UNIVERSITEIT VAN AMSTERDAM

Applicability of Progressive Web Apps in Mobile Development

Tjarco Kerssens
tjarco.kerssens@student.uva.nl

June 2019, 54 pages

Thesis Supervisor:	Dr. Adam Belloum A.S.Z.Belloum@uva.nl
Host Organization:	KPMG, Digital Enablement
Host Supervisor:	Lie Yen Cheung, Cheung.LeiYen@kpmg.nl

Faculty of Science
MSc Software Engineering
Master thesis
The Netherlands
Science Park 904
1098 XH Amsterdam

Abstract

Since there exists much fragmentation in mobile platforms, developers have searched for alternatives in cross-platform development. One new methodology called Progressive Web Apps (PWAs) aims to unify the web and native approaches by combining the best principles of both, hence bridging the gap. This research aims to analyze the applicability of PWAs as an alternative to traditional native and web app development, in order to provide developers with arguments for the consideration of a development approach and to fill the knowledge gap in academic literature concerning mobile development. By comparing the performance and energy consumption of a PWA to native and web app implementations on iOS and Android, we found that the PWA version performed similar in most cases, better in some and only worse in launch time on iOS. Hence, the methodology seems to become a viable alternative for mobile development in the context of performance and energy consumption

Keywords: Progressive Web Apps, Native Apps, Web Apps, Mobile Development, Performance, Energy Consumption

Preface

The basis for this research stemmed from my passion for mobile development and advancements in this technological age. Exciting new things are happening and the world is moving forward at a fast pace. New technologies like Progressive Web Apps are contenders to shape the future of mobile development, hence my motivation to add to the scientific knowledge of this subject. This motivation and passion came together with the opportunity provided by KPMG to execute this research at their premises, with their help and advice.

Acknowledgements

This research could not have been performed without the support of multiple people. First of all, my academic supervisor Adam Belloum, who has given me a lot of advice and help with the process. Also, Lie Yen Cheung has helped me a lot with advice and daily supervision at the office of KPMG. Not only for the thesis but also more generally with my growth to become a professional. Furthermore, all the colleagues at the Digital Enablement department at KPMG are to be mentioned for creating a comfortable work environment and advising and challenging me. Morover, Ivano Malavolta has given me great advice on the subject and has lent me additional devices to support my research. Last but not least, Ana Oprescu from the Software Engineering department at the UvA has given structure to the process and has also taught me a lot about performing research. Thank you all.

Contents

Chapter 1

Introduction

Due to high fragmentation in the mobile world, the costs of creating a mobile application with the natural, native approach, could become very high [1, 2]. With this classic approach, developers need to develop, test and maintain for all the different platforms and devices, which is a time-consuming and therefore costly process [3, 4]. Therefore, developers and service providers have searched for ways to simplify the process of deploying their applications to the end-user [5].

On the other hand, mobile usage has been rising [6–8], meaning that the market's potential is also rising. Because of this, many companies could see a potentially big rise in the number of users if it were to implement its service as a mobile application. This observation, combined with the large costs in developing an application for each separate platform, makes it increasingly important to create a mobile application that runs on multiple platforms, the so-called cross-platform development approach [8].

Several approaches for cross-platform development as an alternative to native development have been established over the years [9]. These approaches can be broadly divided into Web, Hybrid, interpreted and compiled approaches, which will be explained in more detail in the background [2].

However, in recent years, a new approach called Progressive Web Apps has been created in order to bridge the gap between the web and native applications [9, 10]. The potential to create one application that runs on all devices with a browser, while circumventing the need to install and to depend on an app marketplace like the App or Play Store and being always up to date [10] seems promising. However, the Progressive Web App could still be installed and work offline, use native capabilities and would feel like a native application [11]. This is how the technology bridges the gap between native and web, it aims to take the best of both worlds [10]. This research aims to establish whether these Progressive Web Apps are a viable alternative to native applications and web applications.

1.1 Research Questions

This research is focused on identifying the current state of the Progressive Web App methodology and establishing whether it can be a viable alternative to the traditional native and web approaches. While the technology seems promising in theory, the question remains how it practically compares to other approaches.

In this thesis, the following research questions will be answered.

RQ: *Are Progressive Web Apps a viable alternative to native and web app approaches in mobile development?*

In order to answer this question, multiple concepts have been identified as being related to the viability of a mobile development approach. The motivation for these principles will be explained in the background section. These related concepts resulted in the following sub-questions:

- **RQ 1.1:** *Do PWAs and traditional web apps differ in performance?*
- **RQ 1.2:** *Do PWAs and traditional native apps differ in performance?*
- **RQ 1.3:** *Do PWAs and traditional native apps differ in energy consumption?*

As far as we know, these questions have not yet been answered by scientific research. Answering these questions will further contribute to the state of the art in the field of mobile development, as outlined in the next section.

1.2 Contributions

Since not much research has been done into the subject of Progressive Web Apps [9], this research might help developers make a decision about the adaption of this methodology in their services. The existing work suggests that there are promising advantages of adopting the technology, for example a decrease in installation size and fewer development hours. However, the current state of the art is mainly focused on comparing the PWA with other cross-platform alternatives, rather than the true native approach. Therefore, this research contributes to the knowledge gap in determining a development approach for mobile development.

Also, no other research concerning the energy consumption differences between Native and PWA implementations is known. Establishing this knowledge will further contribute to the consideration of mobile development approaches. The importance of this energy consumption will be further explained in the background section.

Furthermore, to our knowledge, no research has yet been performed concerning PWA for iOS, while iOS support is important for the adaption of a cross-platform development approach [2, 9]. Therefore, this research includes this platform in order to be able to answer the main research question more completely.

1.3 Outline

This thesis continues with a background in chapter 2, where the central concepts are explained and motives for the research questions are presented. It also reports an analysis of the problem and the status quo of the adaption of Progressive Web Apps. In chapter 3, Method, an overview of the measurements, tools, devices, setup and analysis is given. After that, a summary of methods and results of relevant research is presented in chapter 4, Related work. In chapter 5, the results of our experiments are presented. The discussion and conclusion in chapter 6 and chapter 7 conclude this thesis with answers to the research questions and add suggestions for future work.

Chapter 2

Background

In this chapter, the terminology and concepts, such as Native, Cross-platform and Progressive Web Applications related to this research are explained in the first sections. After that, the current state of adaption of the Progressive Web App approach is explored in section 2.4. Furthermore, an exploration and argumentation for the research direction is provided in the problem analysis in section 2.5. Finally, limitations in the research are identified in section 2.6.

2.1 Native Applications

Naturally, mobile development would be done using a native approach. The vendor of the platform provides a native developer kit that allows developers to create an application specifically tailored for that platform [9]. This means that this application will only run on the platform it was specifically created for, with no opportunities for code reuse in other platforms.

Furthermore, the different platforms require specific in-depth knowledge of the tools involved in order to develop a Native application for said platform. Developers also need to take the certain costs and restrictions into account when taking the native development approach, such as needing an Apple developer license and approval to distribute the application via the App Store [12]. This approval requires that the application meets the guidelines provided by Apple.

Native applications tend to be fast and reliable [12] since the platform it runs on is optimized for the Native approach and the vendor of the platform provides support for the development. Also, each vendor has its own guidelines for user interface design, meaning that there are differences in how the application is supposed to look and feel. With a Native approach, it is easier to follow these guidelines for each specific platform.

Furthermore, developers of Native applications can leverage the deployment services provided by the platform. Applications can be distributed via the application stores that the platforms provide and the selling options can be exploited to monetize the app.

Each platform has its own toolset and usually has only a few programming languages available for writing the application. In the case of iOS, the Swift programming language is the language Apple propagates as the default language for app development [13]. Furthermore, the platform that is used for developing and publishing the application is Xcode [14]. For Android, the language was traditionally Java, but since recently, Google promotes Kotlin as the main programming language for Android applications [15]. The Integrated Development Environment that is recommended for Android development is Android Studio [16]

2.2 Cross-Platform Approaches

Multiple other cross-platform approaches exist as alternatives for mobile development besides Progressive Web Apps. This section will shortly give an overview of these approaches. This research does not focus on these approaches, but they are interesting to take into account in the discussion.

Hybrid Approach
With the hybrid approach, the application is developed using web languages and ran inside a web view, rendering it as a native application. Native functionalities are exposed to the application through an abstraction layer [17]. Examples of Hybrid Approach frameworks are Cordova, PhoneGap, and Ionic, but these have been criticized for delivering sub-par performance [2].

Interpreted Approach
With the interpretation approach, a dedicated engine translates the source code of an application in real-time to an executable program [17]. Frameworks that use this approach include React Native and NativeScript [2].

Cross-Compiled Approach
The Cross-Compiled approach consists of compiling a single codebase to multiple native binaries for different platforms [17]. Rather than being dependent on an interpreter or web view, native binaries can be distributed. One example of a Cross-Compiled approach Framework is Xamarin [2].

2.3 Progressive Web Apps

The term Progressive Web App was firstly coined by Russell in 2015 [10]. The characteristics he identified then, are still advocated as being relevant for PWAs today:

- **Responsive**: To fit any form factor, i.e. any screen.

- **Connectivity independent**: Progressively-enhanced with Service Workers to let them work offline.
- **App-like-interactions**: Adopt a Shell + Content application model to create native app like navigation and interaction.
- **Fresh**: Always up-to-date via the Service Worker update process
- **Safe**: Served via TLS (a Service Worker requirement) to prevent snooping.
- **Discoverable**: PWAs are identifiable as “applications” thanks to W3C Manifests and Service Worker registration scope allowing search engines to find them.
- **Re-engageable**: A PWA can access the re-engagement UIs of the OS, for example, push notifications.
- **Installable**: Save the app on the home screen through browser-provided prompts, allowing users to “keep” apps they find most useful without the hassle of an app store.
- **Linkable**: Meaning that Progressive Web Apps do not require installation and are easy to share.

The term *Progressive* in Progressive Web Apps relates to taking advantage of enhancements that are available in the environment, rather than having fixed requirements [18]. In other words: continuously being able to add to the application when the browser allows it. Progressive also relates to not making an immediate choice about whether to install the app, it can progressively become an application rather than being just a website. [10]

Furthermore, Russell defined in a later article that Progressive Web Apps should feel and act more like a native app, rather than a web app which you can save to your home screen [11]. He also defines more characteristics that a Progressive Web App should have:

- Work without default browser environment: the application should load without a URL bar.
- Device independent: the application should work across browsers and devices.
- Near instant loading and fluid animations: this would make the application feel more like a native application.

Web App Manifest
The Web App Manifest is a JSON file that is used to configure the Progressive Web App [2]. It defines application metadata like icons, application name, and the base URL [19]. W3C further defines the manifest as being able to restrict the application to a single URL for security reasons. Furthermore, events are defined for several installation events [20].

Service Worker
The service worker allows the application to actually become a Progressive Web App, by providing features to enable the background operation separate from the rest of the website [2]. They provide the technical foundation to implement the app-like features such as an offline experience, background sync and push notifications [21].

Service workers are not able to directly access the DOM, but will rather act as an API for your application. It works asynchronous and event-driven in order to avoid blocking the application [2].

Application Shell
The Application shell is the first interface that a Progressive Web App renders. It allows for a user experience where the launching feels more native by loading a cached shell that allows for immediate interaction [2].

Google Web Fundamentals group describes the following benefits of using an application shell, which enable some of the Progressive Web App characteristics described above [22]:

- Reliable performance that is consistently fast.
- Native-like interactions such as offline support.
- Minimal data usage by caching the seldom changing shell.

2.4 Status Quo of PWA adaption

Scientific coverage of the concepts of PWAs is still low [9]. However, several companies seem to adopt these concepts in their products. During the Google I/O 2017, a developer conference hosted by Google, several initiatives and statistics related to Progressive Web Apps were presented [23]. In the talk, several companies are presented that have reported increased usage of their services and reduces application sizes after converting their applications to a Progressive Web App. These are companies such as Twitter, Financial Times, Lyft, AliExpress, Forbes, Flipkart, Expedia, Flipkart, Tinder, and Housing.com.

Furthermore, in the Google I/O conference of 2018, more adaption of the PWA methodology has been presented. Companies like Starbucks, Spotify, Editora Globo, and 1-800 Flowers have also reported successes after implementing a PWA [24]. Google also implements PWA functionalities in many of its own applications, for example, Search, Bulletin, Maps, and Gmail [24].

In chapter 4 we further explore the adaption of PWAs by several companies.

2.5 Problem Analysis

This section provides a motivation for the research questions established in the introduction. It explores two important characteristics for the adaption of Progressive Web Apps as a viable approach in mobile development, namely performance and energy consumption. The next sections will give motivation for these characteristics. There are still other characteristics relevant for a complete comparison, which are, due to time constraints, left as future work suggestions in chapter 7.

2.5.1 Performance

It has been stated that performance is important for the user experience and that user experience is critical for having a successful mobile application [25]. Consequently, an important part of this user experience and thus the success of the mobile application is the response and launch time [6, 26]. Even more, it has been established that the quality of the mobile experience can have an influence on the customer's opinion of the brand [1].

For websites, it has been shown that the threshold for acceptable load time is between 2-3 seconds. Longer load times lead to users leaving the web page [27]. Therefore, it will be interesting to find out whether PWAs can have a positive influence on the load times and increase the engagement with the users of the service.

Furthermore, it has been shown that a smaller app size can have a positive influence on the user adaption of the application [9]. It has been shown that Progressive Web Apps have a much smaller application size than native apps or other approaches like hybrid and interpreted applications on Android devices [2]. It is unclear if this is also the case for the iOS platform.

Since performance is closely related to user experience and user experience is related to the success of the mobile application, it is interesting to research how the performance of a Progressive Web App relates to native applications and traditional web applications.

2.5.2 Energy Consumption

While there are many technical improvements in smartphones, trends indicate that the ability of batteries will not have significant improvements anytime soon [28]. It has been established that energy is one of the most limited resources available on a smartphones [19], while research suggests that users will reject mobile applications that have a large impact on the battery life [29].

Even more, experiments have shown that advances in network connectivity, such as the transition from 3G to 4G LTE, can also have a negative impact on battery life [30]. Thus, since progress in battery life is unlikely, it is important that mobile applications make efficient use of the available power [3, 28, 30].

It has been shown that network connectivity can take up to 70% of power usage on mobile devices when the screen is off [31, 32], or up to 40% when the screen is on [33]. This indicates that saving on network requests can have a positive impact on power usage. Indeed, Dutta *et al.* have shown that caching approaches to reduce network requests can decrease battery usage [30]. Since caching resources is in the nature of a PWA, the energy consumption might be similar to a native implementation.

Since energy consumption is clearly an important characteristic in the field of mobile development, the viability of a Progressive Web App is probably related to its energy efficiency. It has already been established by Malavolta et al. that service workers do not have a negative impact on battery life [19], hence, there are no energy consumption problems when compared to Web Apps.

2.6 Limitations

As with all research, this research does not come without its limitations. First of all, the measurement of energy consumption on iOS is not as straightforward as it is for Android. For the Android OS, measurement tools have been developed and validated with scientific research. For iOS, measurements are to be done with Apple's own Instruments tool, which has no scientific reporting on its accuracy.

Furthermore, the research is conducted with a single mobile device for iOS and with a single application. This is to have as little differences between the implementations as possible, which would not be the case when multiple existing applications would have been used rather than an application specifically developed for this research. For Android, we use three devices, which is still not a lot.

Finally, this research does not take all the factors involved in mobile development into account for answering the question of whether Progressive Web Apps are a viable alternative to traditional counterparts. Effects of for example security implications, development effort, user adaption, and long-time maintenance effort could also be factors for choosing an approach in mobile development. These characteristics are described in the future work section in chapter 7.

Chapter 3

Method

In this chapter, the method of the research is described, as well as all the relevant variables and tools.

Most existing research in Progressive Web Apps is focused solely on Android, but it has been argued that for the adoption of PWAs as a true cross-platform approach, the technology should also work on iOS [9, 19]. Therefore, this research will be focused on both iOS and Android.

Comparisons will not be made between the different platforms since this is not possible due to the many differences in devices and platforms. Since the hardware and the platform of the devices are different, no useful data can be obtained for comparing mobile development approaches. Also, it is not relevant to the research question to compare between devices, since that would effectively be a study to compare platforms, rather than a comparison between the different methodologies of mobile development.

3.1 Performance

Most of the research concerning performance in mobile applications is focused on characteristics such as launch time, size of the installed application, time from app-icon to toolbar render and memory usage [2, 6, 25, 34]. Other research has taken scalability into account, studying the influence of the increase in users on performance [35].

Several methods have been proposed to make these kinds of measurements. Corral et al. measured within the application itself, by printing time-stamps before and after an operation [25]. Biørn-Hansen et al. used the Android Debug Bridge and a digital stopwatch to measure activity launch time and app icon to toolbar render respectively. [2].

Other research into the performance of web applications made use of the web

Platform	Measurement	App type	Method
iOS	First Paint	PWA	Safari Web Inspector
		Native	Run Script
		Web	Safari Web Inspector
	Install size	PWA	Safari Web Inspector
		Native	iOS reported
		Web	Not applicable
Android	First Paint	PWA	Debug Bridge
		Native	Android reported
		Web	Debug Bridge
	Install size	PWA	Android Reported
		Native	Android Reported
		Web	Not applicable

Table 3.1: Measurements for performance

development tools from a browser, i.e. Chrome, Firefox or Safari in order to test the performance. Furthermore, Lighthouse can be used to analyze a Progressive Web App [34].

It is hypothesized that caching and reloading, which can be enabled by service workers, will ensure a reliable and instant user experience [19]. Therefore, it is interesting to see whether the launch time of a cached progressive web app is lower than with a traditional web application.

3.1.1 Measurements

The properties that will be measured on each device, as well as the method of measurement, are presented in table 3.1.

Since recently there exists a Performance Timeline API, which currently has the working draft status at W3C. With this API, it is possible to retrieve high-resolution performance metric data. Furthermore, it contains methods to retrieve timing information about the first paint metric [36]. This allows for more accurate measurement on Chrome for Android. However, Safari for iOS currently does not support this API. Therefore, the web inspector will be used to measure the first paint for the iOS version of the PWA and Web App [37].

First Paint

The First Paint is defined as the time it takes from starting the launch of the application until the first pixel is drawn on the screen. This metric gives insight into the time until the application can be perceived as useful since it can provide information when it can draw to the screen. The initial launch of the application has been shown to be important for user acceptance of the application [6] and is, therefore, a useful metric for determining the viability of a PWA as an alternative to other approaches.

Furthermore, this metric will give insight into the influence of the caching provided by the service worker on the loading time of the PWA when compared to the traditional Web App. It is expected that the PWA loads faster since it is already cached locally on the device. In order to further test this hypothesis, the same test will also be performed on a simulated, throttled 3G network. The expectation is that the difference in launch time between the Web App and the PWA will be even larger.

For all implementations, this metric will be measured 50 times. In the next paragraphs, the different approaches for performing the measurements are described.

iOS measurements
For the iOS native application to measure the first paint, two parts of the launching process need to be measured. First, we need to measure how much time it takes from the initiation of the launch to the point where the application code gets the control. This pre-main stage, referring to the calling of the first method, the main method, can be measured by setting a run-time variable: DYLD_PRINT_STATISTICS = 1 [38]. Secondly, we need to measure the time from loading the application to actually painting the first screen. Therefore, we immediately make a time-stamp of when the application is launched and compare this to the moment the view is actually visible, in the lifecycle method viewDidAppear of the first screen [39]. This will be referred to as the post-main time. Hence, the actual first paint for the native iOS application is defined as $\textit{pre-main} + \textit{post-main}$.

The measurements for the iOS native application can be automated, by using a running script that installs the application on the device, runs it and parses the output to a file. This procedure can then be called inside a loop for repetition, hence automating the measurement progress. The running script writes two values per launch, the pre-main and post-main timestamps. Therefore, these values need to be added when processing the data.

The measurements for the PWA and the Web App on the iOS device, are performed in Safari since Apple only supports PWAs in its own browser. Furthermore, since the Performance Timeline API described above is not yet supported, the measurements will be performed using the Safari Web Inspector. The network tab shows the time when the first layout rendering takes place. This is a manual process, performed by connecting the iPhone to a MacBook which is running the Safari Web Inspector for remote debugging.

Android Measurements
For Android, the time to paint is reported to the console when running the application [40]. This makes it possible to create a script that runs the application, parses the output and saves the time to paint to a file. This procedure can then be run in a loop to collect measurements.

The measurements for the PWA and the Web App on the Android device are done using Chrome on Android and the remote debugging functionality of Chrome for Mac OS. With this setup, we can use the Performance Timeline API to print the first paint metric to the console.

Installation size

Many companies have attributed successes of conversion to a Progressive Web App partly on installation size [9, 23, 24]. Since the research has mainly been focused on Android, it will be interesting to see how the size relates to native applications on iOS. Validation of the reported successes with installtion size on Android is also done in this research.

The installation size for the native applications is reported by the operating system. For the PWA on Android, the installation size of the PWA is also reported. For iOS, we measure the installation size by using the remote debugging tool in Safari and calculating the size of the resources stored locally.

Lighthouse

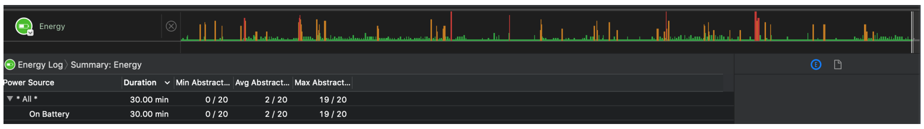
Lighthouse is a tool for analyzing the quality of web pages. Furthermore, it has support for analyzing the quality of a Progressive Web App. It is integrated into the Chrome Development tools and is able to measure several metrics [41]. In this research, it will be used to make sure that the PWA that is developed is of decent quality. The goal is to develop a PWA that has a score of at least 90 out of 100, since that is considered to be in the highest quality scale.

3.2 Energy Consumption

The method of measurements differs for Android and iOS. These differences are described below. For both platforms, twenty measurements are made for the Native App and the PWA. Each measurement will be done with a timeframe of 30 minutes.

Android measurements
Extensive research has been performed into measuring energy consumption on Android devices, where a software-based tool called Trepn Power Profiler has been reported to be sufficiently accurate [42]. The reported error margin for this tool is 99%. This method should be sufficient for measuring energy consumption on Android devices when enough data is collected. It can be used to measure the native Android application as well as the PWA running on the Android device.

iOS measurements
For iOS, Apple provides a tool called Instruments, which includes a energy con-



Energy Log 〉 Summary: Energy

Power Source	Duration	Min Abstract...	Avg Abstract...	Max Abstract...
▼ * All *	30.00 min	0 / 20	2 / 20	19 / 20
On Battery	30.00 min	0 / 20	2 / 20	19 / 20

Figure 3.1: A single measurement in the instruments tool by Apple

sumption module which can be used to analyze energy consumption [43]. This tool will be used to measure the energy consumption of the applications running on the iPhone. The tool reports a value between 0 and 20, where a higher value indicates more energy usage. However, no known validation of the accuracy of this method is provided. However, other research has been performed using this method of measuring energy consumption [44]. Also, Kwete has performed a study into energy measurement for iOS and lists only a few options, where the measurements using Instruments seems to be the best [45]. Other options are noting the battery percentage difference and noting the time it takes to completely drain a device [45]. The research lists an accuracy of 5%, but no source is given, hence it can not be said whether this is true. To give an indication of how this measurement looks like, figure 3.1 shows a single measurement in instruments.

3.3 Mobile devices

The devices used for this research, are the OnePlus X running Android and the iPhone 7 running iOS. The details of these devices can be found in table 3.2. The devices both run the latest stable version of their corresponding operating system, meaning they should be representative for the near future.

	iPhone 7	**OnePlus X**
OS	iOS 12.2	Android 9
Chipset	A10 fusion (16nm)	Qualcomm Snapdragon 801 (28nm)
CPU	Quad-core 2.34 GHz	Quad-core 2.3 GHz
Battery	1960 mAh	2525 mAh
Memory	2GB	3GB
Resolution	750x1334	1080x1920
Screen size	4.7 inch	5 inch

Table 3.2: Hardware details of the devices used in the research. The specifications are retrieved from GSMArena [46, 47].

Validation Devices
Two other Android devices are used to validate the results of this research, namely the Samsung Galaxy J7 Duo and the LG Nexus 5X. The details of these devices can be found in table 3.3.

	Samsung Galaxy J7	**LG Nexus 5X**
OS	Android 8	Android 8
Chipset	Exynos 7885 (14nm)	Qualcomm Snapdragon 808 (20nm)
CPU	2x2.2GHz 6x1.6GHz	2x1.8GHz 4x1.4GHz
Battery	3000 mAh	2700 mAh
Memory	4GB	2GB
Resolution	720x1280	1080x1920
Screen size	5.5 inch	5.2 inch

Table 3.3: Hardware details of the devices used to validate the results for Android in this research. The specifications are retrieved from GSMArena [48, 49]

3.4 Application Design

A total of four mobile applications will be developed for this experiment, 1) a native iOS application, 2) a native Android application, 3) a web app and 4) a PWA version of the web app. Since networking can have a large influence on battery life [30], it would be an interesting feature to have in the application. Also, accessing lower hardware like a GPU might differ between the applications, so graphical processing is also an interesting feature to include. Finally, accessing the location of the device is an interesting feature to have in the comparison since it has been reported to consume relatively much energy [50] and might be subjected to optimizations that are only available to native applications.

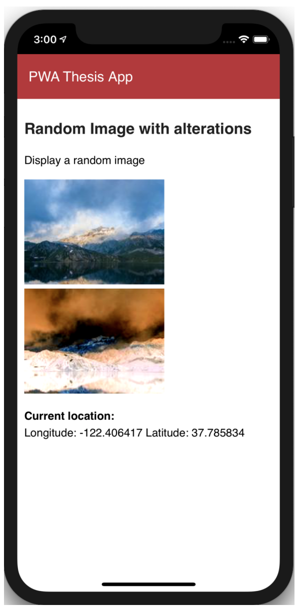
Therefore, Each application consists of a single screen that fetches a new, uncached image every five seconds, after which it inverts the colors, applies a contrast filter and draws it on the screen alongside the original image. Furthermore, the current location of the device is being followed continuously and is updated and shown on the screen. The code for the application can be found on GitHub [1]. Figure 3.2 shows screenshots of the implementations.

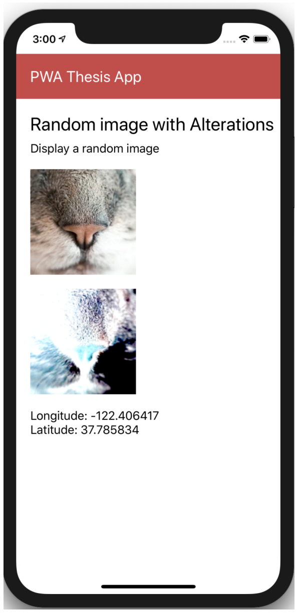
3.5 Experimental setup

For each measurement, the following steps are taken in order to ensure that the least amount of variance will occur between the measurements:

- Ensure that no other applications are running in the background.
- Have the device fully charged and have no energy saving mode enabled.
- Have the screen brightness set to the minimum value.
- Disconnect from the power source just before the measurement.

[1] https://github.com/TjarcoKerssens/thesis-pwa


3:00
PWA Thesis App
Random Image with alterations
Display a random image
Current location:
Longitude: -122.406417 Latitude: 37.785834
(a) PWA on iOS



(b) Native on iOS

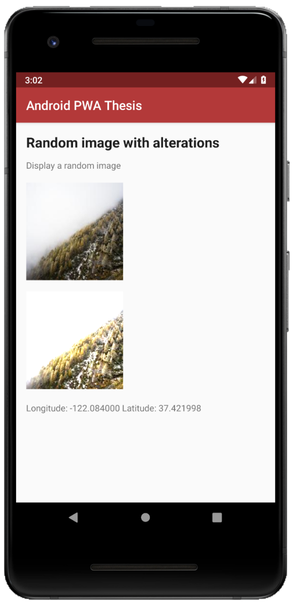

3:02
Android PWA Thesis
Random image with alterations
Display a random image
Longitude: -122.084000 Latitude: 37.421998
(c) Native on Android

Figure 3.2: Screenshots of the implementations

- Disable options that turn off the screen automatically.
- Disable Bluetooth and cellular connections. Only WiFi is needed.
- Connect to the same WiFi endpoint every time.

After this, the measurement is performed as described above.

3.6 Analysis

In order to establish meaningful insights from the results, multiple things are to be considered when analyzing the data. The first thing that needs to be shown is whether the data follows a normal distribution since this influences what test of comparison can be used. A Shapiro-Wilk test will be used to gain information about the normality of the distribution since this test has been shown to give a valid insight into normality [51]. Furthermore, QQ-plots are drawn for the data and visually analysed to support the assesment of normality.

If the collected data does seem to follow a normal distribution, the t-test can be used to analyze the differences in the means of the data. It has been argued by Ruxtan that an unequal variance t-test is better in many cases since it does not require the assumption that the variance of the data is equal [52]. Therefore, the Welch t-test will be used to analyze the data when it seems to be normally distributed. It has been shown by Skovlund and Fenstad that this test performs better than the student's t-test and Mann Whitney U test when the variance in the data is unequal [53]. The null hypothesis of this test is that the mean of the underlying data is similar. If the p-value is smaller than the significance level 0.01, we can reject this hypothesis and assume that there is a difference in the mean of the underlying data. Otherwise, the test adds support to the hypothesis that the means are similar.

Otherwise, if the data does not seem to follow a normal distribution, the Mann Whitney U test will be used to be able to analyze similarity in the data. This is a test that does not make any assumptions about the distribution of the underlying data. The null hypothesis with this test is that the two samples come from the same population, i.e. are similar. When the p-value is smaller than the significance level 0.01, we can quite confidently reject this hypothesis and accept that there is a difference in the underlying data. Otherwise, there is a strong indication that the data is actually similar, which adds support to the hypothesis that the underlying data is similar[54]. Additional support of the hypothesis is given by using the Komogorov-Smirnov test, which tests whether the two data sets are from the same continuous distribution [55].

It has been argued that failing to reject the null hypothesis does not inherently mean that we can accept the null hypothesis as being true [56]. However, it does give support for this to be the case. With a still low p-value, while larger than the significance level 0.01, this support is thin. In order to increase the strength of the support, a reverse test can be used in order to be able the reject the null hypothesis [56]. One way of doing that is making use of the two one-sided t-test (TOST) [57]. This test operates with two null-hypotheses, one that the true mean is lower or equal and one that the true mean is larger or equal than the true mean of the other data set. Rejecting both these hypotheses allows us to accept that the true means of the data sets are equal. However, this test can only be used for normally distributed data since it is using the t-test.

Chapter 4

Related Work

This chapter gives an overview of existing work in the field related to the performance and energy efficiency of mobile development approaches and summarizes the methods and results of these. This literature search forms the basis of the goals of this thesis, as well as giving an indication of answering the question of whether PWAs are similar in performance and energy consumption to other approaches. Furthermore, some cases of PWA implementation from the industry are presented.

4.1 Mobile Performance

Bjørn-Hansen et al. have compared Progressive Web Apps to Hybrid and Interpreted mobile applications on the Android operating system. In the research, performance is measured using the metrics size of the installation, activity launch time, and Time From App-icon Tap to Toolbar Render [2]. They found that PWAs have a much smaller installation size compared to both Hybrid and Interpreted apps, 157 times smaller than the Interpreted app made with React Native and 43 times smaller than the Hybrid implementation in the Ionic framework. The launch time of the PWA was similar to the Interpreted application, while it was faster than the Hybrid application. However, the time to toolbar render was shorter for the Interpreted application compared to the PWA, while the Hybrid application was the slowest. The research concludes there is a lot of potential in PWAs as a methodology for bridging the gap between native and web development. This indicates that the concept of a PWA might be considered as a viable alternative to native development [2]. The researches also encourage more research into the subject of Progressive Web Apps, since they concluded that the lack of academic involvement indicates a knowledge gap between the industry and academic community [2]. Finally, the research concludes that for adoption of Progressive Web Apps as a true alternative to other mobile development approaches, iOS needs to support the underlying techniques [2]. Since this support in iOS is currently available, although limited, this thesis aims to include this platform in the experiment.

Research by Majchrzak et al. from 2018 discusses that PWAs seem to be a promising alternative to traditional approaches, due to the properties that make it native-like while still remaining cross-platform and web-based. The study compares the features and performance of several approaches, namely Native, Hybrid, Interpreted, Cross-compiled and PWA. The researchers conclude it is too soon to establish whether it will actually become a definite approach to cross-platform development since the technology is not mature and not much research has been performed. However, the results indicate that the PWA version actually launches faster than all other approaches and that the time to render the toolbar is also faster. Furthermore, it is noted that iOS support is vital for the adoption of PWAs as an alternative to other cross-platform approaches [9].

Research by Johannsen concludes that the added complexity of transforming a Web Application to a PWA is low [58]. This indicates that the added costs of enhancing a Web Application with PWA capabilities are low and therefore a small hurdle in transforming an existing Web Application to a PWA.

Corral et al. have investigated the performance differences between a Native Android Application and a web-language based multi-platform framework called Cordova. The experiment is performed by logging the time difference between starting and finishing a function, hence assessing performance by analyzing execution time. The execution time is measured over multiple functionalities, like for example I/O operations, GPS requests, and network information requests. The research concludes that the performance of the web-based implementation in Cordova was worse and that the performance differences should be taken into account when making a decision for an approach for the sake of user experience [25].

In research by Gambhir and Raj, a Progressive Web App is compared to a native Android application on the metrics input latency and first meaningful paint. The research shows that the PWA was actually faster than the native Android implementation on these metrics. The caching process has a positive influence on the performance of the PWA since it reduces the number of network requests and therefore the load times [35]. It will be interesting to find whether these results indicate an outcome in favor of PWAs in this thesis. However, in a thesis by Yberg, it was found that the performance of a native Android application was better than the PWA implementation [34]. The comparison was, however, made by measuring the metrics for the PWA on a laptop rather than on a mobile device.

Pande et al. created a framework for inserting a Service worker into an existing web application, in order to improve the performance by adding a local cache. They found that their framework reduced the unnecessary data transfer between server and client, therefore improving the performance [4]. This is an interesting indication that the launch time of a Progressive Web App might be lower than the launch time of a traditional web app.

4.2 Mobile Energy Consumption

Malavolta et al. analyzed the impact of service workers on the energy consumption of multiple Progressive Web Apps, compared to their web app counterpart on the Android operating system. The research was performed using seven existing PWA implementations with either service workers turned on or off. It was found that service workers do not have a negative impact on the energy consumption of the application [19]. Even more, with service workers turned off, the medium of consumed energy was consistently lower than with service workers turned on. On the low-end testing device, the difference was 8.9 Joules in favor of the PWA, while on the high-end device the difference gets minimal with only 0.42 Joules [19].

In research by Ciman and Gaggi, a native application is compared to cross-platform approaches in terms of energy consumption. The native application was compared to multiple approaches, namely web, Hybrid, Interpreted and Cross-Compiled implementations, on both iOS and Android. For the measurement of energy consumption, a Monsoon PowerMonitor was used, which directly measures energy consumption at the hardware level. The results seem to indicate that adoption of a cross-platform approach will result in an increase in energy usage [3]. It will be interesting to see whether the Progressive Web App approach also increases energy consumption.

Since networking is an expensive operation in terms of energy consumption [31, 33], Dutta and VanderMeer hypothesized that device-level caching could reduce the energy consumption of mobile applications. The research was performed on the Android operating system. The authors found that local caching can reduce energy consumption related to network activity by up to 45% [30].

Metri et al. have published research where they measured the energy consumption of various background tasks. An interesting thing to note about this research is the fact that the measurements are also performed on an iPhone, rather than only an Android device as most research does [44]. In order to measure energy consumption, the Energy Profiler in the Instruments tool shipped with Xcode is used. This is the same method that will be used in this research. Deducting from the results, the measurements are about thirty minutes every time[44], which is also the running time in our method. However, they seem to only have done the measurement once, rather than multiple times.

Research by Li et al. has shown how optimizing HTTP requests can have a positive impact on energy consumption. Since HTTP requests have such an impact on energy consumption, this feature is included in our application. They conclude that energy is critical for mobile devices and that they are able to reduce energy consumption by 38% by optimizing the HTTP requests [59].

4.3 Industry implementations

Several companies have already leveraged the advantages of using a Progressive Web App and have reported successes in blog posts. A collection of success stories is maintained at pwastats.com [60]. Table 4.1 shows an overview of some of these stories.

Company	**Description**	**Result**
Twitter [61]	Twitter transformed the website ta a Progressive Web App, using The React Framework	Twitter has realized a significant improvement in engagement and reduction of data usage. There has been a 75% increase in Tweets sent
AliBaba [62]	The online trading platform AliBaba published a case study about the implementation of a PWA.	An increase of 76% in conversion rates is reported.
FlipKart [63]	A large e-commerce website in India, which converted its web app to a PWA, published a case study.	FlipKart reports an increase of three times as much time spent in the app, as well as three times lower data usage
AliExpress [64]	A global online retailer that has published a case study of converting their website to a PWA.	AliExpress reports an increase in the conversion rate of 104% and an increase of 74% of the time spent in the application.
Uber [65]	Uber implemented a PWA and published a case study.	A minimal installation size of 50kb and a load time of maximal 3 seconds on slow 2G networks is reported
Pinterest [63]	A social media website that published a case study where they compared their PWA implementation to their old mobile web app and the native applications for iOS and Android.	Pinterest reports an increase in time spent of 40% and an increase in user-generated revenue of 44% when compared to the old mobile web app. Furthermore, users spent 5% more time in the PWA implementation when compared to the native variant. Also, the installation size of the PWA is reported to be much smaller than the installation size of the native applications

Table 4.1: An overview of Progressive Web app implementations in the industry

Chapter 5

Results

This chapter presents the collected data and the statistics and graphs representing meaning in the underlying data. The data is collected as presented in the method chapter and the analysis performed as described in the analysis section.

The quality of the PWA is analyzed using the Lighthouse tool, as described in the method chapter. We determined that the score should be at least 90, in order to make sure that the implementation is of good quality. The final implementation of the PWA has the highest possible score of 100, as can be seen in figure 5.1.

5.1 Performance

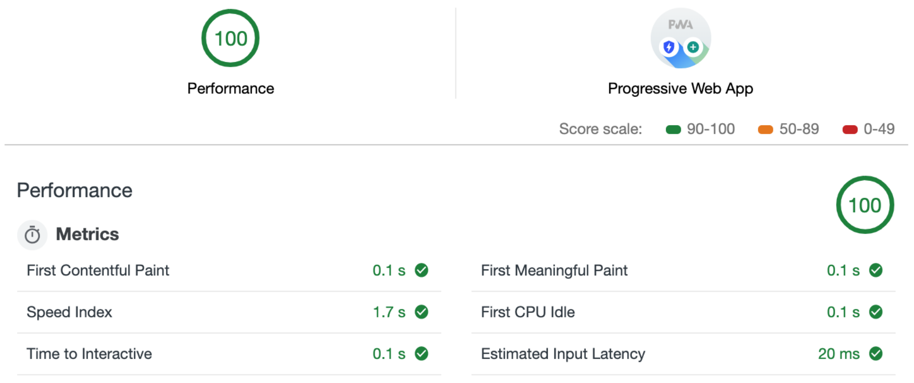
Performance is measured with the metrics first paint and installation size, as described in the method in chapter 3. In the next sections, the results for Android and iOS are presented. The statistics are computed in R and the code is available in the GitHub repository for this research [1].

5.1.1 Android

First Paint

In order to determine what test to use for the comparison of the data, we first need to establish whether the underlying data seems to follow a normal distribution. This is done by combining a Shapiro-Wilk test with an analysis of a Q-Q plot in R. Table 5.5 shows the results of these tests. Since all p-values of the test, except the value for the native app and the Web App on 3G, are lower than the significance level 0.01, we can reject the null hypothesis that the

[1]https://github.com/TjarcoKerssens/thesis-pwa


100
Performance
Progressive Web App
Score scale: ● 90-100 ● 50-89 ● 0-49
Performance
100
Metrics

First Contentful Paint	0.1 s ✓	First Meaningful Paint	0.1 s ✓
Speed Index	1.7 s ✓	First CPU Idle	0.1 s ✓
Time to Interactive	0.1 s ✓	Estimated Input Latency	20 ms ✓

Figure 5.1: The quality of the PWA is audited using Lighthouse and given a perfect score

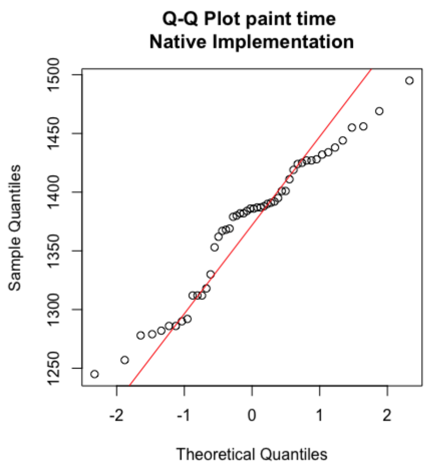
Statistic	**p-value**
Shapiro-Wilk Native p-value	0.044054
Shapiro-Wilk Web p-value	0.005125
Shapiro-Wilk Web 3G p-value	0.408082
Shapiro-Wilk PWA p-value	0.000346
Shapiro-Wilk PWA 3G p-value	0.000875
Mann-Whitney U Native - PWA	<2.2e-16
Mann-Whitney U Web - PWA	<2.2e-16
Mann-Whitney U 3G Web - PWA	<2.2e-16

Table 5.1: First Paint results Android

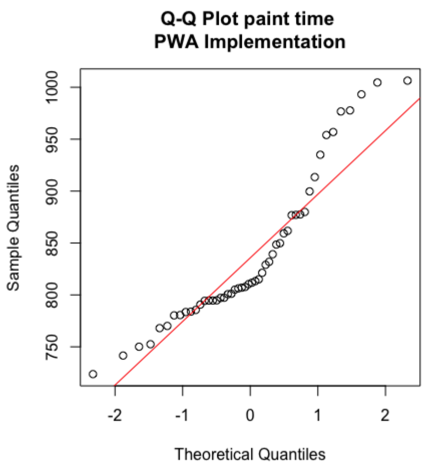
data is normally distributed. When looking at the Q-Q plots in figure 5.2, we see that the data sets tend to approach a normal distribution, but does deviate often. Based on these observations, we will use the Mann-Whitney U test to analyze the data.

The p-values for the Mann-Whitney U tests for comparing the PWA to the Native and Web App implementations are presented in table 5.1. The test operates under the null hypothesis H0 that the underlying data sets come from the same class, or the same data set. Since all the p-values are below the significance level of 0.01, we can reject H0 and conclude that there is a significant difference in the data. The box plots in figure 5.3 indicate that the PWA launches faster than the Native and Web App implementations. Due to the caching of the resources in the PWA by the service worker, the difference in first paint times between the PWA and Web App becomes even larger on a slower 3G network.

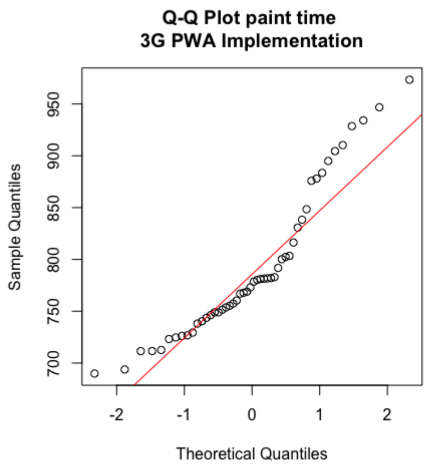
The first paint time might be influenced by the installation size since a smaller installation size requires fewer resources to be loaded into memory. The next section shows that the PWA is much smaller than the Native implementation,



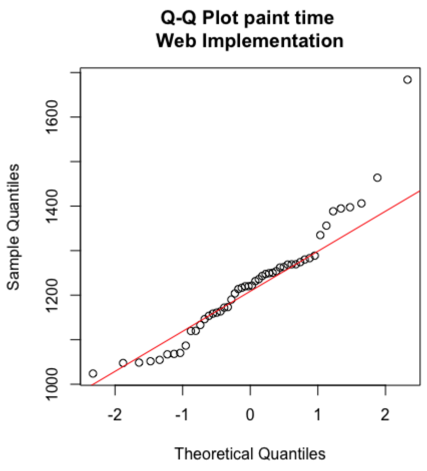
(a) Native



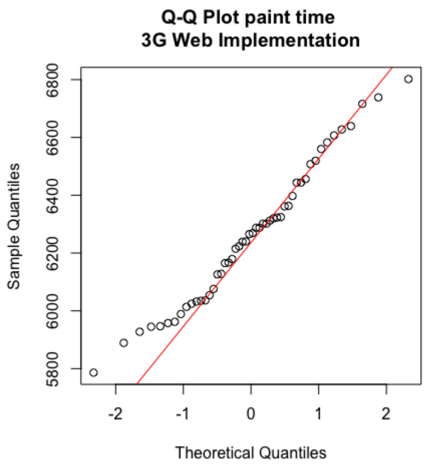
(b) PWA



(c) PWA 3G



(d) Web App



(e) Web App 3G

Figure 5.2: Q-Q plots to assess normality in paint time data for Native, PWA and Web App implementations on Android

Statistic	**p-value**
Shapiro-Wilk Native p-value	1.073e-07
Shapiro-Wilk PWA p-value	0.041503
Shapiro-Wilk Web p-value	0.152083
Mann-Whitney U Native - PWA	4.41e-15
Mann-Whitney U Web - PWA	<2.2e-16

Table 5.2: First Paint results on the Samsung Galaxy J device

Statistic	**p-value**
Shapiro-Wilk Native p-value	5.165e-12
Shapiro-Wilk PWA p-value	0.000346
Shapiro-Wilk Web p-value	0.005480
Mann-Whitney U Native - PWA	<2.2e-16
Mann-Whitney U Web - PWA	<2.2e-16

Table 5.3: First Paint results on the Nexus 5X device

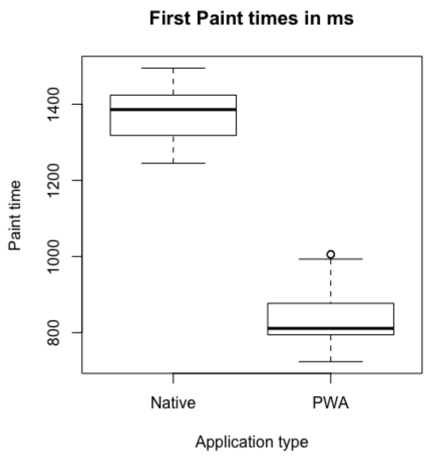
which might influence the observation that the PWA launches faster. We test this hypothesis by increasing the PWA size by adding a very large image and caching this image with the service worker. The total size of the PWA then becomes 8.1 MB, roughly the same as the native implementation. Next, the first paint time is measured twenty times and analyzed using the same method as above. We found that for this larger installation size, the Mann-Whitney U test confirms the hypothesis that the size influences the launch time. With a p-value of 0.146799, we can not reject the null hypothesis that the underlying launch times differ and we accept that they are similar. Figure 5.4 shows the box plot of the larger PWA and the Native implementation.

The first paint measurements were also performed on two other devices in order to validate the results. The results of the analysis of the measurements can be found in table 5.2 for the Samsung Galaxy J and in table 5.3 for the Nexus 5X.

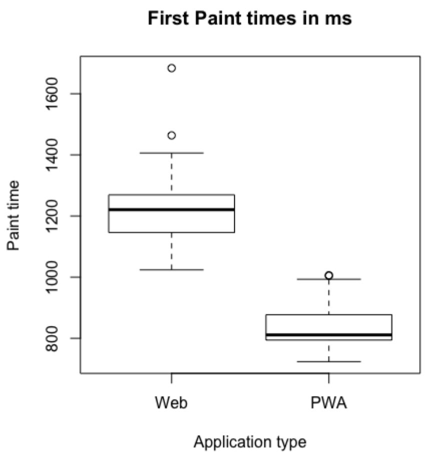
The validation experiments have the same results as the initial experiment, validating the result that the PWA launches faster on Android. The box plots in figure 5.5 show the first paint times for the PWA and Native implementation on the Samsung Galaxy J device. The box plots in figure 5.6 show the results for the Nexus 5X.

Installation size

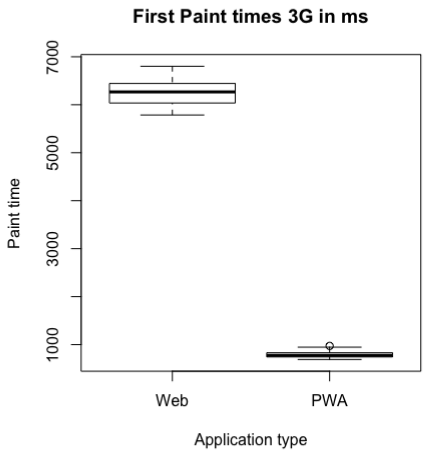
Table 5.4 shows the installation sizes for the PWA and Native implementations of the application. The Web App cannot be installed and is therefore not in the results. The PWA implementation is much smaller in installation size, as expected.



(a) Native and PWA



(b) Web App and PWA



(c) 3G Web App and PWA

Figure 5.3: Box plots of the first paint times on Android in milliseconds

Native	**PWA**
8.73 MB	0.30 MB

Table 5.4: Installation sizes for the PWA and Native app on Android

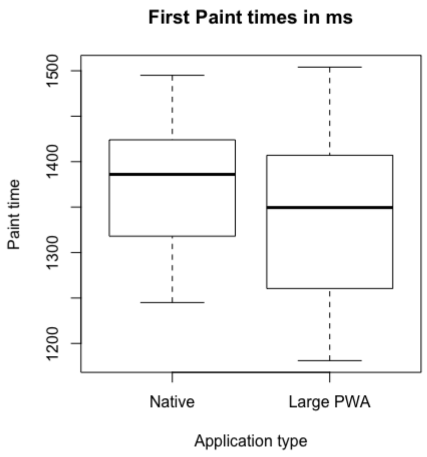
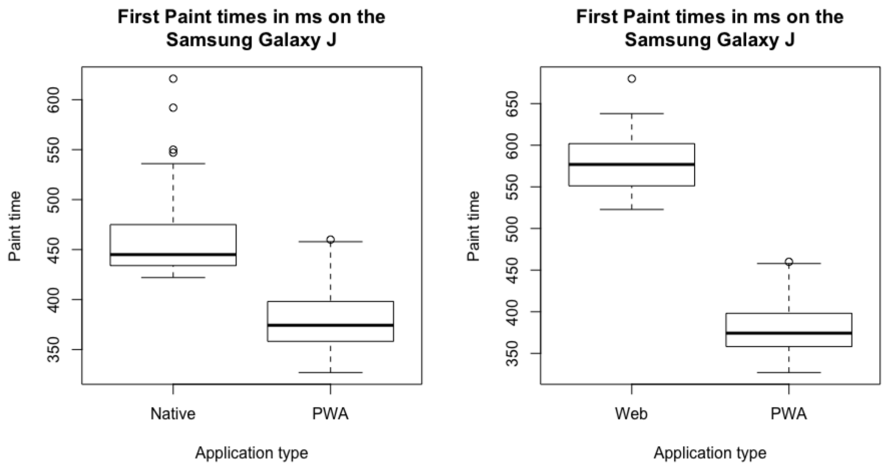


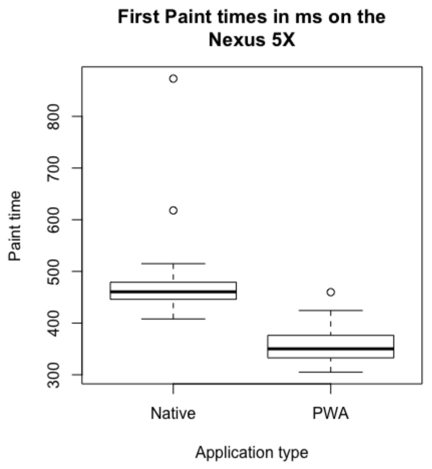
Figure 5.4: Box plot showing the first paint time on Android for the Native and larger PWA implementation. The PWA has been made bigger by adding large images.



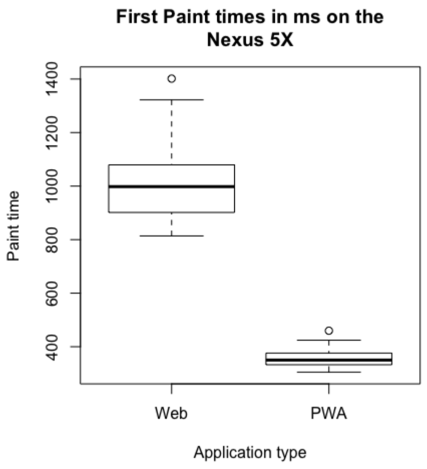
(a) Native and PWA

(b) Web App and PWA

Figure 5.5: Box plots showing the first paint time on the Samsung Galaxy J7.



(a) Native and PWA



(b) Web App and PWA

Figure 5.6: Box plots showing the first paint time on the Nexus 5X

Statistic	**p-value**
Shapiro-Wilk Native p-value	0.007179
Shapiro-Wilk Web p-value	0.000609
Shapiro-Wilk Web 3G p-value	0.000306
Shapiro-Wilk PWA p-value	0.007978
Shapiro-Wilk PWA 3G p-value	0.107025
Mann-Whitney U Native - PWA	<2.2e-16
Mann-Whitney U Web - PWA	<2.2e-16
Mann-Whitney U 3G Web - PWA	<2.2e-16

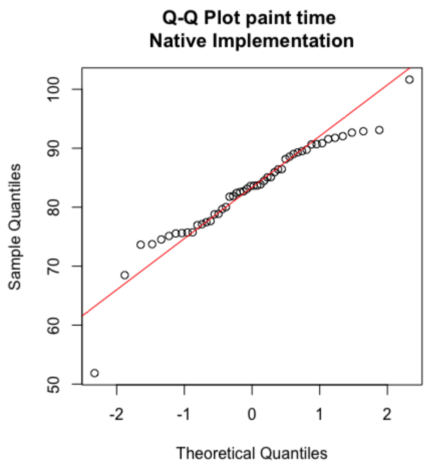
Table 5.5: First Paint results iOS

5.1.2 iOS

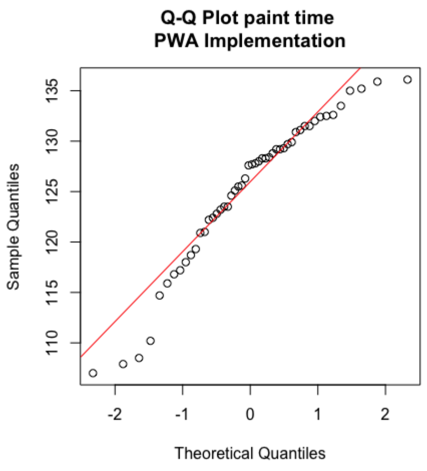
First Paint

First, the Shapiro-Wilk test is used to gain insight into the distribution of the data. The null hypothesis is that the underlying data is normally distributed, which can be rejected when the p-value is lower than the significance level 0.01. The results for the Shapiro-Wilk tests for the different data sets can be found in table 5.5. For all measurements, except for the PWA under 3G simulation, the p-value of the Shapiro-Wilk test is smaller than 0.01. Therefore, we can say that the data is likely not normally distributed. Figure 5.7 shows the Q-Q plots for the first paint data, which also shows that there can not be enough confidence in the normality of the underlying data.

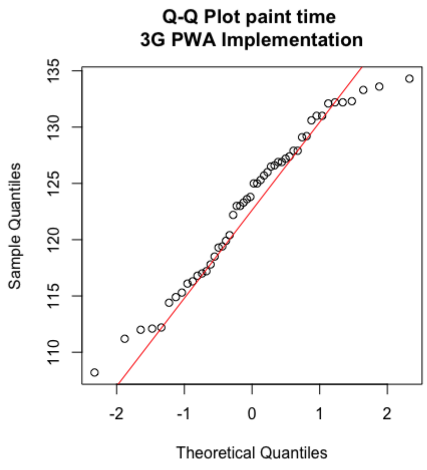
Since we can not assume that the underlying data is normally distributed, the



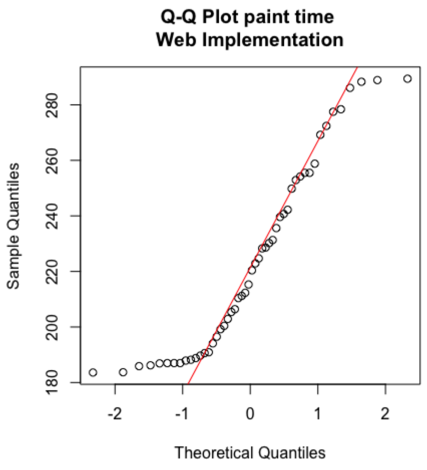
(a) Native



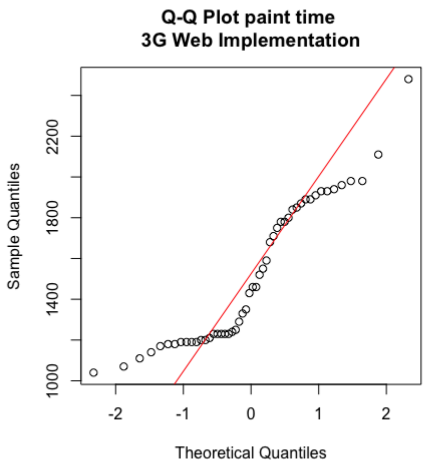
(b) PWA



(c) PWA 3G



(d) Web App



(e) Web App 3G

Figure 5.7: Q-Q plots to assess normality in paint time data for Native, PWA and Web App implementations

Native	PWA
2.10 MB	0.56 MB

Table 5.6: Installation sizes for the PWA and Native app on iOS

assessment of similarity is performed by using the Mann-Whitney U test. The resulting p-values of the comparisons are reported in table 5.5. Under the null hypothesis, the underlying data is collected from the same class of data, meaning that the launch time of the compared implementations is similar. Since the p-value is lower than the significance level 0.01, We reject the null hypothesis and accept that there is a difference in the first paint times between all implementations. More specifically, when analyzing the box-plots presented in figure 5.8, we find that the native implementation launches faster than the PWA implementation and that the PWA implementation launches faster than the Web App implementation. This result is even more apparent on slow networks, where the caching of the PWA gives a bigger advantage over the Web App.

Installation Size

On iOS, the installation sizes for the Native App and PWA also differ. However, the difference is smaller. The installation sizes for both the implementations are presented in table 5.6.

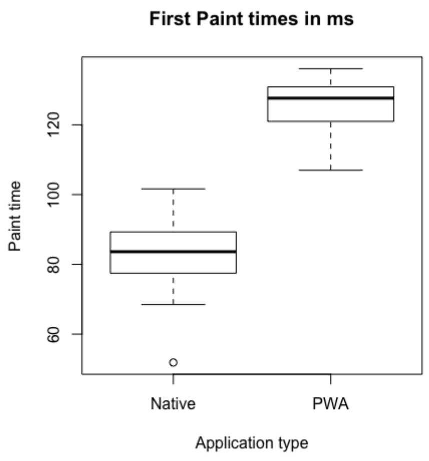
5.2 Energy Consumption

The next sections will present the results for the measurements in energy consumption for Android as well as iOS. The statistics are computed in R and the code is available in the GitHub repository for this research [2].

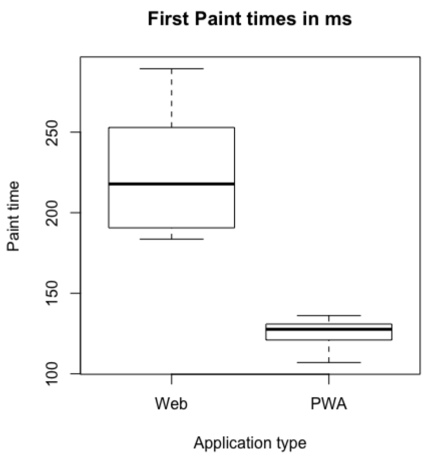
5.2.1 Android

First, the Shapiro-Wilk test is used to establish whether the data is normally distributed. The null hypothesis H0 is that the underlying data is normally distributed, hence with a p-value lower than the significance level 0.01, we will reject H0 and conclude that the underlying data is not normally distributed. The Shapiro-Wilk test resulted in a p-value of 0.258585 for the energy consumption data of the native implementation and a p-value of 0.290080 for the PWA implementation. This means we can not reject H0, however, the p-value is quite low and hence concluding that the data is normally distributed might be a risk to validity. Therefore, we compute the t-test statistic as well as the Mann-Whitney U statistic to see whether we obtain a similar result.

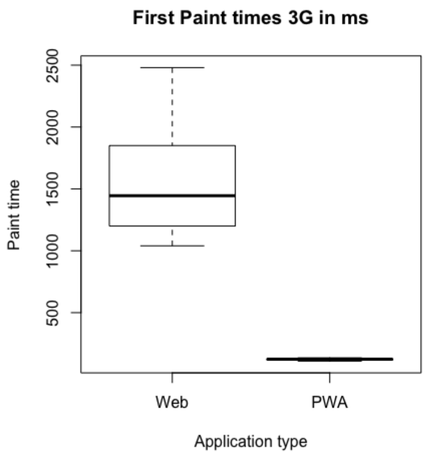
[2]https://github.com/TjarcoKerssens/thesis-pwa



(a) Native and PWA



(b) Web App and PWA



(c) 3G Web App and PWA

Figure 5.8: Box plots of the first paint times on iOS in milliseconds.

Furthermore, the normality of the data can be graphically represented with the drawing of a Q-Q plot. As shown in 5.10, the data seems to be normally distributed since most data points are near the expected line.

With the t-test, we have the null hypothesis H0 that the means of the underlying data sets are the same. Hence, with a p-value of lower than the significance level 0.01, we can reject H0 and conclude that the means do differ. The t-test resulted in a p-value of 0.014620, meaning that we can not reject H0 and that we have support that there is not a significant difference in the Energy Consumption of the PWA and Native implementations. However, the p-value is very low, making the support poor.

When using the Mann-Whitney U test, we have the null hypothesis H0 that the underlying data comes from the same data set, which can, in this case, be interpreted as the set of similar energy consumption values. The test resulted in the p-value of 0.003534, meaning that we can reject H0 and conclude that the energy consumption values are in a different class. Hence, we have a slightly different result with the t-test and Mann-Whitney U test. The p-value of the Mann-Whitney U test is however still quite close to the significance level 0.01.

In order to increase support for the hypothesis that energy consumption is similar, the Kolmogorov-Smirnov statistic is calculated [55]. The test is also performed using R and operates under the null hypothesis that both data sets are drawn from the same continuous distribution [66]. The test resulted in a p-value of 0.012299, which is larger than the significance level 0.01, meaning that we can not reject the null hypothesis and that we find support for the hypothesis that the energy consumption is similar. However, the p-value is, just as with the other tests, very close to the significance level 0.01and the support is therefore not very strong.

Parkhurst argues that failing to reject a null hypothesis does not give strong support to accept the null hypothesis and that further testing with a reverse test is needed. That is, a test where the null hypothesis is that the mean of the underlying data is unequal rather than equal [56]. One way of analyzing this hypothesis is by making use of the two one-sided t-tests (TOST), which tests both the hypotheses that the true mean is lower or equal and that the true mean is higher or equal when comparing two data sets [57]. Rejecting both these hypotheses leads to accepting that the true means of the data sets are equal.

The first one-sided t-test tests whether the true mean of the energy consumption of the Native implementation is larger than the true mean of the energy consumption of the PWA. The null hypothesis is that the true mean is not greater or equal, hence we can reject H0 and accept that the mean is greater or equal for a p-value smaller than the significance level 0.01. The test gives a p-value of 0.007310, which is smaller than the significance level 0.01, allowing us to reject H0 and accept that the energy consumption of the native implementation is larger.

The second one-sided t-test test whether the true mean of the energy consumption of the Native implementation is smaller than the true mean of the energy consumption of the PWA. The null hypothesis is that the true mean is not smaller or equal, hence we can reject H0 and accept that the true mean is smaller or equal for a p-value smaller than the significance level 0.01. We found a p-value of 0.992690, meaning we can not reject H0. Since the p-value is quite high, it gives strong support for the alternative hypothesis that energy consumption of the Native implementation is higher than the energy consumption of the PWA.

Since we can only reject one hypothesis of the two one-sided t-tests, we can not conclude that the true mean is equal. All tests combined give strong support for concluding that there is a significant difference in energy consumption between the Native and PWA implementation. The average energy consumption of the PWA implementation is 2.287084 kilojoule, which is slightly lower than the average energy consumption of the Native implementation, which is 2.328142 kilojoule.

Table 5.7 gives an overview of the results for the energy consumption on the OnePlus X running Android. The box plot in figure 5.9 also indicates that the average energy consumption of the PWA implementation is lower than the energy consumption of the Native implementation on Android.

Since the results for the energy consumption on Android is not as expected, additional experiments are needed. We hypothesize that the difference in energy consumption might be caused by an incorrect assessment of the measurement tool what consumption of energy actually belongs to the PWA. It might be possible that some consumption is done by the browser engine, which would not be measured with the application specific measurement. Therefore, we perform additional measurements where the whole system's energy consumption is measured.

The results of these extra tests can be found in table 5.8. For all statistical tests, the p-value is slightly bigger than the significance level 0.01. Therefore, can not reject the null hypothesis that the energy consumption is similar. While support is thin, we can not conclude that there is a difference in energy consumption, contrary to the app-level measurements. A box plot for the results of these tests is shown in figure 5.11.

The other Android devices did not report values for energy consumption after performing several measurements. It seems to be the case that the Trepn profiler does not support energy measurements on these devices.

5.2.2 iOS

The same method will be applied to the energy consumption data from the iOS platform. However, the data is quite different, since we could only obtain abstract energy consumption scores rather than direct values as with Android. The Shapiro-Wilk test on the iOS data results in a p-value of 0.001507 for the Native implementation and a p-value of 0.000205 for the PWA implementation,

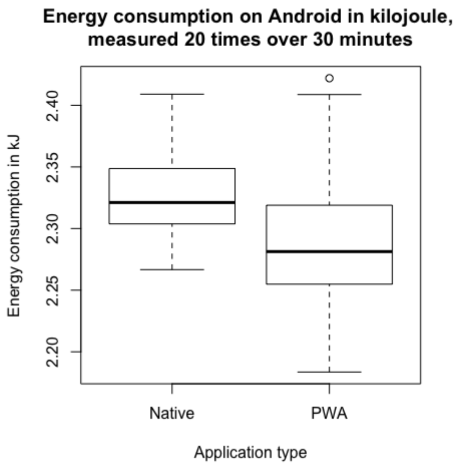


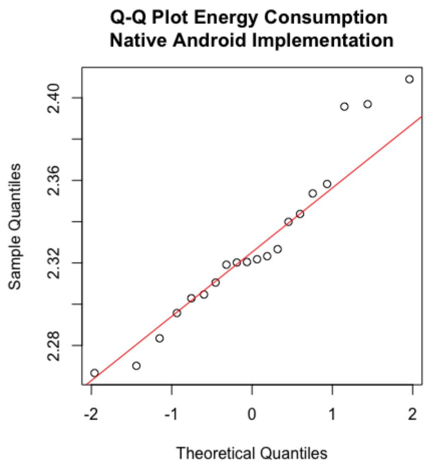
Figure 5.9: Boxplot showing energy consumption data for Native and PWA on Android

Property	**Value**
Shapiro-Wilk Native p-value	0.258585
Shapiro-Wilk PWA p-value	0.290080
T-test p-value	0.014620
Mann-Whitney U test p-value	0.003534
Kolmogorov-Smirnov p-value	0.012299
Mean energy consumption Native	2.328142 kJ
Mean energy consumption PWA	2.287084 kJ

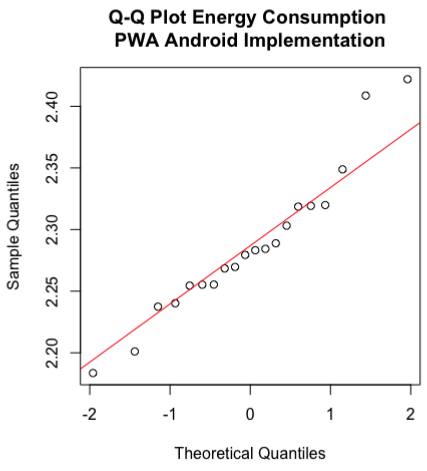
Table 5.7: Results for energy consumption on Android, measured on a OnePlus X

Property	**Value**
Shapiro-Wilk Native p-value	0.746166
Shapiro-Wilk PWA p-value	0.111150
T-test p-value	0.202099
Mann-Whitney U test p-value	0.217600
Kolmogorov-Smirnov p-value	0.167821
Mean energy consumption Native	2.354426 kJ
Mean energy consumption PWA	2.327175 kJ

Table 5.8: Whole system energy consumption results for Android, measured on a OnePlus X.



(a) Native



(b) PWA

Figure 5.10: Q-Q plots to assess normality for PWA and Native implementations for Android

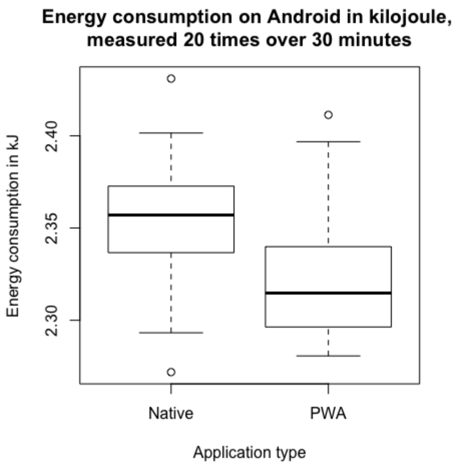


Figure 5.11: Box plot showing energy consumption data for Native and PWA on Android, while measuring the whole system's energy consumption.

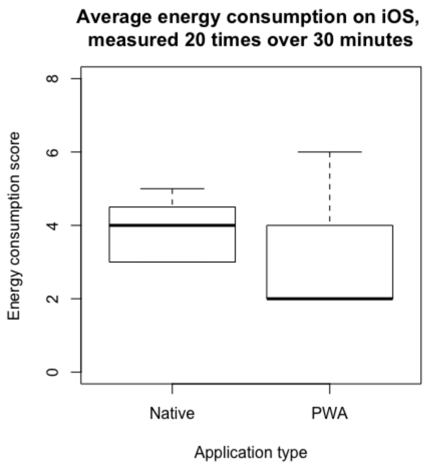


Figure 5.12: Boxplot showing energy consumption data for Native and PWA implementations on iOS

Property	**Value**
Shapiro-Wilk Native p-value	0.231147
Shapiro-Wilk PWA p-value	0.289544
Mann-Whitney U test p-value	0.003534
Kolmogorov-Smirnov p-value	0.004716

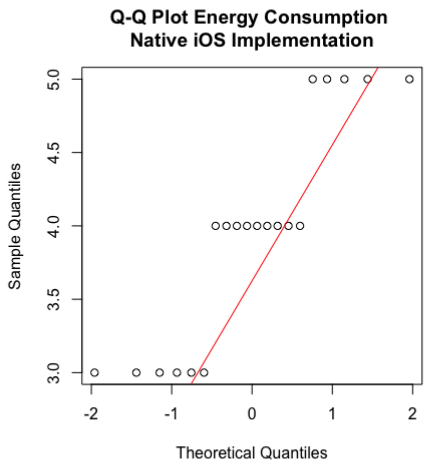
Table 5.9: Results for iOS

meaning we can reject the null hypothesis that the data is normally distributed. This can also be found by examining the Q-Q plots of the data, presented in figure 5.13. Hence, we can not use the t-test to assess similarity in the underlying data and the Mann-Whitney U test will be used.

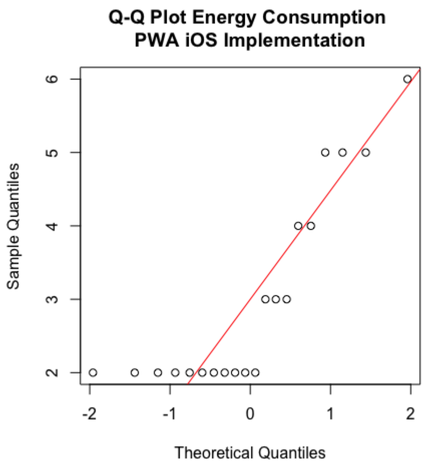
The Mann-Whitney U test gives a p-value of 0.006792, meaning we can reject the null hypothesis that the data for energy consumption comes from the same class. When looking at the box-plot of the data in figure 5.12, it seems to be the case that the PWA implementation has lower energy consumption than the Native implementation.

In order to further test the hypothesis that the energy consumption is similar, the Kolmogorov-Smirnov statistic is calculated, just as with the Android version. The null hypothesis is that the data sets are drawn from the same continuous distribution. The test gives a p-value of 0.004716, which is lower than the significance level 0.01, allowing us to reject the null hypothesis.

Table 5.9 shows the results of the tests for the Native and PWA implementations on iOS



(a) Native



(b) PWA

Figure 5.13: Q-Q plots to assess normality for PWA and Native implementations for iOS

Chapter 6

Discussion

This chapter explains the results described in chapter 5 and puts them in context of the state-of-the art in mobile development. It is divided into sections for iOS and Android

6.1 Android

6.1.1 Performance

First Paint

Since we could not establish the normality of the data with enough confidence, the data was analyzed with the Mann-Whitney U test. It is interesting to see that there actually is a difference in performance for the Native and PWA implementation and that the PWA implementation seems to launch slightly faster. These results are complementary to the work published by Bjørn-Hansen et al. [2], where the interpreted implementation of the same app was found to launch slightly slower than the PWA implementation. However, an interpreted implementation is different from a pure native implementation and it is interesting to see that the PWA also launches slightly faster than the purely native implementation, contrary to our hypothesis that the PWA would be slightly slower.

The first paint results also seem to be in line with the results from the study by Majchrzak et al. [9], where the PWA was found to launch faster than the Hybrid and Interpreted implementation. The study also found that the time to render the toolbar, the same launch metric used as in the research by Bjørn-Hansen et al. [2], was faster with the PWA implementation than the Hybrid implementation, and only slightly slower than the Interpreted version [9].

These results are also in line with the work by Gambhir and Raj [35], which also concludes that the PWA implementation is faster on the metric first meaningful paint. The only related work that seems to favor the Native implementation in terms of launch time, is the thesis done by Yberg [34]. The fact that this research was not done on mobile devices, makes it fundamentally different from this research and the related work discussed.

One reason why the PWA actually launch faster might be that the size of the application is a lot smaller, as shown in the next section. This would mean that it would take less time for the app to be loaded from the storage into memory and that this would actually be a bottleneck in the launch process. This size difference is also apparent from the related work described in chapter 4. We have tested this hypothesis, as described in the results section, and found that the first paint time was actually similar when the size of the PWA was increased to be the same as the native implementation. Therefore, it seems to be the case that installation size has an influence on the launch time.

The data also indicates that the PWA launches faster than the Web App version, which makes sense since the PWA is installed locally rather than fetched via the network. The differences become even more apparent when the network was throttled to a 3G connection, where the Web App launched a lot slower than the PWA. This means that on slow networks, a PWA will have a huge gain in launch time over traditional Web Apps.

It is interesting to see that the results combined with the related work seem to indicate that the PWA implementation will launch faster than the Native implementation of the same app. This will be an important aspect of establishing whether a PWA is a good approach to developing a mobile application.

Installation Size

The installation size of the PWA implementation is 29.1 times smaller than the installation size of the Native application on Android. This result is in line with all the related work and the reported successes from the companies also discussed in chapter 4.

6.1.2 Energy Consumption

The results seem to indicate that there is a slight difference in energy consumption between the PWA and Native implementations. The average energy consumption of the PWA version seems to be lower than the average energy consumption of the Native implementation.

Since research by Ciman and Caggi has shown that the energy consumption of other cross-platform approaches is worse than the consumption of the native implementation on Android [3], the result that the PWA actually consumes slightly less energy is promising. Since it has been shown that HTTP requests have a large impact on energy consumption while they are also made often in mobile apps [59], our prototyping app performs many HTTP requests. Web engines that run the PWA might be more optimized to perform these requests in a more efficient manner than the native libraries provided by the OS. We could not find any research supporting this hypothesis, hence we have tested this assumption by measuring energy consumption of the whole system during a run of the application, rather than the consumption of the isolated application. We found that with the measurement of the whole system, the consumption does not significantly differ, while the box plots indicate a slightly lower consumption for the PWA version.

In any case, the results are promising for the future of the PWA methodology. Worse energy consumption might be an important hurdle in the adaption of the technology as an alternative to traditional approaches [29] and these results seem to indicate that this will not be a problem for the PWA approach.

6.2 iOS

6.2.1 Performance

First Paint

On iOS, the Mann-Whitney U test and the plots seem to indicate that there is a significant difference in the first paint times between the Native and PWA implementations. The data shows that the Native implementation of the same app launches slightly faster than the PWA version. This is contrary to the first paint time on Android, where the PWA actually launches faster. It could be the case that Native Apps on iOS are more optimized than Native Apps on Android since the platform is less fragmented and therefore apps can be more fine-grained for the specific devices.

Another possible explanation is that Android is further ahead in terms of PWA support. iOS has only recently started to support PWAs, which could indicate that the support is not fully optimized. Furthermore, the size difference of the applications is much larger on Android than on iOS. We hypothesized for Android that the size might be related to the launch time since loading the application into memory would take less time for small app sizes. If this is true then it would make sense that a smaller difference in size would reduce the influence of the size on the time it takes to launch the application. This could also be a reason for the different results for Android and iOS.

The PWA implementation does launch a lot faster than the Web version, which can be explained by the fact that the PWA is launched from the local installation and the Web App version needs to be fetched over the network first. Throttling the network to a 3G connection makes this property more apparent since the differences in launch times become larger.

It is interesting to see that on iOS, the Native App still launches faster than the PWA implementation. Whether this is due to more efficient Native development or less efficient PWA support, is yet to be seen. The launch times of both implementations seem rather low, only about a fifth of a second. Whether a user would actually perceive such a difference between the PWA and Native implementation, remains an open question.

Installation Size

As with the Android version and the literature for the Android version, the PWA implementation also has a smaller installation size on iOS, when compared to the Native implementation. However, the difference is much smaller. The native application already is quite small, with only 2.05 MB. The PWA implementation has a larger installation size on iOS when compared to the same PWA on Android, but it is still smaller than the installation size of the Native app on iOS.

Since the PWA is 3.57 times smaller than the Native app on iOS, the results seem to support the hypothesis that the installation size is smaller on iOS as well. However, while this seems like a large reduction, it would be a smaller reduction than expected. One possible explanation is that Apple has introduced optimization's in Swift 5 that reduce the installation size by not including the dynamically linked libraries from the Swift library in the application [67] in combination with the still not mature support for PWAs on the platform.

6.2.2 Energy Consumption

Interpreting the results for the energy consumption values on iOS indicates that there is a significant difference in the energy consumption between the Native and PWA approaches. However, we have to be careful in making statements about these results, since the values provided by the instruments tool are rather abstract, quite varying and not backed by scientific validation of the tool. On the other side, we have performed many and long measurements in order to get this data, which reduces these risks to validity.

One important thing to notice is that from looking at the Q-Q plot in figure 5.13 and the Shapiro-Wilk test for normality, the data is far from normally distributed. This makes it more difficult to make statements about the actual differences in energy consumption. When we look at the box plots for the energy consumption on iOS in figure 5.12, we find that the difference in energy consumption does not seem to differ that much, but that the data is in favor of the PWA implementation. This result is similar to the results we obtained for the energy consumption in Android.

For now, we can say that the energy consumption of the PWA is at least not significantly worse than the Native version and that the results seem to indicate that energy consumption is even slightly less.

6.3 Applicability

The only issue that the results seem to indicate in relation to the applicability of the PWA technology as an alternative to traditional approaches is a slower launch time on iOS.

Since little was known about the behavior of PWAs on the iOS platform, this research gives insight into the general applicability of the PWA methodology as an alternative to traditional approaches for cross-platform development. Also, strong support is given for the applicability on Android with the validation of previous performance results with a different application and the addition of the energy consumption knowledge. Gaining insight into these differences will hopefully help future consideration for a mobile development approach.

Furthermore, the trend in support for PWAs on iOS seems to indicate that Apple is willing to embrace the technology. At this time, it is unclear what iOS 13 will bring as improvements for PWA support, but it might be that it will improve and that the results of the experiments in this research will be different for this next version of iOS.

6.3.1 Possible Improvements

There are several promising advancements in web development that might increase the performance of Web Apps and Progressive Web Apps even more. One of these techniques is WebAssembly, a programming language for the web that aims to be safe, fast, portable and compact [68]. The research by Haas et al. shows that the performance of WebAssembly code is much faster than the JavaScript implementation and that the resulting WebAssembly code is also smaller [68]. Low-level code like this might give rise to more complex applications for the web, like games or other heavy computing implementations. With the gaming engine Unity already supporting WebAssembly [69], it becomes more realistic for the near future to have games be implemented as PWAs rather than as Native Apps.

Another interesting development is that of static site generators like GatsbyJS that make it easier to create a PWA from existing data sources. It applies all the latest web standards by default, making it convenient to have fast and optimized applications without too much trouble [70]. It will be interesting to see whether developments like this will influence the rise of PWAs as a more widely used alternative to traditional approaches.

Even more, PWAs take the cross-platform approach to the next level by also becoming desktop applications. Of course, the Web App could be used on any device running a browser, but support for making the app feel like a native application on a desktop computer is also starting to take off. Google already introduced this with Chrome, with the PWA running in its own screen and context [71].

Chapter 7

Conclusion

This chapter concludes the work done is this research. First, we relate the work to the research questions, after which we provide several opportunities for future research into the subject of Progressive Web Apps.

This research was performed in order to contribute to the knowledge in the field of mobile development, specifically the field of Progressive Web Apps. Since little scientific research is done in this field, we believe that it is important to provide the community with the information needed to make informed decisions about an approach for mobile development. Therefore, we formulated the main research question whether Progressive Web Apps can be a viable alternative to the traditional approaches of Native and Web Apps in mobile development. Next, we scoped the research by assessing the viability with the questions whether there are differences in performance and energy consumption between the approaches, since these characteristics have been shown to be important in mobile development.

We have found that there are differences, but often in favor of the PWA implementation. For answering RQ 1.1, the performance differences between PWAs and Web Apps, we can clearly see that the PWA actually launches faster than the same application without the PWA functionalities. Most likely, this is due to the local installation of the app resources using the service worker, rather than fetching the whole website from a remote source every time. Since loading from storage is faster than loading from a network, this result was to be expected. Hence, we can answer this question with yes, in this research, there is a difference in performance between the web app and PWA approach and this is in favor of the PWA implementation on both iOS and Android.

Another research question this research was set up for to answer was whether there is a difference in performance between Native implementations on iOS and Android and the PWA (RQ 1.2). We have presented the somewhat surprising result that the PWA actually launches faster than the Native implementation on the Android platform. This seems to be due to the installation size of the application, meaning that a similar sized PWA would launch with similar speed as the Native implementation. We tested this hypothesis and found this to be the case. Since we have established that performance is an important aspect of the viability of a mobile development approach, this conclusion gives strong support for a positive answer to the main research question. For iOS, the native application still launches slightly faster than the PWA. Whether this can be attributed to the limited support for PWAs or better optimization for the native implementation, remains an open question.

Finally, we aimed to find whether there are differences in energy consumption between the native implementation on iOS and Android and the PWA (RQ 1.3). We found that there is a slight difference for both iOS and Android, although small. For both platforms, it seems to be the case that the PWA implementation consumes slightly less energy. While it can be argued that the significance of the results is not strong enough to conclude that energy consumption of a PWA is less than energy consumption of a native application, we can say at least that based on the results in this research the energy consumption of the PWA implementation is not worse than that of the Native implementation. Since we have established that energy consumption is an important factor in the acceptance of a mobile development approach, this conclusion gives some support for a positive answer to the main research question.

We have shown that the performance and energy consumption of a PWA is at least similar or even slightly better when compared to the traditional approaches of Native and Web in mobile development. Since these factors have been shown to be important for the adaption of a mobile development approach, the results in our research seem to support a positive answer to the main research question. Based on our measurements on the applications created for this research, we can say that the PWA approach seems to be a viable alternative to native and web app implementations in the context of performance and energy consumption. This is a promising conclusion for the adaption of PWAs in the industry.

7.1 Future Work

There are several more questions that are of relevance for the adaption of PWAs as an alternative to other approaches that are beyond the scope of this research due to time constraints. One of these is the question willing users are to adapt to the PWA technology. It would be interesting to know whether users are just as likely to engage with the application when it is implemented as a PWA rather than as an application from the app or play store.

Another open question is whether the security of a PWA can be warranted for sensitive applications. While a PWA is by default served over HTTPS, without research it can not be said indefinitely that a PWA is secure. For more sensitive information, it is of course important that this information can not be compromised. Were it to be the case that PWAs are inherently less safe than native or web apps, than

Furthermore, the performance of a PWA compared to another approach can also be measured using other metrics. For example the execution time of heavy tasks, or the efficiency of rendering. Since this research was, due to limitations in time, not able to measure everything that can be measured about an application, future research might aim to extend on the knowledge established in this research and further add support for the viability of PWAs as alternative to native and web app implementations

For iOS, the launch time of the native application was slightly faster than the launch time of the PWA implementation. It might be possible that the support for PWAs is going to be improved in future iOS versions, which might make the launch time similar. Future research can be performed to contradict the results in this thesis by performing an analysis with a newer iOS version.

Also, energy consumption can be done more accurately with the help of an external hardware power monitor, like the Monsoon Power monitor used by Ciman and Gaggi [3]. This could also validate the results from this research.

Finally, the development and maintenance effort could be an important factor to decide for an approach in mobile development. It would be interesting to research the effort developers have to take to implement their application as a PWA rather than another approach and how this implementation could be maintained over time.

Bibliography

[1] Peggy Anne Salz. “Monitoring mobile app performance”. In: *Journal of Direct, Data and Digital Marketing Practice* 15.3 (2014), pp. 219–221.

[2] Andreas Biørn-Hansen, Tim A Majchrzak, and Tor-Morten Grønli. “Progressive web apps for the unified development of mobile applications”. In: *International Conference on Web Information Systems and Technologies*. Springer. 2017, pp. 64–86.

[3] Matteo Ciman and Ombretta Gaggi. “An empirical analysis of energy consumption of cross-platform frameworks for mobile development”. In: *Pervasive and Mobile Computing* 39 (2017), pp. 214–230.

[4] Neha Pande et al. “Enhanced Web Application and Browsing Performance through Service-Worker Infusion Framework”. In: *2018 IEEE International Conference on Web Services (ICWS)*. IEEE. 2018, pp. 195–202.

[5] Isabelle Dalmasso et al. “Survey, comparison and evaluation of cross platform mobile application development tools”. In: *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*. IEEE. 2013, pp. 323–328.

[6] WeiMin Zhang et al. “Research on the user time difference threshold during the mobile terminal app launch process”. In: *2018 International Conference on Computer, Information and Telecommunication Systems (CITS)*. IEEE. 2018, pp. 1–5.

[7] Meiyappan Nagappan and Emad Shihab. “Future trends in software engineering research for mobile apps”. In: *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. Vol. 5. IEEE. 2016, pp. 21–32.

[8] Lamia Gaouar, Abdelkrim Benamar, and Fethi Tarik Bendimerad. “Model driven approaches to cross platform mobile development”. In: *Proceedings of the International Conference on Intelligent Information Processing, Security and Advanced Communication*. ACM. 2015, p. 19.

[9] Tim A Majchrzak, Andreas Biørn-Hansen, and Tor-Morten Grønli. “Progressive web apps: the definite approach to Cross-Platform development?” In: (2018).

[10] Alex Russell. *Progressive Web Apps: Escaping Tabs Without Losing Our Soul*. June 15, 2015. URL: `https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/`.

[11] Alex Russell. *What, Exactly, Makes Something A Progressive Web App?* Sept. 12, 2016. URL: `https://infrequently.org/2016/09/what-exactly-makes-something-a-progressive-web-app/`.

[12] Pavel Smutný. "Mobile development tools and cross-platform solutions". In: *Proceedings of the 13th International Carpathian Control Conference (ICCC)*. IEEE. 2012, pp. 653–656.

[13] Apple. *Swift: The Powerful programming language that is easy to learn.* URL: `https://developer.apple.com/swift/` (visited on 04/04/2019).

[14] Apple. *Xcode 10.* URL: `https://developer.apple.com/xcode/` (visited on 04/04/2019).

[15] Google. *Develop Android apps with Kotlin.* URL: `https://developer.android.com/kotlin` (visited on 04/04/2019).

[16] Google. *Android Studio.* URL: `https://developer.android.com/studio` (visited on 04/04/2019).

[17] Wafaa S El-Kassas et al. "Taxonomy of cross-platform mobile applications development approaches". In: *Ain Shams Engineering Journal* 8.2 (2017), pp. 163–190.

[18] Bob Frankston. "Progressive Web Apps". In: *IEEE Consumer Electronics Magazine* (Mar. 2018).

[19] Ivano Malavolta et al. "Assessing the impact of service workers on the energy efficiency of progressive web apps". In: *Proceedings of the 4th International Conference on Mobile Software Engineering and Systems.* IEEE Press. 2017, pp. 35–45.

[20] Marcos Caceres et al. *Web App Manifest.* Dec. 12, 2018. URL: `https://www.w3.org/TR/appmanifest/` (visited on 04/08/2019).

[21] Matt Gaunt. *Service Workers: an Introduction.* Feb. 12, 2019. URL: `https://developers.google.com/web/fundamentals/primers/service-workers/` (visited on 04/08/2019).

[22] Addy Osmani. *The App Shell Model.* Feb. 12, 2019. URL: `https://developers.google.com/web/fundamentals/architecture/app-shell` (visited on 04/08/2019).

[23] Rahul Roy-chowdhury. *The Modern Mobile Web: State of the Union.* May 18, 2017. URL: `https://developers.googleblog.com/2017/05/the-modern-mobile-web-state-of-union.html` (visited on 08/08/2019).

[24] Ben Galbraith. *The web: state of the union (Google I/O 2018).* May 8, 2018. (Visited on 04/08/2019).

[25] Luis Corral, Andrea Janes, and Tadas Remencius. "Potential advantages and disadvantages of multiplatform development frameworks–A vision on mobile environments". In: *Procedia Computer Science* 10 (2012), pp. 1202–1207.

[26] Compuware. *Mobile Apps: What Consumers Really Need and Want, A Global Study of Consumers' Expectations and Experiences of Mobile Applications.* Mar. 2013.

[27] Tammy Everts. "Rules for mobile performance optimization". In: *Communications of the ACM* 56.8 (2013), pp. 52–59.

[28] Jingtian Wang, Xiaoquan Wu, Jun Wei, et al. "Detect and optimize the energy consumption of mobile app through static analysis: an initial research". In: *Proceedings of the Fourth Asia-Pacific Symposium on Internetware*. ACM. 2012, p. 22.

[29] Claas Wilke et al. "Energy consumption and efficiency in mobile applications: A user feedback study". In: *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*. IEEE. 2013, pp. 134–141.

[30] Kaushik Dutta and Debra Vandermeer. "Caching to reduce mobile app energy consumption". In: *ACM Transactions on the Web (TWEB)* 12.1 (2018), p. 5.

[31] Igor Crk et al. "Understanding energy consumption of sensor enabled applications on mobile phones". In: *2009 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. IEEE. 2009, pp. 6885–6888.

[32] Trevor Pering et al. "Coolspots: reducing the power consumption of wireless mobile devices with multiple radio interfaces". In: *Proceedings of the 4th international conference on Mobile systems, applications and services*. ACM. 2006, pp. 220–232.

[33] H. Yan et al. "Client-Centered, Energy-Efficient Wireless Communication on IEEE 802.11b Networks". In: *IEEE Transactions on Mobile Computing* 5.11 (2006).

[34] Viktor Yberg. "Native-like Performance and User Experience with Progressive Web Apps". MA thesis. KTH Royal Institute of Technology, School of Electrical Engineering and Computer Science, 2018.

[35] Abhi Gambhir and Gaurav Raj. "Analysis of Cache in Service Worker and Performance Scoring of Progressive Web Application". In: *2018 International Conference on Advances in Computing and Communication Engineering (ICACCE)*. IEEE. 2018, pp. 294–299.

[36] Shubhie Panicker. *Paint Timing 1*. W3C Working Draft. https://www.w3.org/TR/2017/WD-paint-timing-20170907/. W3C, Sept. 2017.

[37] Mozilla. *PerformancePaintTiming*. Mar. 23, 2019. URL: `https://developer.mozilla.org/en-US/docs/Web/API/PerformancePaintTiming` (visited on 05/13/2019).

[38] Apple. *Logging Dynamic Loader Events*. July 23, 2012. URL: `https://developer.apple.com/library/archive/documentation/DeveloperTools/Conceptual/DynamicLibraries/100-Articles/LoggingDynamicLoaderEvents.html` (visited on 05/10/2019).

[39] Apple. *UIViewController: An object that manages a view hierarchy for your UIKit app*. URL: `https://developer.apple.com/documentation/uikit/uiviewcontroller` (visited on 05/10/2019).

[40] Android. *App startup time*. URL: `https://developer.android.com/topic/performance/vitals/launch-time` (visited on 05/10/2019).

[41] Google. *Lighthouse.* Aug. 21, 2018. (Visited on 04/16/2019).

[42] Mohammad Ashraful Hoque et al. "Modeling, profiling, and debugging the energy consumption of mobile devices". In: *ACM Computing Surveys (CSUR)* 48.3 (2016), p. 39.

[43] *Measure Energy Impact with Instruments.* Sept. 13, 2016. URL: `https://developer.apple.com/library/archive/documentation/Performance/Conceptual/EnergyGuide-iOS/MonitorEnergyWithInstruments.html` (visited on 04/09/2019).

[44] Grace Metri et al. "What is eating up battery life on my SmartPhone: A case study". In: *2012 International Conference on Energy Aware Computing.* IEEE. 2012, pp. 1–6.

[45] Yannick Mingashanga Kwete. *Power Consumption Testing for iOS.* 2013.

[46] GSMArena. *OnePlus X full Phone specification.* URL: `https://www.gsmarena.com/oneplus_x-7630.php` (visited on 04/11/2019).

[47] GSMArena. *Apple iPhone 7 full Phone specification.* URL: `https://www.gsmarena.com/apple_iphone_7-8064.php` (visited on 04/11/2019).

[48] GSMArena. *Samsung Galaxy J7 Duo full Phone specification.* URL: `https://www.gsmarena.com/samsung_galaxy_j7_duo-9153.php` (visited on 06/18/2019).

[49] GSMArena. *LG Nexus 5X full Phone specification.* URL: `https://www.gsmarena.com/lg_nexus_5x-7556.php` (visited on 06/18/2019).

[50] Jeongyeup Paek, Joongheon Kim, and Ramesh Govindan. "Energy-efficient rate-adaptive GPS-based positioning for smartphones". In: *Proceedings of the 8th international conference on Mobile systems, applications, and services.* ACM. 2010, pp. 299–314.

[51] J Patrick Royston. "An extension of Shapiro and Wilk's W test for normality to large samples". In: *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 31.2 (1982), pp. 115–124.

[52] Graeme D Ruxton. "The unequal variance t-test is an underused alternative to Student's t-test and the Mann–Whitney U test". In: *Behavioral Ecology* 17.4 (2006), pp. 688–690.

[53] Eva Skovlund and Grete U Fenstad. "Should we always choose a nonparametric test when comparing two apparently nonnormal distributions?" In: *Journal of clinical epidemiology* 54.1 (2001), pp. 86–92.

[54] Patrick E McKnight and Julius Najab. "Mann-Whitney U Test". In: *The Corsini encyclopedia of psychology* (2010), pp. 1–1.

[55] Frank J Massey Jr. "The Kolmogorov-Smirnov test for goodness of fit". In: *Journal of the American statistical Association* 46.253 (1951), pp. 68–78.

[56] David F Parkhurst. "Statistical Significance Tests: Equivalence and Reverse Tests Should Reduce Misinterpretation: Equivalence tests improve the logic of significance testing when demonstrating similarity is important, and reverse tests can help show that failure to reject a null hypothesis does not support that hypothesis". In: *Bioscience* 51.12 (2001), pp. 1051–1057.

[57] Donald J Schuirmann. “A comparison of the two one-sided tests procedure and the power approach for assessing the equivalence of average bioavailability”. In: *Journal of pharmacokinetics and biopharmaceutics* 15.6 (1987), pp. 657–680.

[58] Fabian Johannsen. “Progressive Web Applications and Code Complexity”. MA thesis. Linköping University, 2018.

[59] Ding Li et al. “Automated energy optimization of http requests for mobile applications”. In: *Proceedings of the 38th international conference on software engineering*. ACM. 2016, pp. 249–260.

[60] Cloud Four. *PWA Stats*. URL: `https://www.pwastats.com` (visited on 04/03/2019).

[61] Paul Armstrong. *Twitter Lite and High Performance React Progressive Web Apps at Scale*. Apr. 11, 2017. URL: `https://medium.com/@paularmstrong/twitter-lite-and-high-performance-react-progressive-web-apps-at-scale-d28a00e780a3` (visited on 04/18/2019).

[62] Google. *Alibaba*. June 23, 2017. URL: `https://developers.google.com/web/showcase/2016/alibaba` (visited on 04/18/2019).

[63] Google. *Flipkart triples time-on-site with Progressive Web App*. Feb. 9, 2017. URL: `https://developers.google.com/web/showcase/2016/flipkart` (visited on 04/18/2019).

[64] Google. *AliExpress*. Feb. 9, 2017. URL: `https://developers.google.com/web/showcase/2016/aliexpress` (visited on 04/18/2019).

[65] Angus Croll. *Building m.uber: Engineering a High-Performance Web App for the Global Market*. June 27, 2017. (Visited on 04/18/2019).

[66] The R Foundation. *Kolmogorov-Smirnov Tests*. URL: `https://stat.ethz.ch/R-manual/R-patched/library/stats/html/ks.test.html` (visited on 06/07/2019).

[67] Apple. *Swift 5 Release Notes for Xcode 10.2*. URL: `https://developer.apple.com/documentation/xcode_release_notes/xcode_10_2_release_notes/swift_5_release_notes_for_xcode_10_2?language=objc` (visited on 06/03/2019).

[68] Andreas Haas et al. “Bringing the web up to speed with WebAssembly”. In: *ACM SIGPLAN Notices*. Vol. 52. 6. ACM. 2017, pp. 185–200.

[69] Marco Trivellato. *WebAssembly is here!* Aug. 15, 2018. URL: `https://blogs.unity3d.com/2018/08/15/webassembly-is-here/` (visited on 07/04/2019).

[70] Ajay NS. *Why you should use GatsbyJS to build static sites*. Dec. 4, 2018. URL: `https://www.freecodecamp.org/news/why-you-should-use-gatsbyjs-to-build-static-sites-4f90eb6d1a7b/` (visited on 07/04/2019).

[71] Pete LePage. *Desktop Progressive Web Apps*. May 29, 2019. URL: `https://developers.google.com/web/progressive-web-apps/desktop` (visited on 06/04/2019).