question (SEQ) and a SUS after performing their assigned task. They based most of their UX method on a paper by Tom Albert and Bill Tullis recommended by their supervisor. Furthermore, they started the project with a survey where people answered whether they preferred mobile applications or websites and why. Their final conclusion was that React Native was not as good as its native correspondent but demonstrates well the capabilities of cross-platform applications.

The UEQ used in the study by Hansson and Vidhall [8] was created by Laugwitz, Schrepp, and Held [11, 12] and focuses on measuring UX quickly in a simple and immediate way while still being comprehensive. The questionnaire was created with the help of usability experts who gave their opinion on which aspects were most important to investigate when conducting quantitative UX research. The questionnaire consists of 26 items were each item is categorized into one out of six categories. These categories are attractiveness, perspicuity, efficiency, dependability, stimulation and novelty and were chosen to best represent a product's overall UX. Each item in the UEQ is represented by two terms with opposite meaning and are on the opposite side of each other on a seven-stage scale. The order of terms is randomized for each item, where half of the negative terms are on the left side of the scale and the other half on the right side. This is to make sure the user stays alert during the entirety of the UEQ. The authors recommend that the questionnaire is complemented with a qualitative study to make the study fully comprehensive.

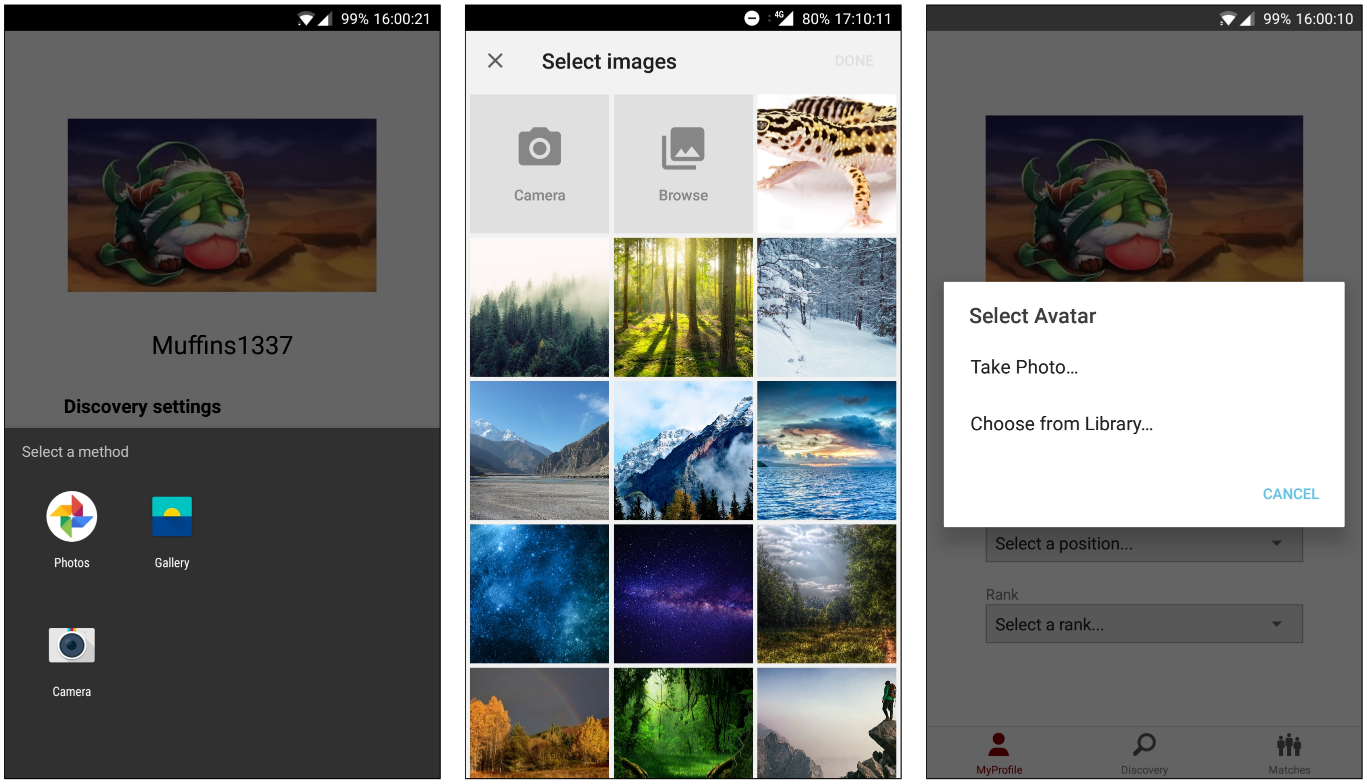
PRE-STUDY

To answer the second research question, a comparison table was created which attempts to span the space of mobile features with focus on the most used native features. This table was created to easily compare which native features each framework can access and thereby giving parties interested in

Features	Android	PWA	React Native
Native Behaviors			
Local Notifications	✓	✓	✓
Push Messages	✓	✓	✓
Home Screen Installation	✓(T)	✓(T)	✓(T)
Foreground Detection	✓	✓	✓
Permissions	✓	✓	✓
Surroundings			
Bluetooth	✓	✓*	3rd
USB	✓	✓	3rd
NFC	✓	X	3rd
Ambient Light	✓	X	✓
Device Features			
Network Type & Speed	✓	✓*	✓
Online State	✓	✓	✓
Vibration	✓(T)	✓(T)	✓(T)
Battery Status	✓	✓	3rd
Device Memory	✓	✓	3rd
Camera & Microphone			
Audio & Video Capture	✓(T, camera)	✓(T, camera)	✓(T, camera)
Advanced Camera Controls	✓	✓	3rd
Recording Media	✓	✓	3rd
Real-time Communication	✓	✓	3rd
Operating System			
Offline Storage	✓(T, Shared Preferences)	✓(T, Local Storage)	✓(T, Async Storage)
File Access	✓(T)	✓(T)	✓(T)
Contacts	✓	X (No longer in development)	✓
SMS	✓	X	✓
Storage Quotas	✓	✓(Only an estimate)	3rd
Task Scheduling	✓	X (No longer in development)	3rd
Input			
Touch Gestures	✓(T, swipe/touch)	✓(T, swipe/touch)	✓(T, swipe/touch)
Speech Recognition	✓	✓*	3rd
Clipboard (Copy & Paste)	✓	✓*	✓
Seamless Experience			
Offline Mode	✓(T)	✓(T, Service Worker)	✓(T)
Background Sync	✓	✓	3rd
Inter-App Communication	✓	✓(Possible with web share)	✓
Payments	✓	✓	3rd
Credentials	✓	✓	3rd
Location & Position			
Geolocation	✓	✓*	✓
Geofencing	✓	X (No longer in development)	3rd
Device Position	✓	✓*	3rd
Device Motion	✓	✓*	3rd
Proximity Sensors	✓	X (No longer in development)	3rd
Screen & Output			
Virtual & Augmented Reality	✓	✓*	3rd
Fullscreen	✓	✓*	3rd
Screen Orientation & Lock	✓(T)	✓(T, not possible to lock)	3rd (T)
Wake Lock	✓	X*	3rd
Presentation Features	✓	✓	X

Table 1. Comparison table

(T): Implemented in the developed applications.
3rd: Third-party solution exists.
✓*: Partially accessible.
X* Possible, but too easy to abuse, therefore not implemented.



(a) Android application. (b) Progressive Web Application. (c) React Native application.

Figure 1. The profile-picture selection menu in the developed applications.

application development a good overview over which framework might suit them best, depending on which features are crucial for their application.

To present the features a table structure was chosen, mainly because it gives a good overview of each application's feature accessibility where comparison of the applications remains easy. Several others have also chosen this structure [1, 3, 5, 14, 15], however the created table in this study is more extensive than those tables. To find which native features were most common in today's mobile applications, seven applications were explored. The selected applications were Facebook, Google Maps, Instagram, Slack, Snapchat, Tinder and Tink and were chosen due to their wide use in the mobile community and their difference compared to each other, see Appendix B. The table got extended further by exploring each framework's API[2][3] and a detailed summary[4] of the PWA's access to native features.

Table 1 summarizes the most important native features common in most popular applications today. As seen in the table Android has access to all of the mobile device's native features. React Native lacks access to the presentation feature, which allows users to connect their application to an external device such as a monitor. The React Native API however does not support all of the native features as of now and instead third-party solutions have been created to fill most of this gap. Links to the third-party sources that access the specified features in the table can be found in Appendix A. The PWA lacks access to a few of the native features and have only partially integrated a few others, but still have access to most of them. Each feature is explained in detail in the summary[4] previously mentioned.

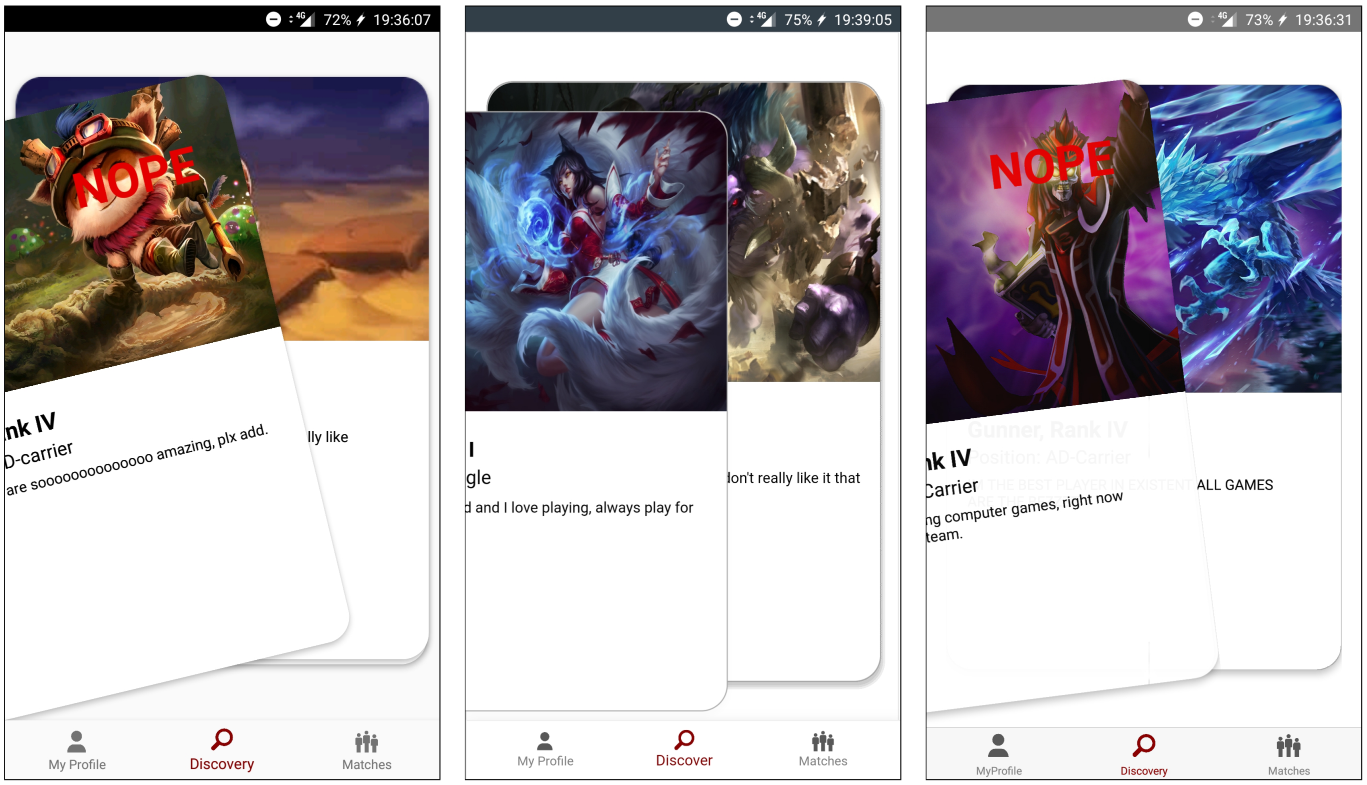
DEVELOPED APPLICATIONS

Three applications were also developed, one native in Android, one interpreted application in React Native and one PWA in ReactJS. They were developed to look as similar to each other as possible while still incorporating the most common components based on the previously mentioned popular applications. These developed applications acted as a complement to the comparison table to be able to decide whether there were any subjective differences between the applications.

Selection and implementation of features

Based on the comparison table eight of the native features were chosen to be implemented in the developed applications to see how they performed and how hard they were to implement. The selected features are marked with a (T) in Table 1. These applications were however mainly developed to discover whether or not people noticed if they were using a native or a cross-platform application. Due to this, it was important to make sure that all aspects of native components were

[2]https://facebook.github.io/react-native/docs/components-and-apis.html
[3]https://developer.android.com/reference/classes.html
[4]https://whatwebcando.today/



(a) Android application. (b) Progressive Web Application. (c) React Native application.

Figure 2. The developed applications' Discovery views.

tested as well. Therefore, an investigation of seven widely used applications was conducted, where the main components used in these applications were: 1) Navigation bar, either top, side or bottom. 2) Scroll functionality. 3) Listview. 4) Go back-functionality. 5) Swipe around elements. 6) Select. 7) Text-input. 8) Button. 9) Switch. 10) Checkbox. 11) Calendar-picker. 12) Clock-picker. 13) Seekbar. 14) Lightbox.

One of the investigated applications found in Appendix B was Tinder, an application where people meet others by swiping right on those profile pictures that they want to match with and left on those they do not. Tinder incorporated all of the eight chosen native features from the comparison table, while still using the most common components described in the list above. Due to these facts, Tinder was chosen as a template for the developed applications in this study. The developed applications incorporated the same main functionality as Tinder, but instead targets the gaming community. The application's goal is to make it easier for people to find new friends or team members to play games with. The name of the application thus came to be Teamfinder.

All of the native components found in Appendix B were implemented in the developed applications with the exception of some of the more unusual components such as the seekbar, calendar-picker and clock-picker. The checkbox and switch were not implemented either since they already exist on many places on the web and had no obvious place in the developed application. With specific CSS[5] these components can look native since it is rather a question of design than interaction and overall feeling. Instead focus was put into making the different applications behave native overall with natural interaction and navigation. A bottom navigation was chosen since a majority of the investigated applications employed one.

The native features chosen from the comparison table were implemented according to each framework's API. They all worked as expected, except for the screen orientation lock in the PWA which was not supported on the selected device, however it was still possible to get the screen orientation of the device. The source code can be found at GitHub[6].

Teamfinder

There are four views in the application Teamfinder: "My profile", "Discovery", "Matches" and "Matched player". 30 test subjects tested the three applications and gave their opinion on each.

My profile

In "My profile" the user is able to change her profile picture and nickname by tapping them. The player then chooses whether she wants to use an existing image on the mobile device or take a new one with the camera. This menu differs between the three applications as seen in Figure 1. In Android

[5]https://github.com/styled-components/styled-components
[6]https://github.com/rasmusfredrikson/teamfinder

the menu is black and stays at the bottom of the screen with three options: Photos, Gallery and Camera. In the PWA the user is immediately shown all images the phone has access to with the camera option in the upper left corner. In React Native the user may choose between the Camera and the Gallery represented by a white lightbox menu centered on the screen. Five people thought the Android application was the best application partly because of this selection menu, two people chose React Native due to this and three people chose the PWA. This menu was also the main reason why people thought Android was the native application, where a total of eight out of eleven people were able to guess it solely on this factor.

The interaction when editing the nickname is the same on all applications with the small anomaly that the Android's keyboard was not shift sensitive. One person noticed this bug and therefore thought a lot worse about the Android application which might have affected the person's overall opinion.

The user also has the possibility to filter which players she wants to match with. In the application there are three dropdown settings for the user to choose amongst: Game, Position and Rank. The dropdown design is a bit different between the three applications. In the PWA and React Native a lightbox pops up with all the different alternatives whereas in the Android there is just an ordinary dropdown. No clear consensus could be found on which dropdown looked best in the qualitative study. Two people thought the dropdown in the PWA looked most native, while one user thought the Android dropdown had the most native design.

Discovery

The Discovery view as shown in Figure 2 is where the main action takes place. Here the user gets to swipe yes (right) or no (left) for each player the user sees. A match will occur if both the user and the opposite player swipe right. These players are just mock data for now and therefore a simple random function was implemented to decide whether or not the user matched with the player. A vibration is felt when the user gets a match. The swipe was a bit different between all the applications. The "Nope" and "Like" that appeared when people liked or disliked a person showed directly after the user started moving the card in the Android application. In React Native it showed when reaching a certain point on either side of the screen, and in the PWA it appeared after the user had swiped the card. The card wiggled a lot in the Android version, whereas in React Native it only tilted the direction the user moved the card and in the PWA it was completely static in its movements. The users had different opinions on which application had the best swiping, where five users thought the PWA was best due to the swiping, two thought React Native and one user thought Android.

Matches and Matched player

In the Matches view the user can see all of her matches and scroll if there are more than five. The user also has the possibility to click on a matched player to get a more detailed view of the player. This view represents the last view: Matched player. From this view it is possible to go back to the previous view by tapping the back button in the upper left corner. Both views were very similar between the applications and no user noticed any real difference here.

USER-EVALUATION

Method

The first research question was answered with the help of a quantitative study and complemented with a qualitative study as suggested by Hansson and Vidhall [8] and Laugwitz, Schrepp, and Held [12] to make the study as comprehensive as possible. 30 test subjects took part in the study and were separated into six test groups where each group tested a different order of the applications than the one before them. This was to avoid the study's result being affected by the ordering of the applications rather than by the applications' performances.

A UEQ was used as the quantitative method due to it being fairly comprehensive while still being a quick method to evaluate the test subjects. As suggested by Hansson and Vidhall [8] a hint was added to the item "secure" and "not secure" since it was common that this item was misinterpreted as "how safe the user's data was" instead of the more correct interpretation "how safe and controllable the interaction is" [12]. The test subjects completed the UEQ after they had performed the assigned tasks for each application. The tasks and their order are listed below.

- Task 1: Change your profile-picture and nickname.
- Task 2: Go to the discovery view and swipe through the card-deck until you reach the end.
- Task 3: Change all of your discovery settings to only find players who play League of Legends with a rank and position of your choice. Match with at least three of these players and keep swiping until you reach the end.
- Task 4: Find out more about your first and last matched player.

To complement the quantitative method a think-aloud session took place while the test subjects performed their task. This was to understand why they were doing certain things and what they were looking for. The users did not know that they were testing cross-platform applications, but instead just thought they were trying out three different versions of the same application. After testing all of the applications the users were asked which application they liked best and why. After answering that question, it was revealed to them that two of the application were cross-platform and were then asked which one they thought was native and why. The think-aloud was transcribed and the essence of it was summarized into a graph found in the next section.

Result

The quantitative data from the UEQ is summarized in Figure 3 and 4 where the mean values for each of the six categories for each application is presented. In Figure 3 the applications are compared to the UEQ's benchmark values. Aside from the novelty category all applications perform above average or better. In Figure 4 a confidence interval for each category was calculated [6, Chapter 12.3] and is represented by the error bars in the graph.

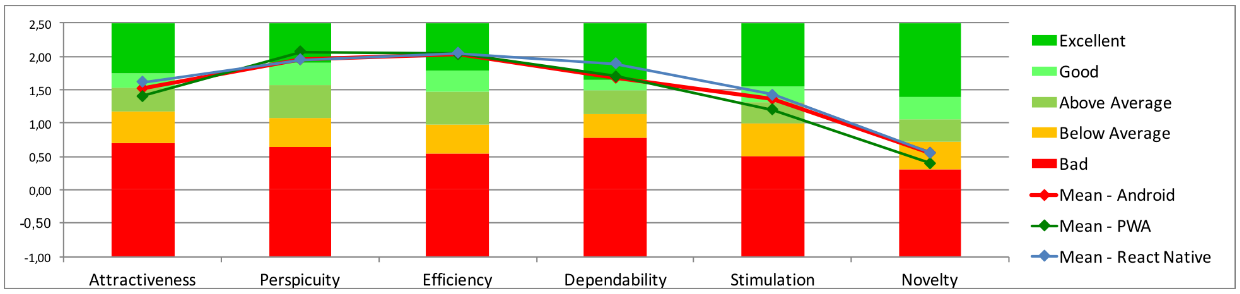


Figure 3. The developed applications in comparison to the UEQ's benchmark values.

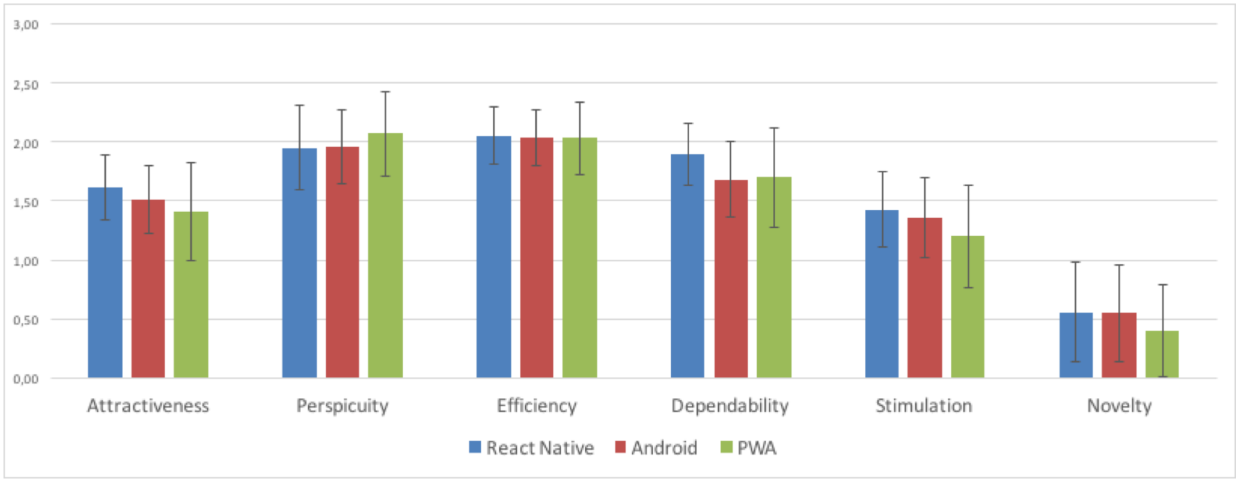


Figure 4. Result from the quantitative study.

Categories	**React Native vs Android**	**React Native vs PWA**	**Android vs PWA**
Attractiveness	0.2843	0.1349	0.4367
Perspicuity	0.9406	0.1946	0.3313
Efficiency	0.9015	0.9088	1.0000
Dependability	0.1398	0.2489	0.8903
Stimulation	0.4888	0.1096	0.2366
Novelty	0.9263	0.1081	0.1217

Table 2. Two-sided paired t-test values. T-value less than 0.05 **indicates a significant systematic difference.**

A t-test was performed to test if a significant difference existed between any of the applications' categories. The samples from each test subject are not pairwise independent whereas the mean value for each category could be either higher or lower than another test subject's category sample. Due to these facts a two-sided paired t-test was used to calculate whether or not a significant difference exists between the applications' categories. The calculated T-value must be lower than an alpha value of 0.05 [6, Chapter 13.6] to be of any useful significance. The result from the t-test can be found in Table 2. With an alpha value of 0.05 no significant difference could be proven for any of the categories, but one could still exist.

The results from the think-aloud can be found in Figure 5. The users got to choose which application they preferred in the qualitative study as shown by the first bar group in the graph. The second bar group shows how many users preferred two applications. The best application in the quantitative study for each user was chosen as the one which performed best in most categories. If two applications performed equally well they were both chosen the best. If a user had the same preferred application in both the qualitative and the quantitative study, they had a consistent result as presented in the third bar group. If a user preferred two applications in the qualitative study and one or both matched with the quantitative study, the matched application was counted for in the third bar group.

The most preferred application with ten people voting for it was the PWA, with the Android and React Native applications on a tied second place with seven people voting for them each. The Android and React Native application were however each chosen three times together with another application when the user could not decide on which application was best. The PWA got chosen twice together with another application as seen in the second bar group. People were most consistent when picking the React Native as their preferred application, where seven out of ten were consistent. The Android had five out of ten consistent users and the PWA six out of twelve.

The fourth and fifth bar groups are similar to the first and second bar groups, with the difference that they represent the test subjects' opinions on which application felt most native. Eleven users thought the Android application looked most native, whereas seven thought the PWA was the native application. Only two people thought the React Native application

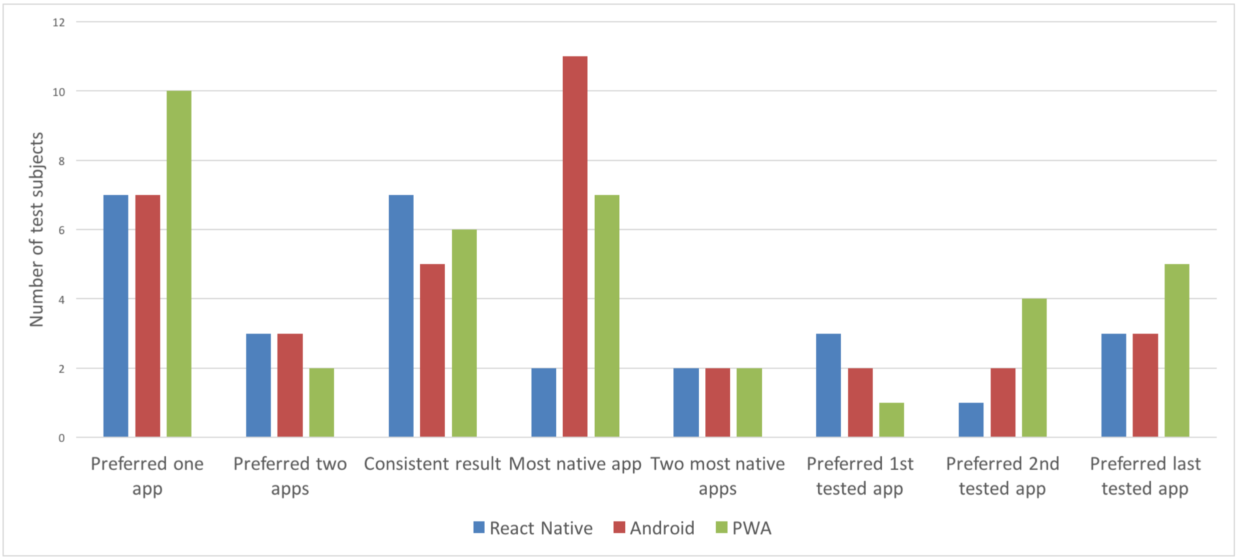


Figure 5. Result from the qualitative study.

was the native application. The Android looked most native according to eleven users, however eight of these only thought that because the selection of the profile picture looked most native. When asked if they would have guessed it was native when ignoring this selection menu, none of these eight would have been able to guess. Five of those who thought the PWA was the most native chose it due to the smoothness and overall familiar feeling of the application.

The three last bar groups tell which application was preferred most regarding to if it was tested first, second or last. The React Native application performed equally when tested first and last with four people voting for it, and only one preferred it when tested second. The Android application performed equally well when tested first and second with two users, and best when tested last with three users voting for it. Lastly the PWA performed worst when tested first where only one user preferred it, whereas much better when tested second and last where four respectively five people preferred it. This shows that all applications performed better when tested last, which might suggest that the order of the application testing is a more prominent factor than the actual difference between the applications.

Other things taken into consideration was whether the person had used Tinder before, if the user used iOS or Android and their previous Android experience. The age-span of the user was also noted as well as whether they had ever developed an application before. However none of these factors showed any correlation with their answers and were therefore not shown in Figure 5.

DISCUSSION

Comparison Table

The comparison table shown in Table 1 tries to summarize which native features each framework has access to. It is fairly extensive and covers all of the features in the investigated popular applications and more. The table helps to answer the second research question "What set of features can effectively describe the capabilities needed for a mobile application framework to be able to facilitate a native experience?"

The Android application can access all native features, whereas React Native lacks access to a rarely used one, the presentation feature. Since the presentation feature is the only feature found so far, which React Native does not have access to it suggests that React Native can emulate a native experience in terms of access to native features. However, to access some of the native extensions not supported by the framework the native code for the different platforms need to be altered. This results in a codebase not entirely cross-platform dependent, since that code differs a bit between iOS and Android. The PWA lacks access to a few features, however most of them are rarely used in today's applications. The PWA has access to the most important features such as offline mode and storage, file access etc., which makes it a likely candidate to be able to emulate a native experience in terms of native features. Unlike React Native no third-party solutions are needed to gain access to these features. The PWA has been tested on an iPhone and works as it should, however no evaluation has been done since this was not in the study's scope.

When choosing a framework it is important to make sure that the framework actually supports all of the feature that are to be implemented. If not, the company will need to reconsider whether this feature is important enough to switch framework or if they can use an alternative solution instead. Overall the comparison table shows that the most important features are accessible by all three frameworks which implies that they are all able to emulate a native application in terms of access to native features. However, if some of the more obscure features are needed then the PWA might not be up to the challenge. This comparison helps to answer the third research question "Out of the evaluated frameworks, how do they compare as to the fulfillment of above features?".

User-study

The PWA was most popular when tested in the qualitative study, but when compared with React Native in the quantitative study it performed worse in all categories except perspicuity. However, since no significant difference could be established in the quantitative study there is still a chance that the PWA is better. The qualitative study showed that those who preferred the PWA chose it due to its general feeling and smoothness. This is interesting since that is probably the hardest part to emulate and in part proves the first research question in regard of whether it is possible to emulate a native experience with cross-platform applications.

It is difficult to find a definite answer to the first research question "How close can a cross-platform application emulate a native experience?". The fact that no significant difference could be established hints that all the applications were able to emulate a native experience. Additionally, the gains of only maintaining one codebase, as mentioned in the introduction, should be enough for companies to consider switching to a cross-platform application. Taking this into consideration, the PWA should perform best since its code-sharing percentage is higher than the other platforms, due to it being developed solely in a web programming language. React Native is not completely platform independent since websites cannot be developed with React Native and if native extensions are to be accessed, the codebase will slightly differ between Android and iOS as well. However, presuming the mechanism and policy code are separated accordingly [16], the policy code which is developed in ReactJS should be consistent between all three platforms. The mechanism code will be different between the web and the mobile applications since React Native does not run on the web. The only code that should differ between iOS and Android is the code developed for native extensions. Therefore, React Native is still a viable alternative even if a website would be part of the codebase.

What is also interesting is that the actual native Android application never performed best in either the qualitative or the quantitative study. This implies that native applications do not necessarily have to be better than cross-platform ones, especially since many people guessed which one was native but still preferred a cross-platform application. This could of course also be due to the fact that each application was developed in two weeks and if further developed, greater differences might appear. However, the differences might also lessen if more time was put into making the applications more similar. A possibility would be to compare three already existing applications implemented with each framework. It would however be difficult to avoid the result being tainted by comparing applications which were not developed with the same unique selling point in mind. It is possible to argue that the applications developed in this study have not been developed by a professional and therefore cannot be used as proof that cross-platform can emulate a native experience. However as can be seen in Figure 3 all applications performed better than the UEQ's benchmark in all categories except novelty. This indicates that the usability factor is high and the result gained from this study should reflect usage in the real world.

The question stands whether this is a good way to compare such similar applications. Neither this nor Hansson and Vidhall's [8] studies were able to show any significant differences between the applications even though 30 or more users participated in each study. The reason could be that the applications are actually so similar that they do not differ in any of the categories, however more test subjects would be needed to be able to say whether or not this is the case. Laugwitz, Schrepp, and Held [12] recommend that at least 20 test subjects are recruited, therefore 30 should be sufficient for an accurate result, especially since both this study and Hansson and Vidhall's [8] show similar results. React Native was a new framework when they conducted their study and has now existed and grown for two years. React Native performs equally or better than the Android in almost all categories in this quantitative study, however no significant results could be established. They do equally well in the qualitative study. These results hint that React Native performs better now than it did when Hansson and Vidhall [8] and Axelsson and Carlström [3] performed their study in terms of UX.

Longitudinal study such as the ones Andrade et al. [1] and Angulo and Ferre [2] conducted might give more accurate results since users tend to discover more of the bugs and get a more extensive feel of the general UX. This study however aimed to see whether users noticed a difference when just using a cross-platform application for a few minutes. The qualitative data shows their conscious opinions whereas the quantitative study shows their subconscious opinions. More work needs to be put into each application to give enough material for a longitudinal study.

Future work

At the start of this study PWAs were not supported on the iOS platform, which is why iOS was not a part of the research done in this project. However, Apple recently released an update where they added support for service workers and web-app manifests[7]. These features are crucial for a PWA to work and are a good hint that PWAs might be the future of application development. An interesting study would be to investigate whether the developed applications in this study would perform equally well on the iOS platform.

The work can also be expanded upon by improving the individual applications with more information and design. Due to time constraints only two weeks of development were put into each application and therefore only the most common features were implemented. However, if a more extensive application was to be created a longitudinal study could be conducted instead which might give different results since people will grow accustomed to the applications and might find small differences not noticed upon first inspection.

CONCLUSION

To summarize this study, it seems like the cross-platform frameworks have finally caught up with the native applications in terms of UX possibilities. No significant difference could be found between the different developed applications in

[7]https://medium.com/@firt/progressive-web-apps-on-ios-are-here-d00430dee3a7

the quantitative study. The React Native application performed slightly better than the other applications in the quantitative study, but this might just as well been a coincidence since no significant differences were established. The PWA performed better than the other applications in the qualitative study. In terms of access to native features the Android application is the best, but React Native only lacks access to one feature, which is rarely used in today's modern applications. The PWA comes up short in this comparison and lacks access to a few of the features, however it still has access to the most common ones.

If a party wants to create an application where not all native features are needed, a PWA might be the perfect choice. It is easy and fast to deploy, and its codeshare percentage is the highest of the compared frameworks. Furthermore, if the developers already maintain a website, the code will most likely be completely re-usable and no new knowledge will be needed, resulting in a fast learning-curve. However, if the party needs access to more native features a React Native application could be recommended. It has a similar learning curve to the PWA, provided the company uses ReactJS as their framework. Presuming the code is well separated, the policy-code will be re-usable between all platforms, whereas the mechanism code will be different between the web and the mobile applications since React Native does not run on websites. If a native extension is needed, some alteration of the code might be needed which results in a lower code-share percentage between iOS and Android. The gains of a shared codebase are decreased development cost and maintenance time which result in a drastically lower total cost of ownership. If a more delicate application is needed which needs access to all of the native features a native application might however be the correct choice. This is due to cross-platform applications not having complete access to all of the mentioned features. The downside with this approach is the need of two completely separate codebases. Finally, considering the huge gains of only maintaining one codebase and most companies not needing all of the features a cross-platform application is strongly recommended.

ACKNOWLEDGEMENTS
I would like to express my great appreciation to the company Challengermode where I have conducted this study and especially to my supervisor Fredrik Lilkaer who has given me much support and feedback during this project. Furthermore, I would like to give my special thanks to my supervisor at KTH Pedro Sanches who has given me great feedback whenever I have asked for it. Lastly, I would like to send an extra thanks to all of the participants in this study.

REFERENCES

1. P. Andrade, A. Albuquerque, O. Frota, R. Silveira, and F. da Silva. 2015. Cross platform app: a comparative study. *CoRR* abs/1503.03511 (2015), 33–40.
2. E. Angulo and X. Ferre. 2014. A Case Study on Cross-Platform Development Frameworks for Mobile Applications and UX. In *Proceedings of the XV International Conference on Human Computer Interaction (Interacción '14)*. ACM, New York, NY, USA, Article 27, 8 pages. DOI: `http://dx.doi.org/10.1145/2662253.2662280`
3. O. Axelsson and F. Carlström. 2016. *Evaluation Targeting React Native in Comparison to Native Mobile Development*. Master's thesis. Lund University, Faculty of Engineering.
4. A. Biørn-Hansen and G. Ghinea. 2018. Bridging the Gap: Investigating Device-Feature Exposure in Cross-Platform Development. *Proceedings of the 51st Hawaii International Conference on System Sciences* (01 2018).
5. A. Biørn-Hansen, T. Majchrzak, and T. Grønli. 2017. Progressive Web Apps: The Possible Web-native Unifier for Mobile Development. *Proceedings of the 13th International Conference on Web Information Systems and Technologies* (2017), 344–351.
6. G. Blom, J. Enger, G Englund, J. Grandell, and L. Holst. 2004. *Sannolikhetsteori och statistikteori med tillämpningar* (5 ed.). 9, Vol. 5. Studentlitteratur AB.
7. R. Fransson and A. Driaguine. 2017. Comparing Progressive Web Applications with Native Android Applications : An evaluation of performance when it comes to response time. (2017).
8. N. Hansson and T. Vidhall. 2016. *Effects on performance and usability for cross-platform application development using React Native*. Master's thesis. Linköping University, Human-Centered systems.
9. T. Lawler Karvonen. 2017. Native versus non native : A comparison of React Native and Angular NativeScript to native mobile applications Parallelism in Node.js applications. (2017).
10. M. Latif, Y. Lakhrissi, E. H. Nfaoui, and N. Es-Sbai. 2016. Cross platform approach for mobile application development: A survey. In *2016 International Conference on Information Technology for Organizations Development (IT4OD)*. 1–5.
11. B. Laugwitz, M. Schrepp, and T. Held. 2006. Konstruktion eines Fragebogens zur Messung der User Experience von Softwareprodukten. In *Mensch und Computer 2006: Mensch und Computer im Strukturwandel*, Andreas M. Heinecke and Hansjürgen Paul (Eds.). Oldenbourg Verlag, München, 125–134.
12. B. Laugwitz, M. Schrepp, and T. Held. 2008. Construction and Evaluation of a User Experience Questionnaire. In *HCI and Usability for Education and Work*, Andreas Holzinger (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 63–76.
13. T. Majchrzak, A. Biørn-Hansen, and T. Grønli. 2017. Comprehensive Analysis of Innovative Cross-Platform App Development Frameworks. *Proceedings of the 50th Hawaii International Conference on System Sciences* (1 2017).

14. T. Majchrzak, A. Biørn-Hansen, and T. Grønli. 2018. Progressive Web Apps: the Definite Approach to Cross-Platform Development? *Proceedings of the 51st Hawaii International Conference on System Sciences* (1 2018).
15. M. Palmieri, I. Singh, and A. Cicchetti. 2012. Comparison of cross-platform mobile development tools. In *2012 16th International Conference on Intelligence in Next Generation Networks*. 179–186. DOI: http://dx.doi.org/10.1109/ICIN.2012.6376023
16. Jerome H. Saltzer and M. Frans Kaashoek. 2009. *Principles of Computer System Design: An Introduction*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
17. D. Vesely. 2017. *Analysis and experiments with NativeScript and React Native framework*. Master's thesis. Masaryk University, Faculty of Informatics, Brno.
18. T. Vilček and T. Jakopec. 2017. Comparative analysis of tools for development of native and hybrid mobile applications. In *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. 1516–1521.
19. N. Wang, X. Chen, G. Song, Q. Lan, and H. R. Parsaei. 2017. Design of a New Mobile-Optimized Remote Laboratory Application Architecture for M-Learning. *IEEE Transactions on Industrial Electronics* 64, 3 (3 2017), 2382–2391.
20. Spyros Xanthopoulos and Stelios Xinogalos. 2013. A Comparative Analysis of Cross-platform Development Approaches for Mobile Applications. In *Proceedings of the 6th Balkan Conference in Informatics (BCI '13)*. ACM, 213–220.

APPENDIX

THIRD-PARTY SOURCES

Third-party sources that access native API with React Native

- Bluetooth
- USB
- NFC
- Battery Status
- Device Memory
- Advanced Camera Controls
- Recording Media
- Real-time Communication
- Storage Quotas
- Task Scheduling
- Speech Recognition
- Background Sync
- Payments
- Credentials
- Geofencing
- Device Position
- Device Motion
- Proximity Sensors
- Virtual & Augmented Reality
- Fullscreen
- Screen Orientation & Lock
- Wake Lock

FEATURES IN POPULAR APPLICATIONS

Facebook

1 000 000 000+ downloads on Google Play

- Top-navigation, fixed, swipe between views
- Scroll
- Text-input
- Back-button
- Lightbox
- Calendar-picker
- Clock-picker
- Checkbox
- Listview
- Button
- Switch

Google Maps

1 000 000 000+ downloads on Google Play

- Side-navigation, on click to open, swipe or click to remove
- Scroll
- Text-input
- Back-button
- Lightbox
- Calendar-picker
- Listview
- Button
- Switch

Instagram

1 000 000 000+ downloads on Google Play

- Bottom-navigation, fixed, no swipe between views
- Scroll
- Text-input
- Back-button
- Lightbox
- Checkbox
- Listview
- Button

Slack

5 000 000+ downloads on Google Play

- Side-navigation, on click to open, swipe or click to remove
- Scroll
- Text-input
- Back-button
- Lightbox
- Listview
- Button
- Switch
- Select

Snapchat

500 000 000+ downloads on Google Play

- Bottom navigation, fixed, swipe between views
- Scroll
- Text-input
- Back-button
- Lightbox
- Checkbox
- Listview
- Button
- Swipe around an object

Tinder

100 000 000 downloads on Google Play

- Top-navigation, fixed, swipe between views
- Scroll
- Text-input
- Back-button
- Lightbox
- Swipe around an object
- Listview
- Button
- Switch
- Seekbar

Tink

100 000+ downloads on Google Play

- Bottom navigation, not fixed on scroll, no swipe between views
- Scroll
- Text-input
- Back-button
- Lightbox
- Listview
- Button
- Vertical seekbar

www.kth.se