Bachelor Thesis Project

Comparing Progressive Web Applications with Native Android Applications

- an evaluation of performance when it comes to response time

Authors: Rebecca Fransson,
Alexandre Driaguine
Supervisor: Johan Hagelbäck
Semester. VT 2017
Subject: Computer Science

Abstract

Web applications are often described as being cross-platform. They are accessible from a multitude of different web browsers, which in turn are running on a multitude of different operating systems. For a time now, developers have used many different tools to create cross-platform applications for mobile devices with web technologies. However, these applications fail to deliver when taken out of their native environment, and often do not feel native at all. Enter Progressive Web Applications, PWA. PWA's are applications written for the web with web technologies, running in a browser, but seasoned with some techniques that can make them behave like a native application when running on a mobile device. They are just ordinary web applications with native behaviour such as offline support, installability, and push notifications. The question that arises is - can this new type of web applications match Native Android Applications in performance, especially the response time when accessing the device's hardware? This report will try to answer that question

Keywords: Progressive Web Application, PWA, Native Android Application, Android Application, NAA, Hardware access, Response time.

Contents

1 Introduction

Web applications are often described as being cross-platform. They are accessible from a multitude of different browsers, running on different operating systems.

In 2014, the number of global users accessing the web on mobile devices surpassed those accessing it on a desktop [1]. This shows that making your web applications mobile-friendly is more important now than ever. Companies often see the need to develop native mobile applications to overcome the limitations that the web as a platform imposes on mobile devices. In many cases, they have to develop their application for both the web, iOS, and Android.

Developers have used web technologies to develop cross-platform mobile applications with tools like Cordova [8] and PhoneGap [9] for some time. These applications are installable from the respective operating systems application store, and run inside a native environment, with all features available to a native application. Taken out of this native environment, these applications fail to deliver this experience due to browser constraints. Progressive Web Applications (PWA) could solve this problem. A PWA is a web application that aims to deliver a native-like user experience on a mobile device, such as offline support and push notifications [2, 10].

This paper will investigate how PWA's compare to Native Android Applications (NAA) when it comes to response time when accessing parts of the mobile devices hardware.

1.1 Background

1.1.1 Progressive Web Applications

Progressive Web Application (PWA) is a buzzword in the technology sector of the web. Many blogs are talking about it, and you see big companies like Ali Express [4], Twitter [27], and Housing.com [5] adopting this new type of web

applications. PWA’s gives the user a reliable, fast and engaging experience, much like that of native application [2].
A PWA is a web application that is enhanced with some technologies that allow for native-like behaviour in a mobile device, while still functioning in a desktop browser. It can be added and launched from the home screen and should load instantly, regardless of the network connection. This is possible with the help of Service Workers [3], a JavaScript Worker [6] that allows caching content for offline access and for push notifications [7].

1.1.2 Service Workers

A service worker is a script that runs in the background of your browser. It is a JavaScript worker that runs separately from the web page, so it cannot access the Document Object Model (DOM), a programming interface for HTML documents [21], directly. Instead, it communicates with the web page through an interface. Service workers allow developers to use features such as push notifications and offline functionality, two of the things that make a web application progressive. They also allow one to control how to handle network requests, for example, to serve cached content [3].
Service workers are as of now supported by Firefox, Opera and Chrome browsers [11], and both Edge and Safari have shown hints of supporting them [12, 13].

1.2 Previous research

Research have shown that progressive enhancements improve the load times in Single Page Web Applications. Techniques such as code splitting and isomorphism improve page loads, and together with offline caching of content with service workers make a PWA load faster, even on flaky connections [17]. It will load faster both when accessing a page for the first time and when returning to it. These techniques will not be discussed in this paper but are still relevant to the research, since they are a part of the enhancements that make a

PWA.

Salma Charkaoui, Zakaria Adraoui and El Habib Benlahmar provided an insight into cross-platform development of mobile applications in their paper *Cross-platform mobile development approaches*, where *Javascript frameworks* were one of the approaches discussed [18]. The web has moved forward since this paper was published. For example, it states that a web application cannot access the Device API of a mobile device, which is not true anymore [16]. Some features are available to a web application today, such as camera, recording media and file access. This makes this research interesting today.

According to Andre Charland and Brian Lerous, "Web apps are cheaper to develop and deploy than native apps". In their paper *Mobile Application Development: Web vs. Native* from 2011, they state the not so controversial fact that native applications are faster and have better user experience than web applications on mobile devices. They further talk about PhoneGap, a framework for building native mobile applications with web technologies, and how it have bridged the native and web environments, allowing web applications to live in a native environment. The rundown is that the web has not achieved the level of performance that native code provides, but it is getting close, giving the example of Quake 3 running in the browser [19].

In 2016 Jan Steczko wrote a thesis about companies experiences with cross platform development compared to native development for mobile devices. Steczko interviewed 13 businesses in order to answer his problem. The thesis concludes that the companies preferred native development and that the advantages are stronger for native development. The companies thought that native applications were faster and could provide better user experience, and this compensates for the fact that creating two different applications for the mobile and the web are more expensive in both time and money. But the companies did also know that they could create cross-platform applications with lower cost and the development of these applications would be faster and simpler. According to Steczko's research, the choice between these two application strategies depended on several factors. For example complexity of the application, budget or quality [20].

1.3 Problem formulation

Many companies are facing the problem of developing different applications for different platforms. They often need to develop two mobile applications, one for iOS and one for Android. On top of that, they need a web application that works well on both a desktop and a mobile device [2, 10]. Web applications are somewhat limited today when compared to native mobile applications, but are moving forward all the time. We want to see if a Progressive Web Application can compete with a native application when it comes to performance, specifically response time of accessing different parts of the phone's hardware.

1.4 Motivation

Web development is moving forward at a blazing pace all the time. Mobile access of the web have already surpassed desktop access [1], and making web applications that works seamlessly on mobile devices is more important than ever.

Many companies are facing a challenge when developing applications [23]. Often they need to develop for three different platforms - iOS, Android and the web. Cross-platform frameworks like Cordova, Xamarin and React Native have tried to solve this for the mobile platforms with a write once and use everywhere approach. What if you could develop one application that worked on all these devices? This is what Progressive Web Applications wants to solve. This could save both time and money for many developers and companies that need to develop applications for all these platforms.

1.5 Research Question

	What limitations are Progressive Web Applications facing compared to Native Mobile Applications when it comes to hardware access of a mobile device ?
	How does a native application for Android compare against a Progressive Web Applications compare when it comes to performance of response time when accessing hardware?

With the first research question, RQ1, we want to gather information about what kind of limitations a Progressive Web Application faces when it comes to accessing different parts of a phones hardware compared to a Native Android Application.

With the second research question, RQ2, we want to answer how notable the difference is in performance of response time between Progressive Web Applications and Native Android Applications are.

1.6 Scope/Limitation

This thesis will focus on smaller pieces of code to be able to conclude more specific answers. All parameters to benchmarking cannot be explored due to time constraint, so response time was chosen, excluding parameters such as memory consumption, battery consumption, cpu profiling etc.
We chose to exclude user-test with the different applications. This decision was made because of the unreliability of the human factor.
Another limitation in our thesis is the availability of different devices. Our benchmarks will only be tested on two different Android phones, Oneplus X and Huawei Honor 8.
Another reservation is the limited time we have to conduct our experiment and case study. Because of this there is not enough time to be spent on testing all the different parts of hardware that the native and Progressive Web Application can access.
Due to the limitations that iOS has when it comes to Progressive Web Applications, iOS devices were excluded from this report. Therefore our experiment will be limited to Chrome in the Android devices.

1.7 Target group

The target group of this thesis are developers that need to develop applications for the web and multiple mobile platforms, and want to know if Progressive Web Applications are an alternative to Native Android Applications when it comes to performance. Furthermore, javascript developers who wants to target mobile users but lack the knowledge of developing Native Android Applications could benefit from this report.

1.8 Outline

Throughout this report experiments that will test the response time of a Native Android and a Progressive Web Application accessing a specific hardware for a mobile device will be conducted and documented.

First, the report will describe the scientific approach. In this chapter the method that will be used to find answers to the research questions will be described. The next chapter will contain information about how the experiments will be conducted, which tools that will be used, and how the experiments will be measured. After the Implementation chapter the report will present the results gathered from the experiments. These results will then be analysed and discussed. The report will finish with a conclusion. The raw results from the experiment can be found in the Appendix.

2 Method

In this chapter there will be a description of the scientific method that was used in order to answer the research questions.

2.1 Scientific Approach

To answer the first research question, RQ1, a qualitative literature study was made. This decision was made to find out what a web application, specifically a progressive one, can do today in terms of native-like behaviour. The result of this literature study served as a stepping stone to the second question.
To answer the second research question, RQ1, a quantitative controlled experiment was made. Native Android development faced Progressive Web Application development in a series of performance benchmarks to determine if a PWA can match a native application in terms of response time when accessing hardware.

2.2 Method Description

To answer the first research question, a search was made with a set of keywords: Progressive Web Applications, access hardware. The information gathered from the search results was summarized and used as a basis for the second research question.
To answer the second research question, a suite of micro-benchmarks was made. Each benchmark was contained in its own application, to minimize the effect of background processes and other components that could slow down the benchmark. The data gathered from the benchmarks produced graphs with information about how much time the applications took when accessing the hardware. The graphs shows 54 dots that represent each run.
There were independent variables that could have made a difference in the result. One of these was the mobile devices that were used to run the benchmarks. Background processes could have impacted the result in a

negative way. Therefore, the benchmarks were run 54 times to give a better view of the actual result.

2.2.1 Android

The applications containing the Native Android benchmarks were written in the Java programming language together with the Android Java API. Android Studio was the development environment of choice for developing the applications and benchmarks for the native applications.

2.2.2 Progressive Web Application

The applications containing the PWA benchmarks were written in the JavaScript programming language with the help of the React library. Visual Studio Code was the development environment of choice for developing the applications and benchmarks.

2.2.3 Devices

Two different Android devices were used to execute the benchmarks. These devices where:

- Huawei Honor 8
- OnePlus X

Both devices have Android operating system 6.0.1.

2.3 Reliability and Validity

Due to the limited access of hardware, the experiment was only conducted on two different Android devices. Other devices than those two specified might produce different results when executing the benchmarks. This is an external validity issue, but this should not harm the internal validity and reliability of the experiment.

One internal validity issue is that the code for the Android benchmarks do not match the code for the PWA benchmarks exactly, due to them being written in different programming languages. This could impact the result of the benchmarks.

Another external validity issue is that the benchmarks are self-contained within small applications. This decision was made to eliminate outside factors, and the results could be different when applied to a real-world application.

One issue with the reliability of the benchmark results is that they could vary due to background processes on the mobile devices interrupting their work. To try to eliminate this issue, the benchmarks will be run multiple times.

If the experiment is conducted again with the same hardware and the same version of software, the result should be close enough to be considered the same as the result in this report.

3 Implementation

This chapter will describe the implementation of the experiments that was conducted to answer RQ2.

3.1 Hardware access

Before conducting the experiments that answers the second research question, RQ2, the first research question, RQ1, had to be answered. A list of all different parts of a mobile devices hardware was gathered, and the results can be viewed in chapter 4 - Results. Testing all the items on the list was impossible due to time constraints, so two of them was selected for the experiment. These two were chosen because of their importance. Both the camera and the geolocation is used in many different native applications and therefore these were more interesting for this report.

3.1.1 Camera

Some of the most popular applications for mobile devices make heavy use of the camera. Instagram and Snapchat are two examples that implement their own custom camera view instead of using the built in application that comes with the mobile device. Facebook has also followed this trend.

3.1.2 Geolocation

Geolocation was chosen because it is used in many applications to give the user a more personalised experience based in their location. Applications can, based on the user's location, serve a map containing the nearest restaurants, breweries, convenience stores, public toilets etc.

3.2 Applications

Every item to be benchmarked was self contained within it’s own application. For each part, two applications were created - one native Android application, and one Progressive Web Application.

The Progressive Web Application was written with React, as React has become a popular Javascript library. React grew over 300% percent in popularity between 2015 and 2016 [24].

In the starting view of the application a button was placed. When the button was pressed by the user the application stated accessing the hardware that the application was build around. At the same time the application started the benchmark.

In Figure 3.1 and 3.2 you will see a flowchart for the NAA and the PWA accessing the camera and saving the taken image. The camera took the picture without a flashlight.

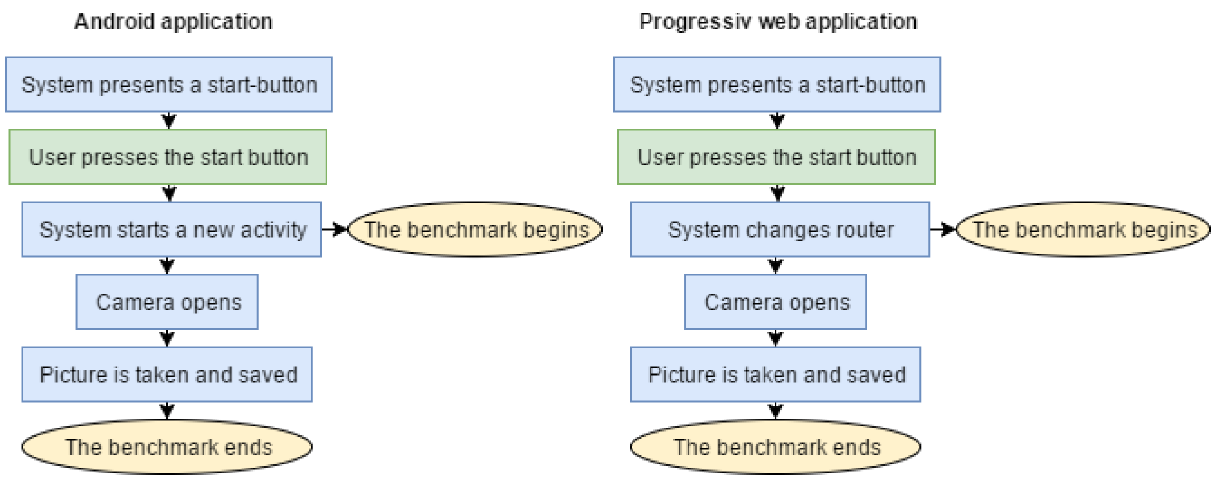
Android application
System presents a start-button
User presses the start button
System starts a new activity
The benchmark begins
Camera opens
Picture is taken and saved
The benchmark ends
Progressiv web application
System presents a start-button
User presses the start button
System changes router
The benchmark begins
Camera opens
Picture is taken and saved
The benchmark ends

Figure 3.1 - Workflow of the NAA accessing the camera

Figure 3.2 - Workflow of the PWA accessing the camera

The following two figures, Figure 3.3 and 3.4, shows the NAA and the PWA accessing the location of the user and displaying it on a map.

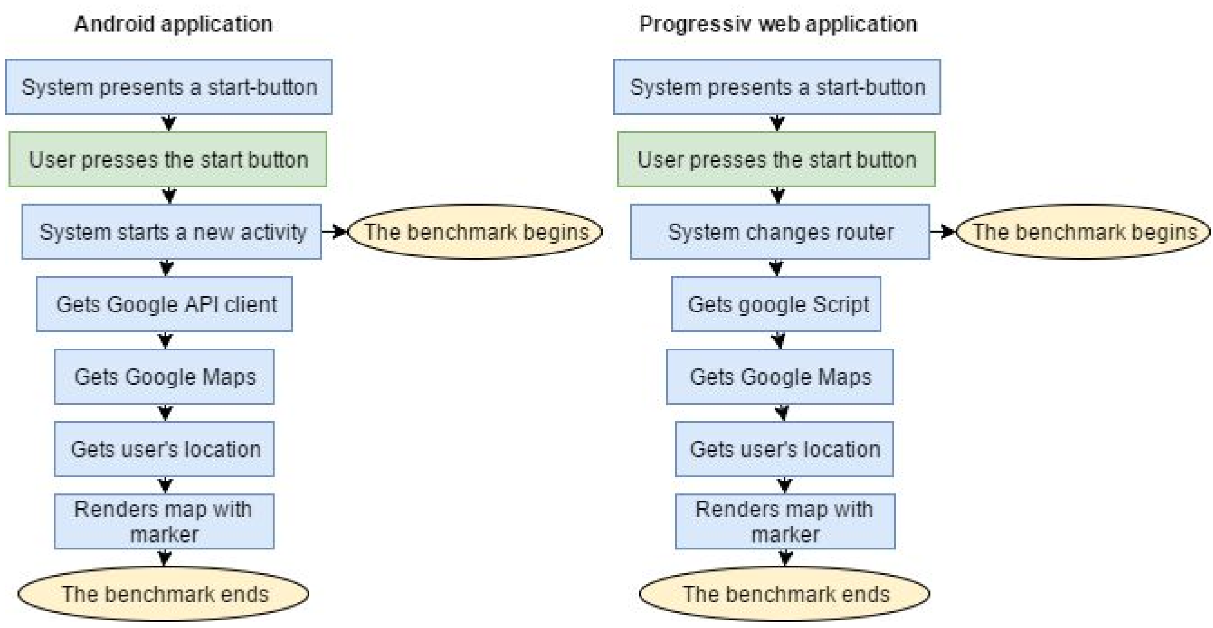

Figure 3.1 - Workflow of the NAA accessing the geolocation

Figure 3.2 - Workflow of the PWA accessing the geolocation

3.3 Benchmarks

The Progressive Web Applications was benchmarked with the Web Performance API [22], an API that allows web pages to access methods to measure performance. Marks were set in the code, one for the start of the benchmark, and one for the end of it. When the benchmark was done, the total time was calculated.

The benchmarks for the Native Android Applications were done with the help of Guava, a library made made by Google[26], specifically their Stopwatch class.

An example of the application accessing the camera would start when the user pressed the button in the application. When the application started to access the camera the benchmarks would also start. When the camera opened successfully the application would take a picture and save it. When the picture was successfully saved the benchmarks would stop.

3.4 Experimental features

When conducting the experiments the authors decided to try out some experimental features that were hidden behind flags in the Chrome web browser. Specifically, the flag that was turned on was the experimental web platform features. This flag allowed accessing the Image Capture API [25], which gives the developer more control over the camera - taking pictures with higher resolution, zooming etc. A decision was made to create two PWA benchmarks for the camera, one with the new, experimental ImageCapture API, and one with the classic canvas method. Since the canvas method does not give truly 4K photos, this benchmark was further divided into two parts - one with 4K resolution and one with 720p resolution. The motivation for this was to show how the canvas method stands against Android when processing in 4K, while still showing how it performs when used with a resolution it can render.

4 Results

4.1 Accessing hardware from mobile devices

The following information is gathered from WhatCanTheWebDo.Today [16]. This site provided a list over which hardware that the web could access in the Chrome browser that was installed on the two different mobile devices. The list is presented in the following table, Table 4.1.

Native behaviors ● Push messages ● Foreground Detection ● Permissions	**Camera and Microphone** ● Audio & video capture ● Recording media ● Real-time communication	**Input** ● Touch gestures ● Speech recognition ● Clipboard ● Pointing device adaptation
Surroundings ● Bluetooth	**Seamless experience** ● Offline mode ● Background sync	**Operating system** ● Offline storage ● File access Storage quotas
Device features ● Network type & speed ● Online state ● Vibration ● Battery status	**Screen & output** ● Fullscreen ● Screen orientations & lock ● Presentations features	**Location & position** ● Geolocation ● Device orientation ● Device motions

Table 4.1 - List of the hardware and native-like behaviour that can be accessed with OnePlus and Huawei device.

There were five areas that the web couldn't access from the mobile devices. There were: Local Notifications, Proximity sensor, Geofencing, Wake lock and Contacts.

4.2 Response time when accessing hardware

To answer the second research question, RQ2, data from the benchmarks discussed in chapter 3, Implementation, were gathered. The benchmarks were run on two different Android devices, Huawei Honor 8 and OnePlus X. Both devices used the Android 6.0.1 operating system, and the web browser used was Chrome 57.

The following figures, Figure 4.2 to 4.8, shows 54 dots that represent the milliseconds for each benchmark run. Every graph contains a trendline to better understand the difference in milliseconds that happens with every run.

Tables 4.2 to 4.5 shows a summary of each sample. The mean, median, standard deviation, and standard error of the mean are all displayed in milliseconds. Standard deviation and standard error of the mean were included to give the reader an idea of how varied the data is. A more detailed view of the variation can be found in Chapter 5, Analysis. The full raw data was too big to include in this chapter, and can be found in Appendix 1.

4.2.1 Android's camera benchmarks

The first benchmark on the OnePlus X device took longer to compute than the benchmark for the Huawei device, whole 1008 milliseconds. The rest of the benchmarks was between the value 970 and 826 milliseconds.

The first benchmark for the Huawei device took 779 milliseconds. After the first run the rest of the benchmarks was in the range between 745 and 778 milliseconds.

Device	**Sample**	**Number of samples**	**Mean**	**Median**	**Standard Deviation**	**Standard error of the mean**
Huawei Honor 8	Android camera_2 API	54	768.17	770	6.93	0.94
OnePlus X	Android camera_2 API	54	899.12	904	37.22	5.07

Table 4.2 - Response time of camera in a NAA

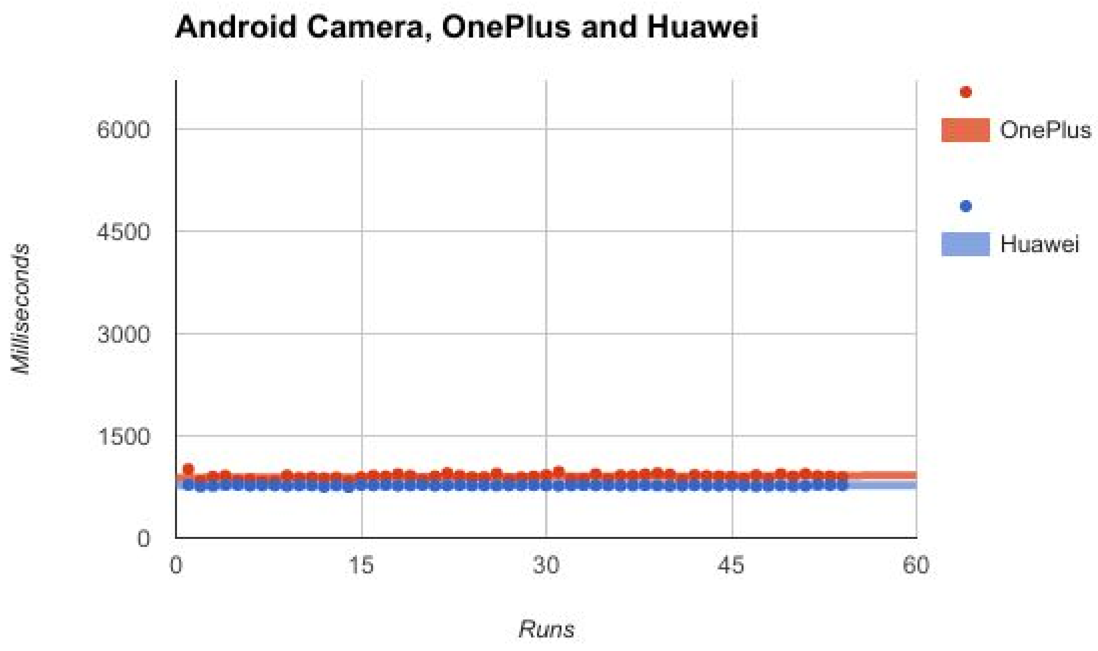

Figure 4.1- Comparing the NAA Camera response time.

4.2.2 Progressive Web Application's camera benchmarks

The Progressive Web Applications was divided and two solutions to this application was created. The first one created the image took by the camera with the help of a canvas. Because of this, the data from two image resolutions created with canvas will be presented. The second application created the image taken by the camera with ImageCapture API. The data for both applications will presented in this chapter.

The benchmark for the camera hardware response time was divided into two parts - one with the canvas rendering technique, and one with the experimental ImageCapture API.

Device	**Sample**	**Number of samples**	**Mean**	**Median**	**Standard Deviation**	**Standard error of the mean**
Huawei Honor 8	ImageCapture API	54	812.94	1115.43	385.36	52.44
Huawei Honor 8	Canvas 11.8 MP	54	6136.08	6136.77	271.03	36.88
Huawei Honor 8	Canvas 0.9 MP	54	562.32	556.91	37.51	6.10
OnePlus X	Canvas 11.8 MP	54	5882.24	5833.37	296.45	40.34
OnePlus X	Canvas 0.9 MP	54	1114.25	1078.54	135.84	18.48

Table 4.3 - Response time of camera in a PWA

4.2.2.1 Canvas on the OnePlus mobile device

The benchmarks with the highest resolution, 11.8MP, on the OnePlus device showed results over several seconds long. The first one did not take longer to compute than the rest of them. The benchmarks were in the range between 6747 and 5245 milliseconds.

The benchmarks with the lower resolution showed significant slower than the benchmarks with the higher resolution did. The benchmarks stayed between 920 and 1568 milliseconds.

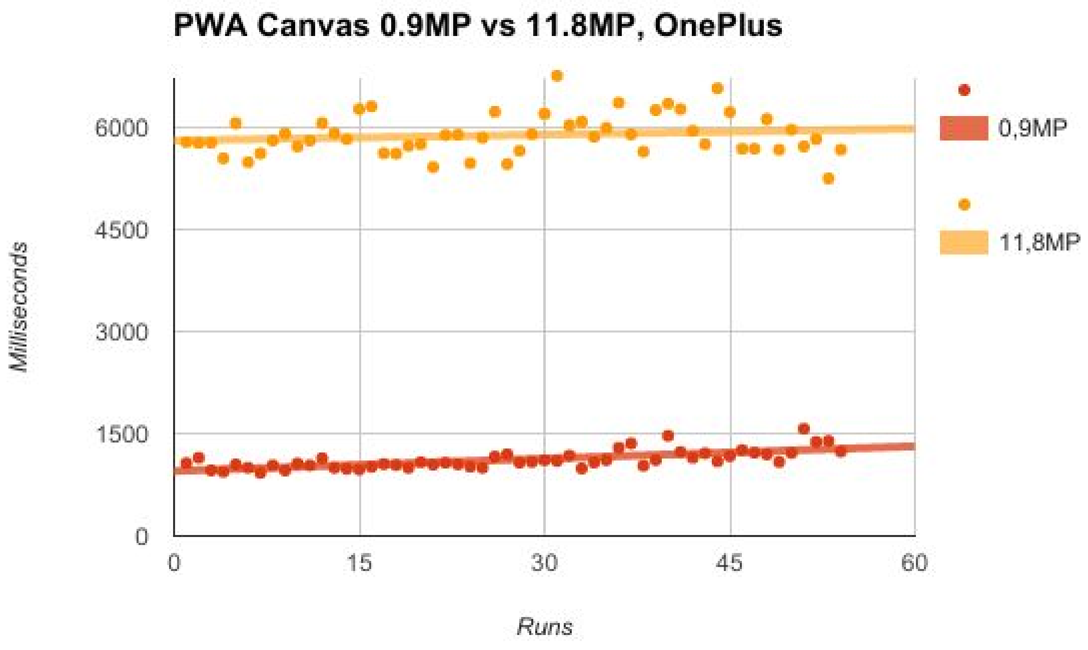

Figure 4.2 - Response time for the PWA on the OnePlus device with the canvas-solution in two resolutions

4.2.2.2 Canvas on the Huawei mobile device

The benchmarks carried out on the Huawei device with the highest resolution, 11.8MP, showed a result of a slower response time than the benchmarks runned on the OnePlus device. But when runned with the lowest resolution the benchmarks on the Huawei device were much faster than the ones on the OnePlus device. The benchmarks with the resolution of 0.9MP took around 576 milliseconds and the benchmarks with the 11.8MP resolution took around 6045 milliseconds.

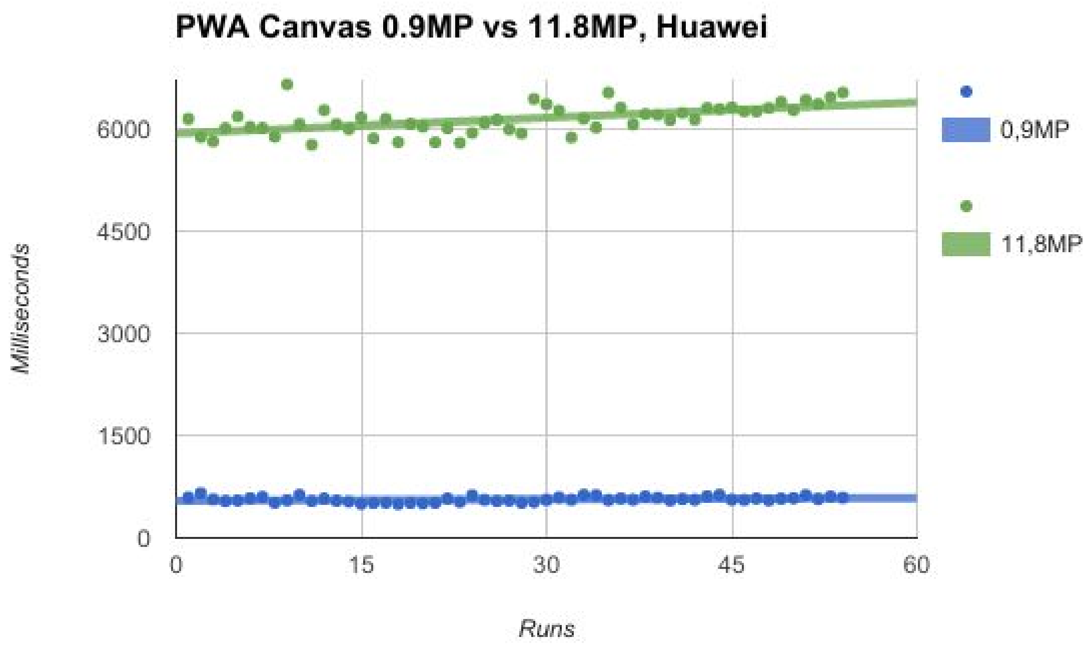

Figure 4.3 - Response time for the PWA on the Huawei device with the canvas-solution in two resolutions

The following figures, Figure 4.4 and Figure 4.5, shows the time difference between the high and the low resolution on the different devices.

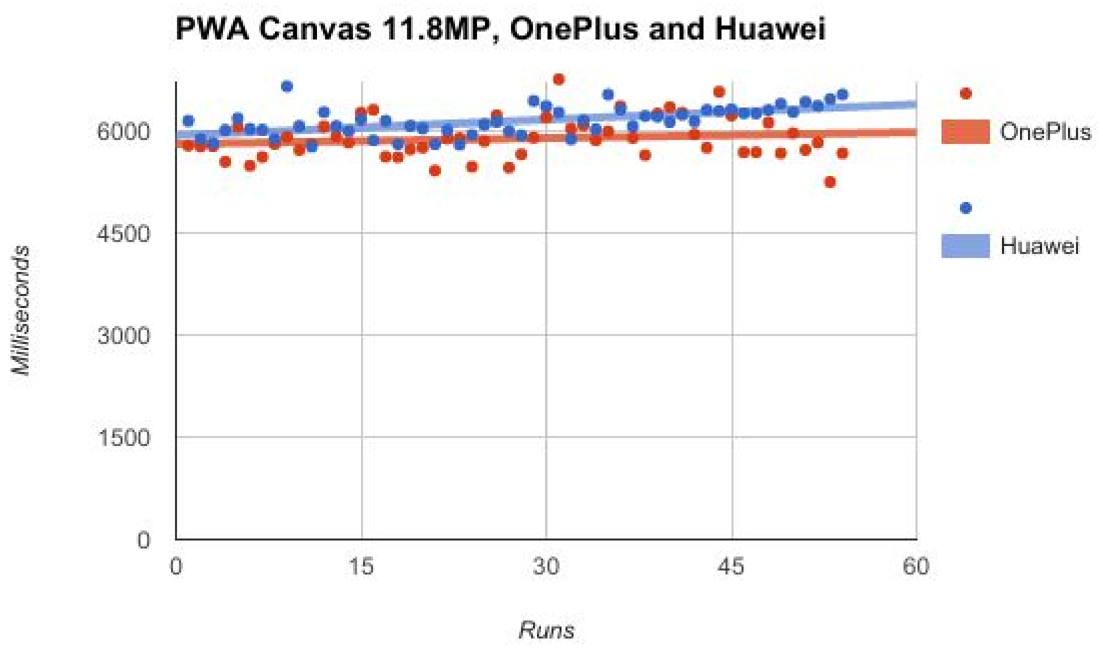

Figure 4.4 - Response time for the PWA with the canvas-solution on the two devices with the highest resolution

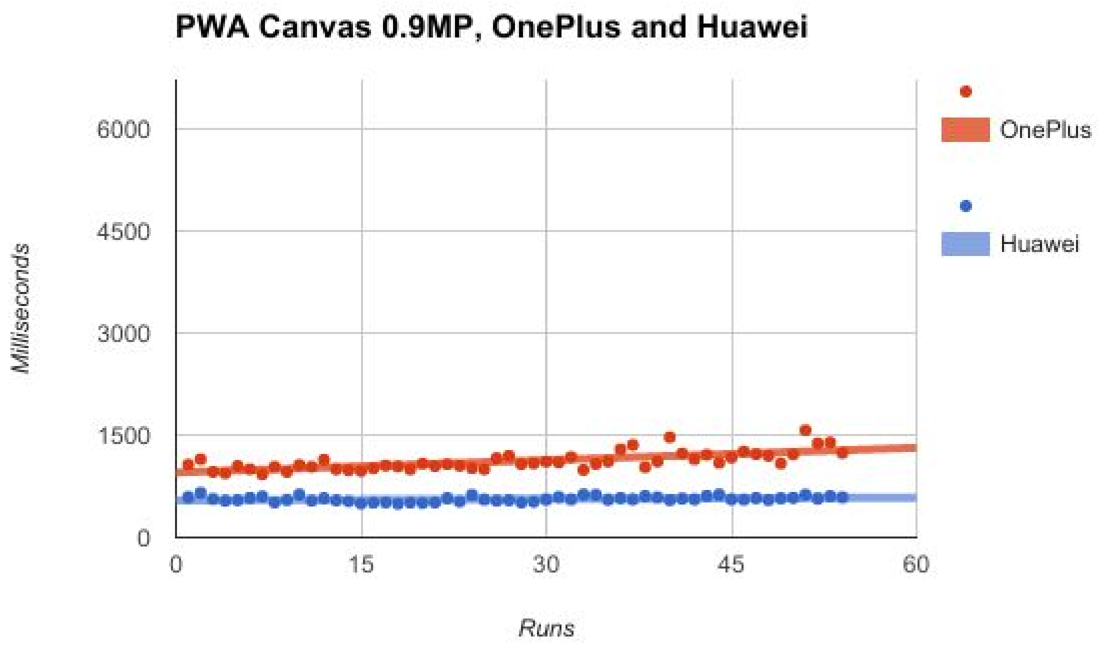

Figure 4.5 - Response time for the PWA with the canvas-solution

on the two devices with the lowest resolution

4.2.1.3 ImageCapture API

The first benchmark for the Huawei device took much longer, whole 1388 milliseconds. The rest of the benchmarks was within 628 and 1051 milliseconds.

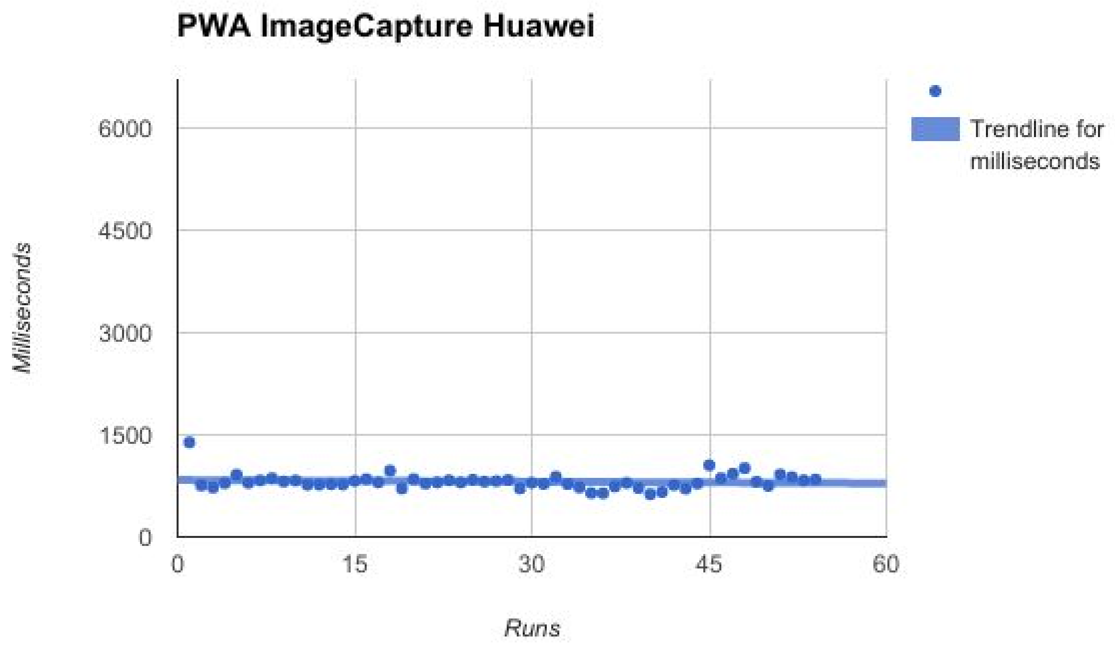

Figure 4.6 - Response time for the PWA with the ImageCapute-solution on the Huawei device

The ImageCapture API was only benchmarked on the Huawei because it did not work as expected on the OnePlus device.

4.2.3 Android's geolocation benchmarks

The result from the OnePlus geolocation benchmarks were more spread out than the ones from the Huawei device. The OnePlus device's results ranged from 875 to 1482 milliseconds, while the ones from the Huawei device ranged between 682 to 923 milliseconds

Device	**Sample**	**Numbe of samples**	**Mean**	**Median**	**Standard Deviation**	**Standard error of the mean**
Huawei Honor 8	Android geolocation	54	715.18	708.28	34.70	4.72
OnePlus X	Android geolocation	54	1164.20	1172.72	137.09	18.66

Table 4.4 - Response time of geolocation in a NAA

As we can see in the following figure, Figure 4.7, the benchmarks for the OnePlus increased several milliseconds after each run but the Huawei device's trendline stayed more horizontal.

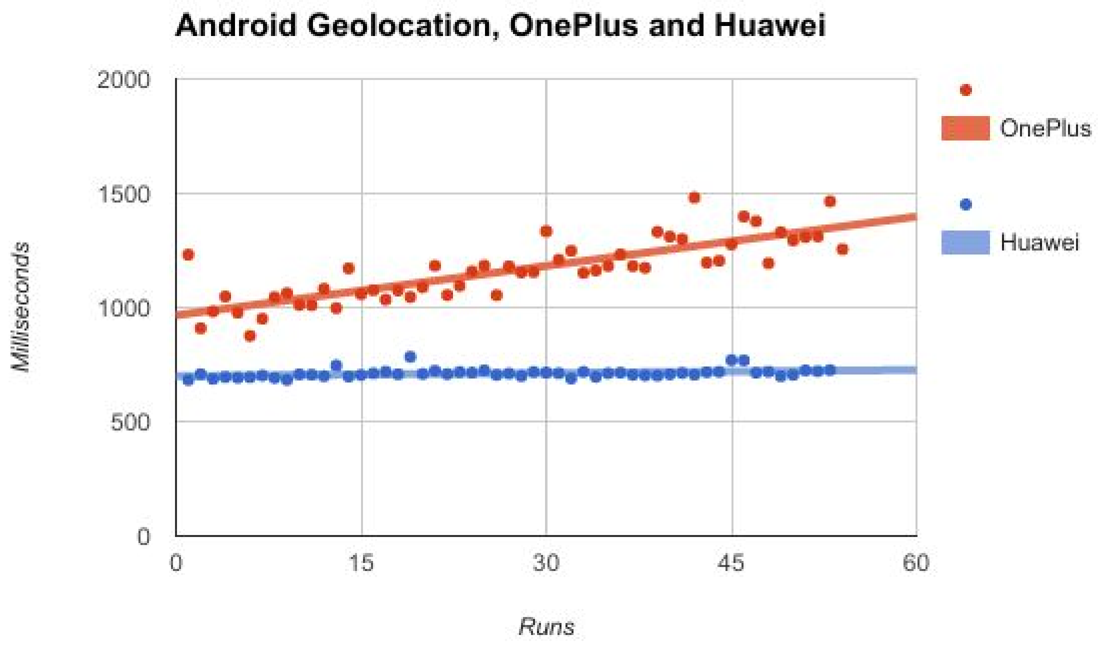

Figure 4.7 - Response time for the Android geolocation on both devices

4.2.4 Progressive web application's geolocation benchmarks

The benchmarks for the OnePlus device stayed between the range of 593 and 1856 milliseconds. The benchmarks for the Huawei were in the range between 357 and 730 milliseconds.

Device	**Sample**	**Number of samples**	**Mean**	**Median**	**Standard Deviation**	**Standard error of the mean**
Huawei Honor 8	PWA geolocation	54	517.28	507.43	75.11	10.22
OnePlus X	PWA geolocation	54	748.27	719.13	168.04	22.87

Table 4.5 - Response time of geolocation in a PWA

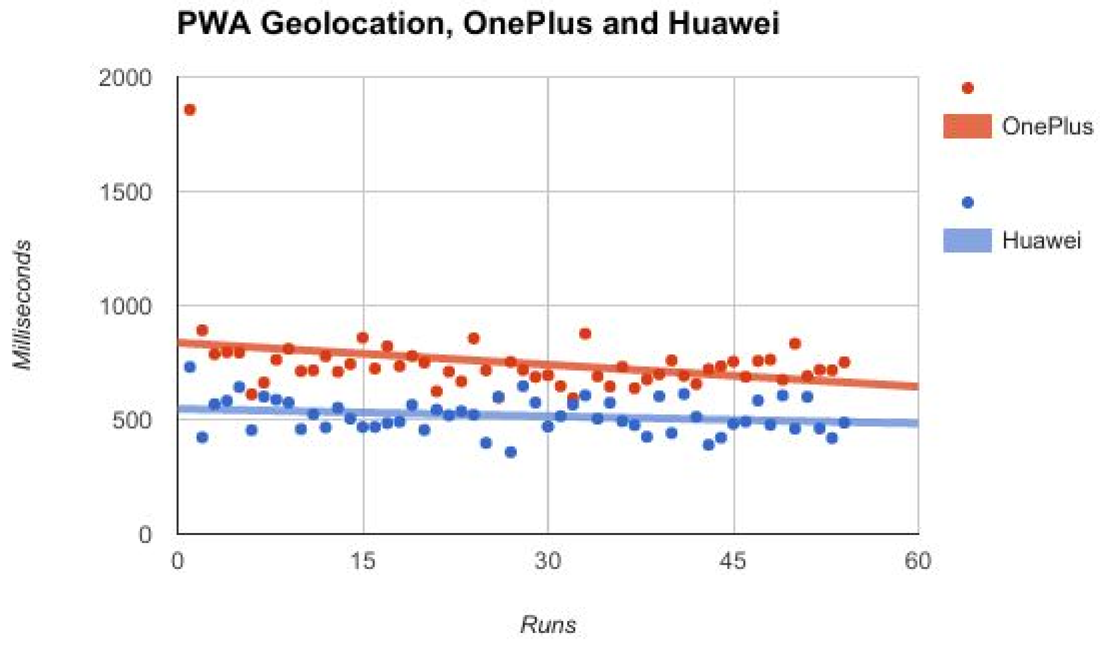

Figure 4.8 - Response time for the PWA geolocation on both devices

5 Analysis

Here follows an analysis of the results gathered when conducting the experiment.

5.1 Camera

Unsurprisingly, the benchmarks for the camera showed that a Native Android Application is significantly faster. The only time the PWA outshined the NAA was when we used a lower resolution with the canvas method. Processing the canvas to a picture when the resolution was set to 4K was way to slow to even be considered using in a real world production application. The fact is that this method does not give you an image with true 4K resolution, hence why the canvas method was also benchmarked with a lower resolution.

If we compare the results of the benchmarks using the experimental ImageCapture API from Chrome, the results are more interesting. The results were significantly faster with the Android application, which was concluded after doing a T-test with the mean values, resulting in a P-value of 0,005576. However, the the results show that the API for capturing images from within a browser is getting better. Since the ImageCapture API is still only an experimental feature, we can expect that it will only get better by the time it is shipped with browsers.

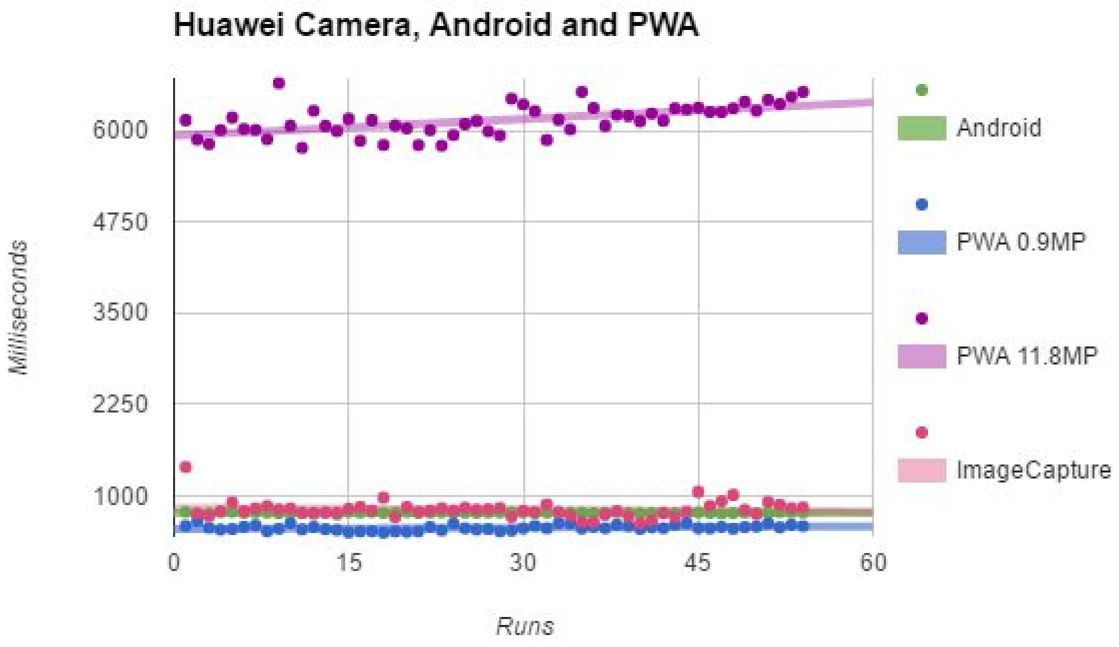

Figure 5.1 - Response time for the Huawei device for all the camera applications

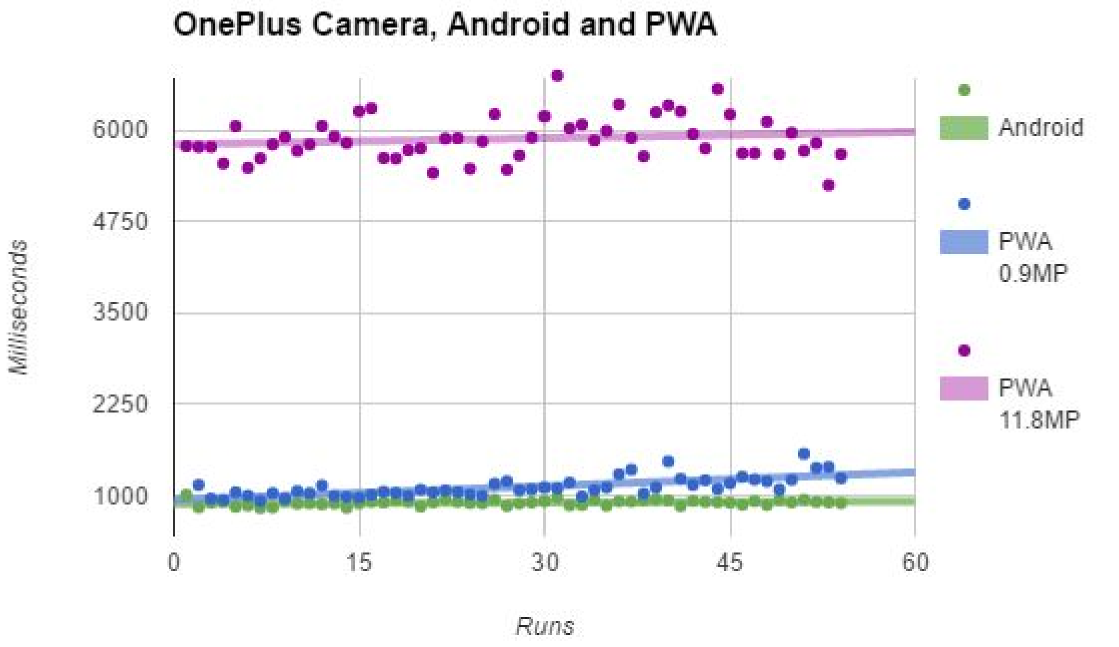

Figure 5.2 - Response time for the OnePlus device for all the camera applications

5.2 Geolocation

Looking at the two figures below we can see that just as with the response time for the camera benchmarks, the first benchmark showed a slower response time. This was probably due to the fact that Google Maps had to be initialized, and the tiles for the map had to be downloaded. For the rest of the runs, this was cached, showing an increase in speed.

While the benchmarks for the camera showed that NAA's are faster than PWA's, the result of the geolocation benchmarks showed another truth. Here, the PWA was faster than the NAA on all occasions, even speeding up a little bit at the end. Looking at the NAA on Huawei Honor 8, Figure 5.3 shows a steady pace throughout the runs, while it slows down over time on the OnePlus X device, probably due to memory management and consumption.

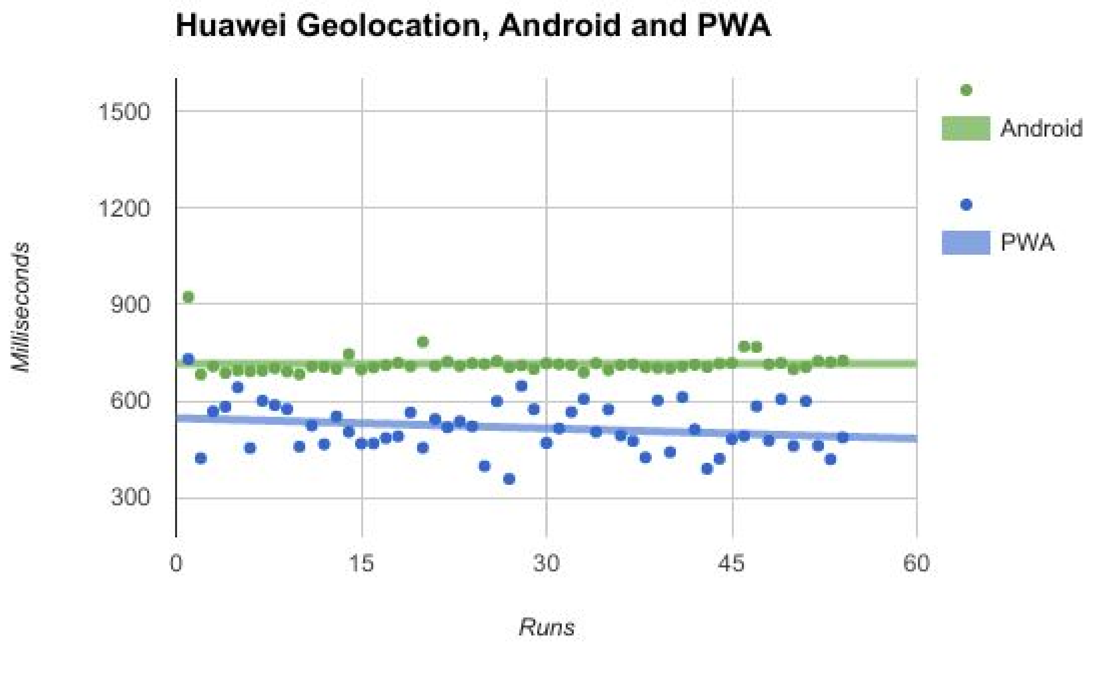

Figure 5.3 - Response time for the Huawei device for both geolocation applications

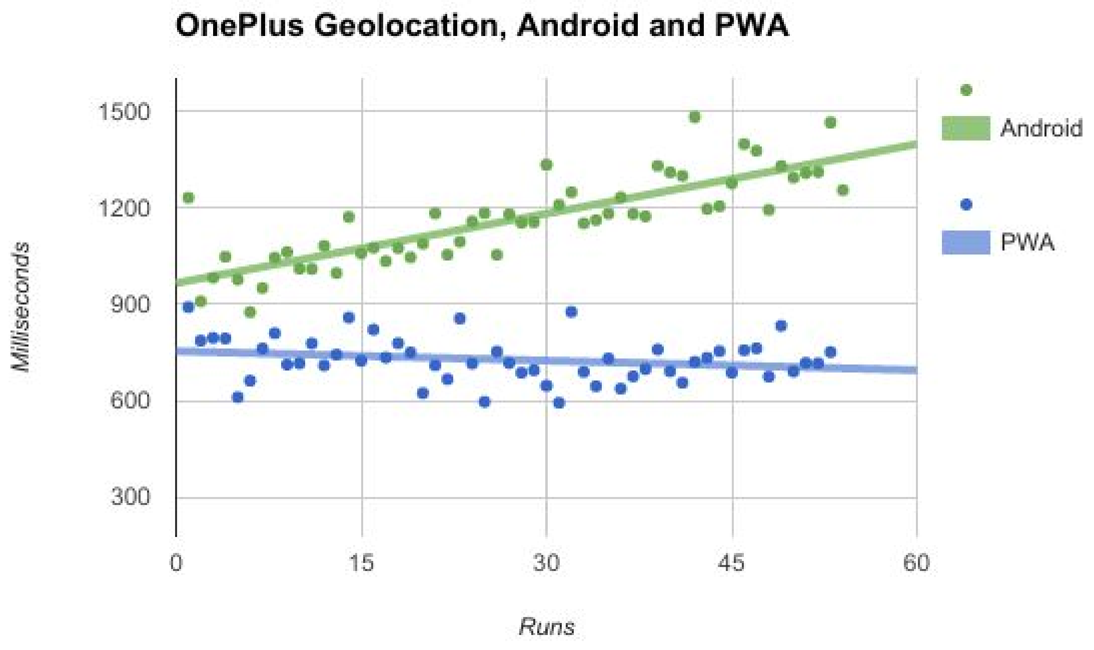

Figure 5.4 - Response time for the OnePlus device for both geolocation applications

5.3 Statistical analysis

How did the Progressive Web Applications and the Native Android Applications compare? More importantly, was the result statistically significant? Here follows a statistical analysis of the results, as well as figures showing the normal distribution compared to the real distribution of the results. The p value was calculated with a T-test to show if the difference was statistically significant.

5.3.1 Camera

Table 5.1 shows the different samples from the camera benchmarks compared to each other. The NAA sample of each device is compared to the PWA samples, together with the p-value of a T-Test and if the results were statistically significant.

Figure 5.1 to Figure 5.7 shows the normal distribution of the benchmarks, together with a histogram of the actual results. The figures shows that the normal distribution is followed approximately, showing some differences.

Sample 1	Sample 2	p value	Statistically significant
Android Honor 8	PWA ImageCapture Honor 8	0.0056	Yes
Android Honor 8	PWA Canvas 0.9 MP Honor 8	< 0.0001	Yes
Android Honor 8	PWA Canvas 11.8 MP Honor 8	< 0.0001	Yes
Android OnePlus X	PWA Canvas 0.9 MP OnePlus	< 0.0001	Yes
Android OnePlus X	PWA Canvas 11.8 MP OnePlus	< 0.0001	Yes

Table 5.1: P value of a T-test with the different samples of the camera benchmarks

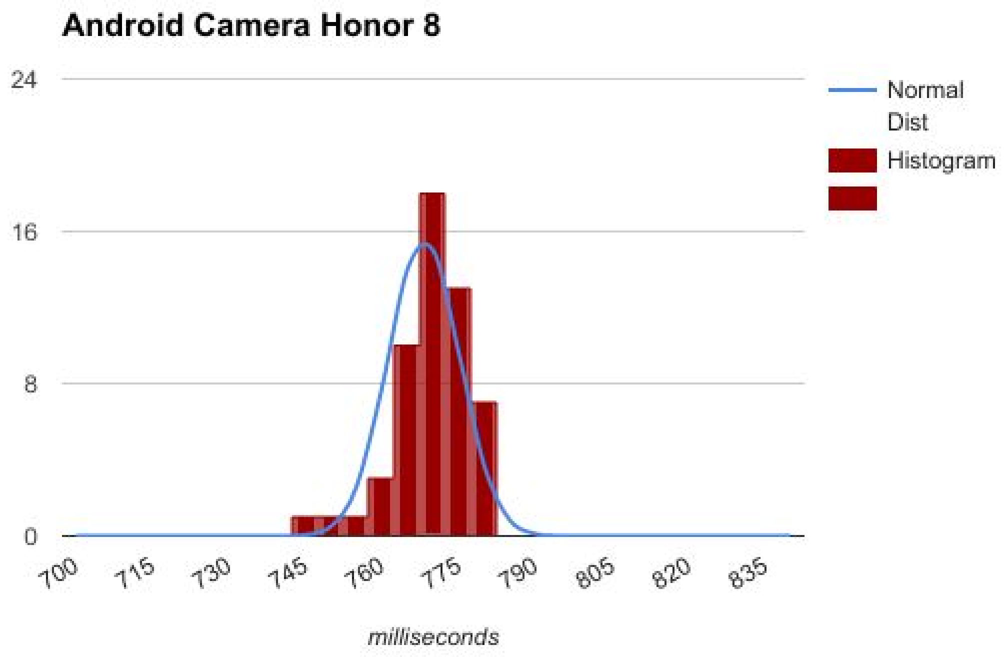

Figure 5.1: Normal distribution vs. Histogram of the Android Camera benchmarks

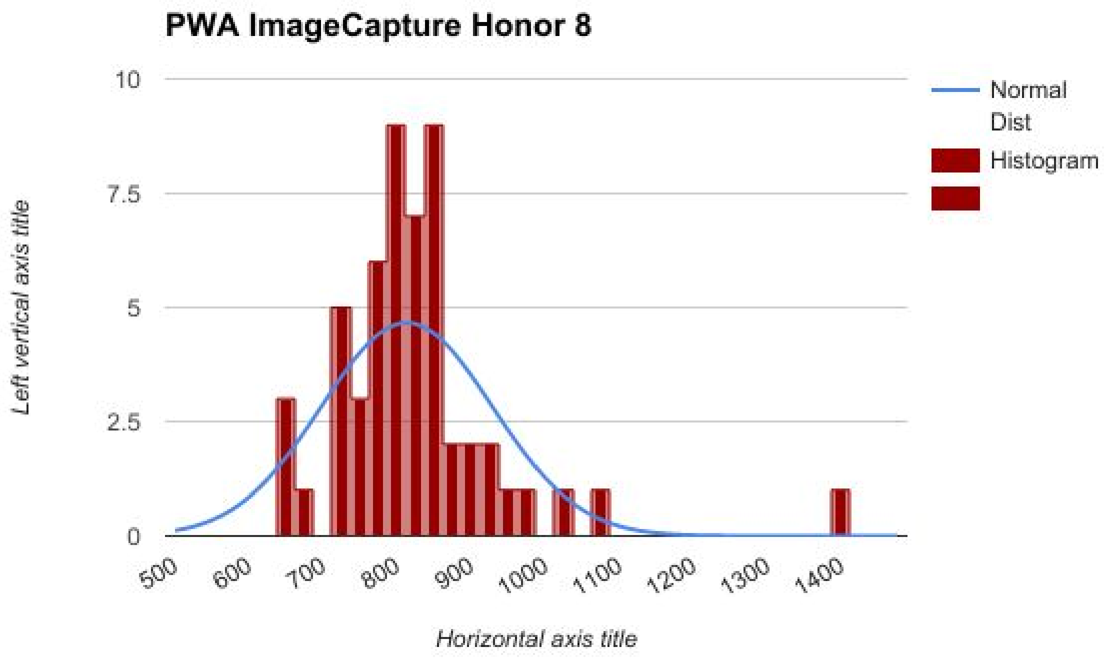

Figure 5.2: Normal distribution vs. Histogram of the PWA Camera ImageCapture
Honor 8 benchmarks

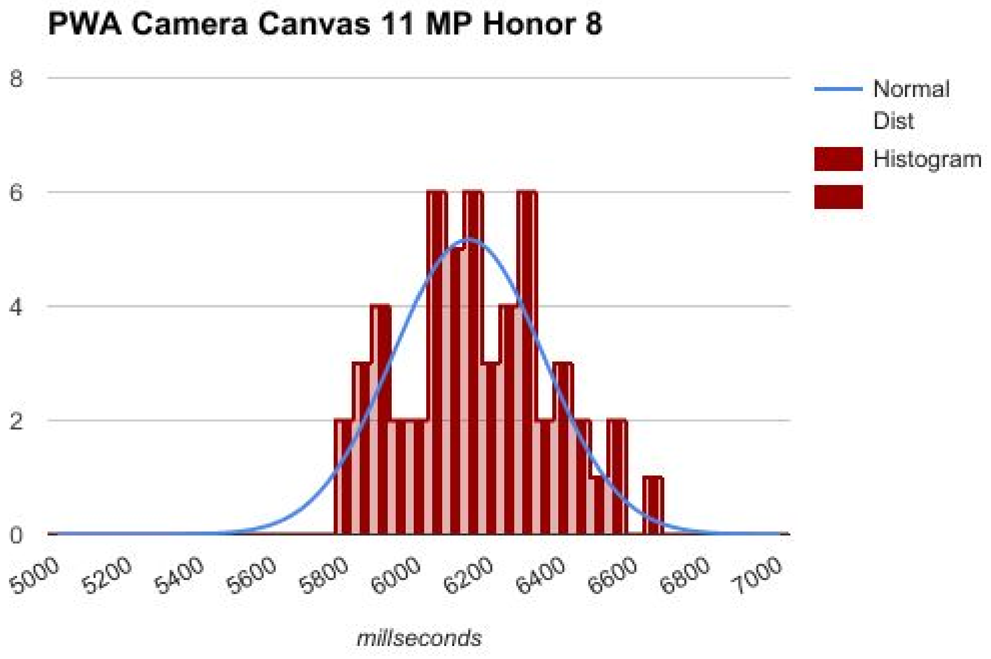

Figure 5.3: Normal distribution vs. Histogram of the PWA Camera Canvas 11 MP
Honor 8 benchmarks

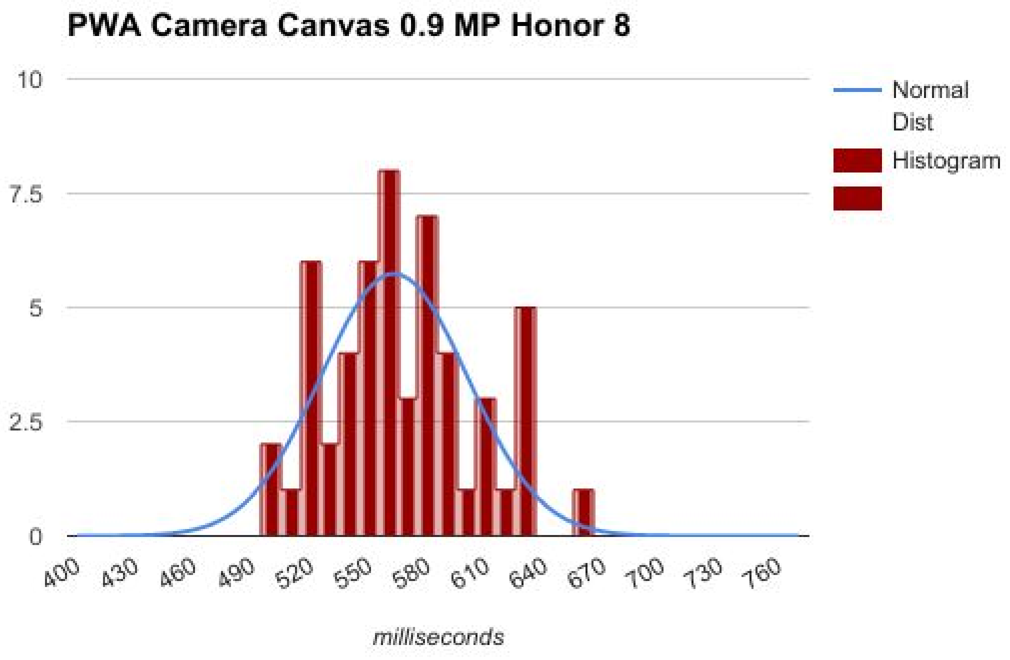

Figure 5.4: Normal distribution vs. Histogram of the PWA Camera Canvas 0.9 MP Honor 8 benchmarks

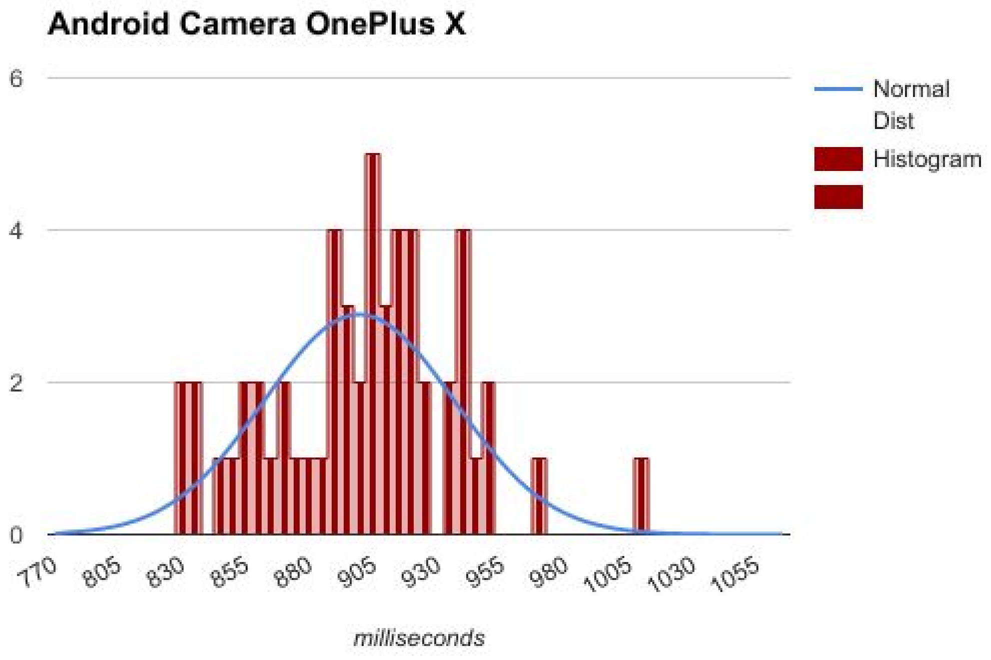

Figure 5.5: Normal distribution vs. Histogram of the Android Camera OnePlus X benchmarks

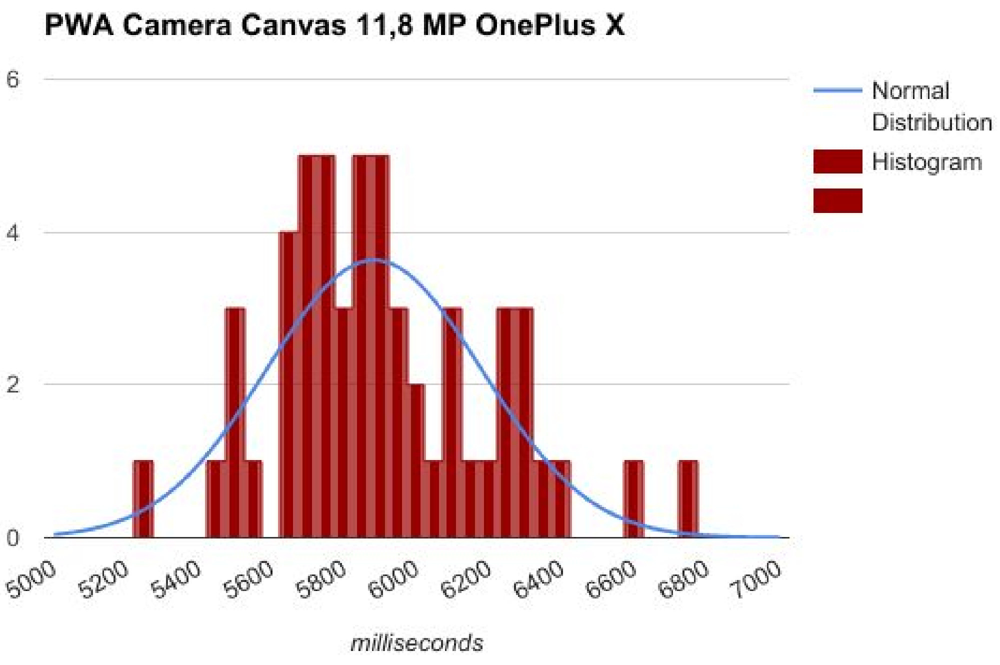

Figure 5.6: Normal distribution vs. Histogram of the PWA Camera Canvas 11.8 MP OnePlus X benchmarks

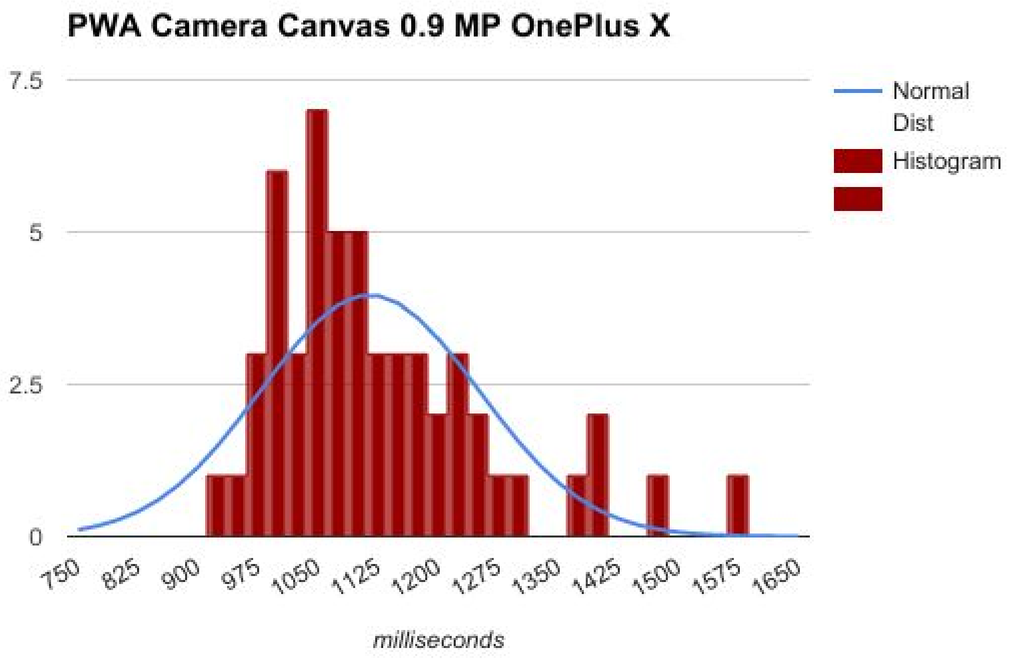

Figure 5.7: Normal distribution vs. Histogram of the PWA Camera Canvas 0.9 MP OnePlus X benchmarks

5.3.2 Geolocation

Table 5.2 shows the different samples from the geolocation benchmarks compared to each other. The NAA sample of each device is compared to the PWA samples, together with the p-value of a T-test and if the results were statistically significant.

Figure 5.8 to Figure 5.11 shows the normal distribution of the benchmarks, together with a histogram of the actual results to get a picture of how varied the results were. The figures shows that the normal distribution is followed approximately.

Sample 1	**Sample 2**	**p value**	**Statistically significant**
Android Geolocation Honor 8	PWA Geolocation Honor 8	< 0.0001	Yes
Android Geolocation OnePlus	PWA Geolocation Honor 8	< 0.0001	Yes

Table 5.2: P value of a T-test with the different samples of the geolocation benchmarks

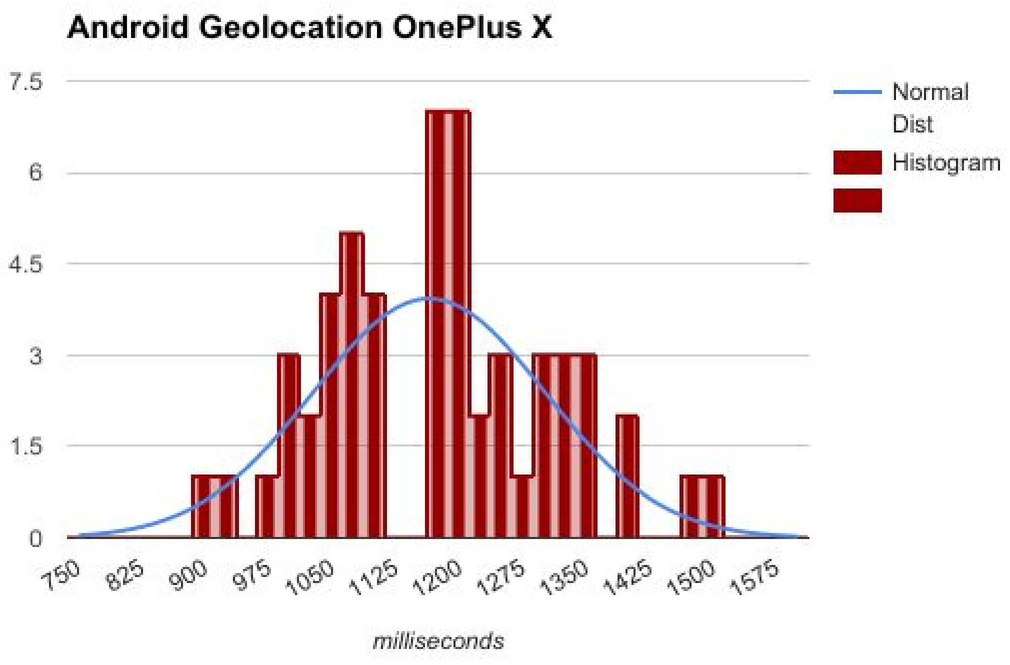

Figure 5.8: Normal distribution vs. Histogram of the Android Geolocation

OnePlus X benchmarks

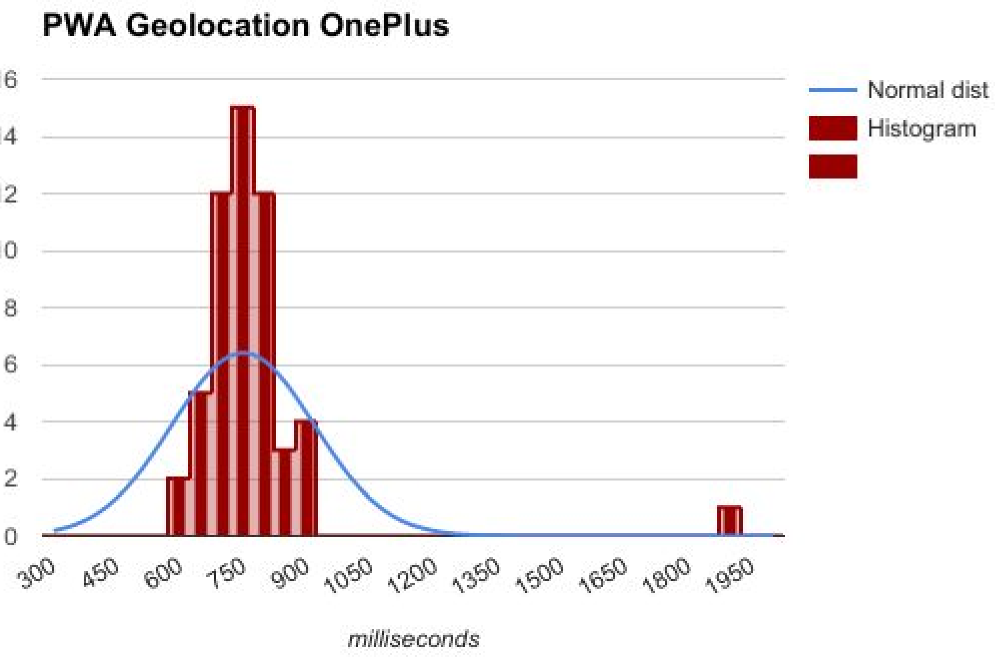

Figure 5.9: Normal distribution vs. Histogram of the PWA Geolocation OnePlus X benchmarks

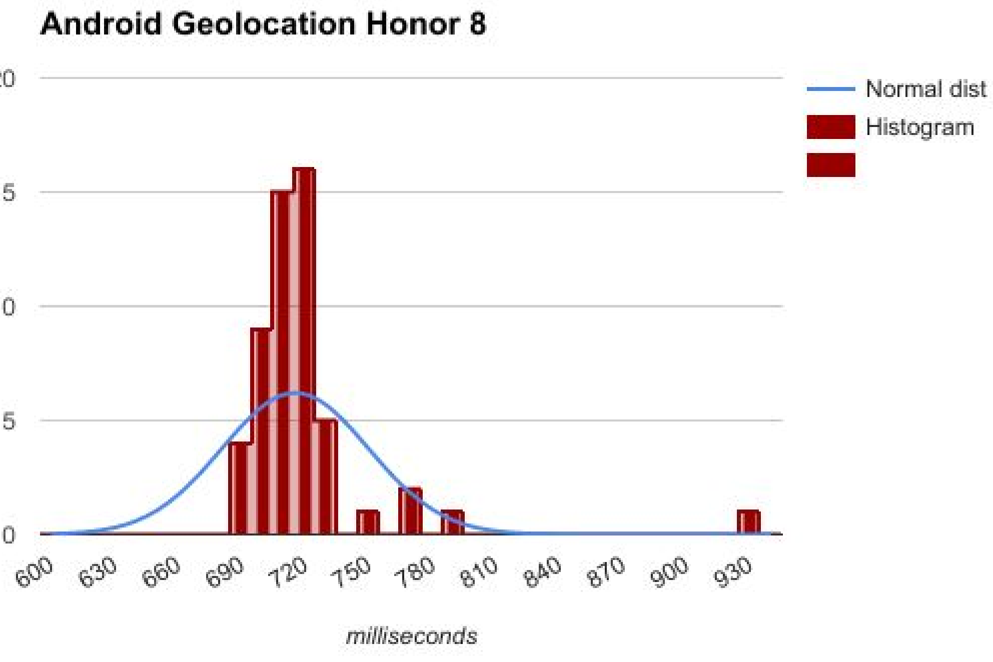

Figure 5.10: Normal distribution vs. Histogram of the Android Geolocation Honor 8 benchmarks

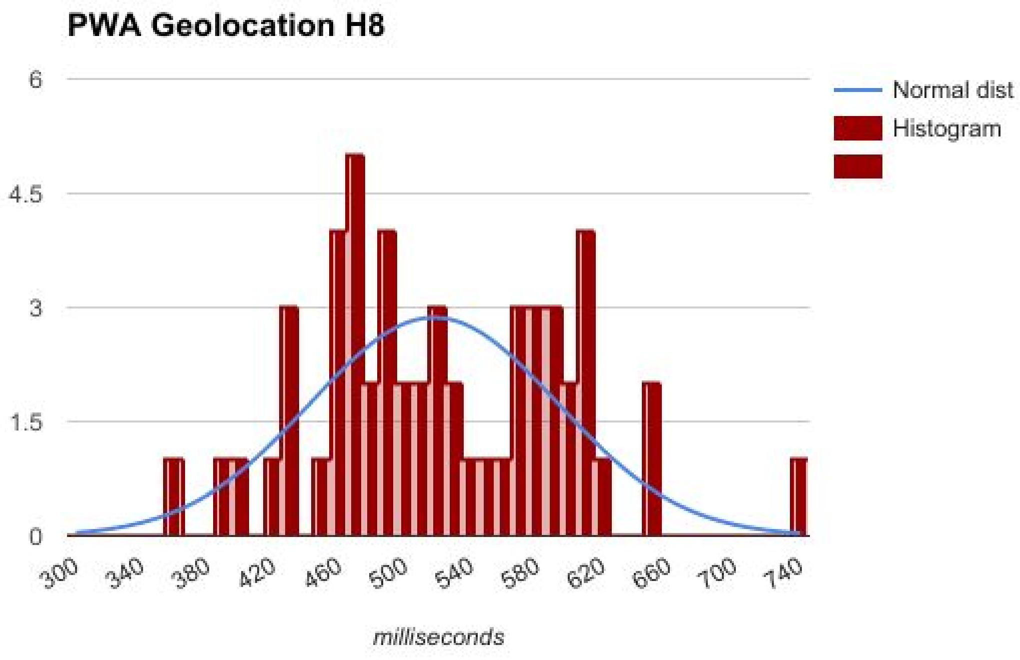

Figure 5.11: Normal distribution vs. Histogram of the PWA Geolocation Honor 8 benchmarks

5.4 Summary

The data gathered during the benchmarks shows that a Native Android Application still has better response time than a Progressive Web Application when it comes to accessing the camera and taking photos. However, the Image Capture API shows that PWA's are making progress, both in terms of speed and functionality. The difference between a NAA and a PWA using the Image Capture API is still significant, but the gap is slowly getting smaller. The data gathered with the Canvas method of a higher resolution is there to show a comparison in time, but cannot be used for a real comparison since the Canvas method does not provide a photo of a true 4K resolution. The Canvas method with the lower resolution shows that the PWA is faster than the NAA, but since the resolution is so much lower, this comparison can not be used to show that a PWA is faster than a NAA.

The benchmarks of the geolocation takes a different turn. Here, the PWA outshines the NAA in terms of response time when accessing the user's location and rendering a map. The PWA even shows a decrease in response time in the last runs.

6 Discussion

As shown in chapter 4.1, the first research question was answered. However, since technology is moving faster all the time, this answer could be falsified within months or years.

The second research question was not fully answered. There are still many parts that needs to be researched until a definite answer can be found, read more about it under the chapter 7.1 - Future Research. However, when it comes to performance, especially response time, of the camera and geolocation API's, the question has an answer.

As previous research shows, load times are important for the user. Techniques such as code splitting and isomorphism already cuts response times for page loads, both on the first and subsequent page views [17]. As we see in the results for the geolocation response time benchmarks, a PWA is already faster when it comes to figuring out the user's location and rendering it on the map. This makes PWA's an excellent choice for map based applications relying on geolocation.

Previous research mentioned Javascript frameworks as one solution to cross platform development[18]. This was not the optimal solution in the past, when web applications could not access the mobile devices Device API. However, this has now changed, and Javascript frameworks such as React paired with technologies for developing a PWA is now a better choice for cross platform development. For example, the new ImageCapture API, while still being an experimental feature, is a very interesting piece of technology, allowing the developer to have more control over the camera, and showing an increase in performance when it comes to response time. In the future, applications such as Snapchat and Instagram could be rewritten as PWA's.

The browsers API's for different parts of mobile devices hardware are improving all the time. Andre Charland and Brian Lerous discussed technologies such as PhoneGap, a framework for building Native Android Applications with web technologies. These technologies are used to bring the web application to a native context, allowing it to access everything a real NAA can do. In the future, these frameworks might be rendered obsolete, due to web API's for device access becoming better all the time.

7 Conclusion

The information that was gathered to answer RQ1 shows that there aren't any significant limitations with the Progressive Web Applications. There are a lot of hardware that the applications can access through the web and different ways to access them. For example it was found that there were two ways to take a picture with the progressive web application. There was the more standard way that took the picture and saved it with the help of an canvas. Then there was the newer that used an experimental feature in Chrome. This shows that the web accessing a device's hardware is evolving and there are a constant progress in this area.

The data gathered during the camera benchmarks shows that a Native Android Application is still faster than a Progressive Web Application on both the OnePlus X and Huawei Honor 8 devices. The canvas method of the PWA was faster than the NAA when taking pictures with a low resolution, but is not really comparable since users rarely take photos with a resolution of 720p on their mobile devices. When upping the resolution to 4K, the PWA canvas method was several seconds longer. Paired with the fact that the resolution of the photo was not really 4K, this solution is not viable. As of the date this report was written, making PWA's that make use of the camera hardware directly is not a viable option. However, the ImageCapture API in Google Chrome shows a bright future camera-based PWA's. A PWA using the ImageCapture API is still statistically significantly slower than a NAA, but the results shows that it is catching up in terms of response time. The ImageCapture API also allows more control of the camera, such as zooming, taking pictures with the full resolution of the mobile device's camera etc.

When it comes to the geolocation part of the benchmarks, the PWA was significantly faster than the NAA on both devices. This result shows that applications that relies heavily on maps could be developed as a PWA instead of a NAA, making the development process faster since two platforms can use the exactly same code.

7.1 Future Research

With the ImageCapture API catching up with the native android camera api, and geolocation already being faster in a Progressive Web Application than a Native Android Application, one can make the assumption that the web is catching up with native applications on mobile devices. However, it is impossible to answer if a PWA’s is on par with NAA’s. More research is needed on this topic. More API’s need to be tested, such as push messages, startup time of applications, file access etc. More parameters also need to be included in the benchmarks, such as memory profiling, cpu profiling, battery consumption etc. More extensive testing on different devices should also be made, since the Android eco system of mobile devices is a wide landscape. Applications that have a PWA counterpart to their NAA, such as Twitter’s native client and their Twitter Lite PWA could be used in user tests to see how users perceive the applications when they are compared to each other.

The ImageCapture API should be tested more when it is more mature. As of now, it is still hidden behind a flag in Google Chrome, not being accessible by a majority of users. In the future, this part alone could be a topic for research, comparing it against the native camera API of Android.

Including iOS in future research like this one could also be a good choice. Even though iOS does not support many of the features that make up a PWA (such as offline support and the ability to add it to the homescreen), one can still make comparisons between different device API’s such as the camera.

Many mobile users of the web have an iPhone, so this would be relevant research.

One could also compare the performance of a PWA with mobile applications built with tools that allow developers to use web technologies when developing native applications, such as Cordova or PhoneGap.

Appendix 1

Raw results from the benchmarks

1A Camera

Android Camera Honor 8		**Android Camera OnePlus X**		**PWA Camera ImageCapture Honor 8**	
Runs	**Milliseconds**	**Runs**	**Milliseconds**	**Runs**	**Milliseconds**
1	779	1	1008	1	1387.92
2	753	2	835	2	755.84
3	761	3	899	3	723.63
4	777	4	914	4	788.28
5	778	5	843	5	911.19
6	766	6	859	6	791.75
7	770	7	826	7	828.65
8	770	8	833	8	860.45
9	760	9	918	9	813.68
10	770	10	885	10	827.13
11	770	11	890	11	765.56
12	749	12	878	12	766.24
13	769	13	887	13	772.63
14	745	14	830	14	769.17

15	778	15	889	15	821.88
16	770	16	914	16	848.47
17	777	17	904	17	801.35
18	766	18	937	18	971.97
19	770	19	914	19	709.5
20	777	20	846	20	848.69
21	766	21	904	21	782.34
22	770	22	949	22	799.7
23	773	23	914	23	830.95
24	768	24	893	24	800.2
25	770	25	891	25	837.67
26	766	26	944	26	811.91
27	771	27	853	27	818.22
28	772	28	889	28	835.09
29	774	29	902	29	712.02
30	771	30	920	30	794.34
31	764	31	970	31	778.96
32	770	32	862	32	882.62
33	775	33	868	33	775.27
34	771	34	937	34	725.78
35	771	35	858	35	644.95
36	765	36	919	36	641.2
37	769	37	916	37	740.06
38	773	38	934	38	792.84
39	768	39	950	39	717.32
40	760	40	932	40	626.24
41	765	41	853	41	660.39
42	772	42	923	42	760.35
43	765	43	910	43	707.57

44	764	44	906	44	783.5
45	772	45	899	45	1051.25
46	765	46	871	46	862.63
47	758	47	921	47	927.55
48	762	48	868	48	1008.99
49	769	49	936	49	808.5
50	761	50	904	50	748.82
51	763	51	940	51	915.92
52	778	52	910	52	877.66
53	773	53	905	53	831.14
54	772	54	893	54	842.94

PWA Camera Canvas 11 MP Honor 8		**PWA Camera Canvas 0.9 MP Honor 8**		**PWA Camera Canvas 11 MP OnePlus X**	
Runs	**Milliseconds**	**Runs**	**Milliseconds**	**Runs**	**Milliseconds**
1	6,139.54	1	589.08	1	5,780.20
2	5,876.59	2	650.85	2	5,767.97
3	5,813.60	3	562.02	3	5,769.24
4	6,006.45	4	537.85	4	5,541.17
5	6,174.44	5	545.18	5	6,053.79
6	6,021.01	6	576.08	6	5,483.67
7	6,008.41	7	596.68	7	5,612.98
8	5,880.22	8	514.28	8	5,803.93
9	6,642.61	9	549.28	9	5,906.64
10	6,062.32	10	626.03	10	5,715.68
11	5,763.60	11	539.44	11	5,802.19
12	6,264.30	12	572.82	12	6,055.22
13	6,061.18	13	542.99	13	5,910.35
14	5,992.75	14	531.23	14	5,823.52
15	6,156.57	15	498.24	15	6,257.14
16	5,858.02	16	511.39	16	6,298.06
17	6,139.27	17	514.42	17	5,616.84

18	5,802.27	18	494.80	18	5,609.99
19	6,066.08	19	512.27	19	5,724.54
20	6,029.17	20	505.16	20	5,749.55
21	5,800.55	21	512.56	21	5,413.07
22	6,004.69	22	573.36	22	5,883.43
23	5,792.70	23	527.17	23	5,889.61
24	5,938.25	24	619.63	24	5,469.57
25	6,083.64	25	555.04	25	5,843.62
26	6,126.72	26	538.32	26	6,217.56
27	5,988.92	27	544.19	27	5,457.27
28	5,930.79	28	512.42	28	5,651.48
29	6,429.47	29	523.52	29	5,897.41
30	6,350.86	30	555.26	30	6,188.49
31	6,259.33	31	588.56	31	6,746.99
32	5,871.48	32	556.45	32	6,024.09
33	6,143.88	33	626.31	33	6,075.14
34	6,016.15	34	622.20	34	5,857.73
35	6,522.82	35	551.99	35	5,988.18
36	6,302.52	36	575.27	36	6,350.64
37	6,060.97	37	556.96	37	5,893.18
38	6,208.20	38	603.01	38	5,640.06
39	6,196.14	39	584.67	39	6,243.84
40	6,125.04	40	546.18	40	6,337.54
41	6,225.76	41	568.15	41	6,257.01
42	6,132.26	42	556.88	42	5,944.44
43	6,297.64	43	606.62	43	5,748.93
44	6,282.18	44	626.46	44	6,562.67
45	6,306.09	45	558.08	45	6,214.01
46	6,247.46	46	553.75	46	5,682.06
47	6,245.22	47	574.51	47	5,683.19
48	6,296.53	48	548.74	48	6,113.50
49	6,388.21	49	572.71	49	5,668.34
50	6,263.88	50	578.93	50	5,962.69
51	6,415.46	51	622.49	51	5,716.31
52	6,357.23	52	568.07	52	5,822.76

53	6,455.89	53	601.80	53	5,245.00
54	6,522.83	54	585.40	54	5,668.33

PWA Canvas 0.9 MP OnePlus X	
Runs	**Milliseconds**
1	**1062.514**
2	**1142.495**
3	**955.5**
4	**936.2**
5	**1042.975**
6	**995.9**
7	**920.8**
8	**1030.455**
9	**955.5**
10	**1058.155**
11	**1030.38**
12	**1135.55**
13	**994**
14	**979**
15	**968.85**
16	**1008.355**
17	**1049.71**
18	**1034.77**
19	**993.25**
20	**1076.545**
21	**1040.515**
22	**1068.275**
23	**1045.385**
24	**1010.07**
25	**993.5**
26	**1159.45**
27	**1193.415**
28	**1070.8**
29	**1080.845**

30	1106.39
31	1095.83
32	1174.445
33	984.95
34	1072.685
35	1107.78
36	1288.425
37	1352.335
38	1024.585
39	1108.425
40	1465.11
41	1226.94
42	1142.14
43	1207.12
44	1089.29
45	1165.78
46	1255.01
47	1216.75
48	1192.065
49	1080.53
50	1213.43
51	1568.585
52	1375.73
53	1388.445
54	1233.655

1B Geolocation

Android Geolocation Honor 8		Android Geolocation OnePlus		PWA Geolocation Honor 8	
Run	Milliseconds	Run	Milliseconds	Run	Milliseconds
1	923.40	1	1,231.90	1	730.33
2	682.24	2	909.07	2	421.45
3	707.73	3	982.74	3	567.35
4	686.79	4	1,048.06	4	581.81
5	694.47	5	977.09	5	642.19
6	691.80	6	875.55	6	453.21

7	694.05	7	950.70	7	600.34
8	701.61	8	1,045.21	8	586.82
9	690.92	9	1,062.76	9	574.25
10	682.62	10	1,011.02	10	457.51
11	706.57	11	1,010.10	11	522.91
12	705.21	12	1,081.86	12	465.05
13	699.10	13	997.12	13	551.64
14	745.41	14	1,171.74	14	503.58
15	697.05	15	1,058.35	15	467.05
16	703.88	16	1,075.73	16	467.45
17	711.05	17	1,034.37	17	484.20
18	718.51	18	1,074.74	18	489.83
19	707.26	19	1,046.12	19	563.86
20	783.59	20	1,088.47	20	454.11
21	708.60	21	1,183.06	21	543.22
22	722.53	22	1,054.37	22	517.77
23	707.95	23	1,094.79	23	536.56
24	717.12	24	1,158.07	24	520.63
25	714.45	25	1,183.68	25	397.04
26	723.96	26	1,054.04	26	598.95
27	704.21	27	1,180.59	27	356.97
28	710.32	28	1,153.66	28	646.37
29	699.08	29	1,155.59	29	574.20
30	717.05	30	1,334.42	30	469.19
31	714.34	31	1,209.71	31	514.41
32	711.73	32	1,248.86	32	565.25
33	688.61	33	1,152.17	33	605.84
34	717.21	34	1,162.30	34	503.45
35	695.77	35	1,181.99	35	573.29
36	711.38	36	1,232.97	36	492.21
37	714.20	37	1,181.23	37	475.04
38	704.46	38	1,173.71	38	424.06
39	702.14	39	1,331.19	39	601.51
40	700.86	40	1,310.84	40	440.31
41	706.80	41	1,299.95	41	611.75

42	**712.65**	**42**	**1,482.01**	**42**	**511.29**
43	**705.28**	**43**	**1,196.90**	**43**	**388.90**
44	**716.02**	**44**	**1,205.00**	**44**	**420.07**
45	**717.66**	**45**	**1,276.29**	**45**	**480.54**
46	**769.17**	**46**	**1,398.95**	**46**	**490.51**
47	**767.71**	**47**	**1,377.95**	**47**	**583.21**
48	**713.74**	**48**	**1,194.11**	**48**	**476.33**
49	**718.16**	**49**	**1,330.50**	**49**	**605.34**
50	**698.55**	**50**	**1,294.24**	**50**	**459.77**
51	**704.73**	**51**	**1,308.63**	**51**	**599.19**
52	**724.00**	**52**	**1,311.72**	**52**	**460.55**
53	**720.42**	**53**	**1,465.34**	**53**	**418.43**
54	**725.39**	**54**	**1,255.46**	**54**	**486.06**

PWA Geolocation OnePlus	
Run	**Milliseconds**
1	**1,856.68**
2	**891.19**
3	**786.78**
4	**795.25**
5	**793.45**
6	**610.45**
7	**662.02**
8	**762.20**
9	**809.85**
10	**712.40**
11	**715.73**
12	**778.03**
13	**709.40**
14	**743.11**
15	**858.67**
16	**723.66**
17	**821.11**
18	**734.11**

19	778.61
20	749.84
21	623.17
22	709.83
23	666.94
24	855.65
25	715.43
26	596.17
27	752.70
28	716.97
29	686.76
30	694.54
31	646.32
32	593.19
33	876.42
34	689.39
35	644.72
36	731.11
37	637.28
38	675.49
39	698.09
40	759.19
41	691.23
42	655.77
43	720.82
44	733.39
45	753.97
46	687.10
47	756.83
48	762.78
49	674.51
50	832.88
51	690.55
52	717.44
53	716.21

54	**751.18**

References

[1] D. Chaffey, "Mobile marketing statistics 2016," in *smartinsights.com*, Smart Insights, 2016. [Online]. Available: http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics
Accessed: Feb. 13, 2017.

[2] "Web | Google developers," in *developers.google.com*, Google Developers. [Online]. Available: https://developers.google.com/web/progressive-web-apps
Accessed: Feb. 13, 2017.

[3] M. Gaunt, "Service workers: An introduction," in *developers.google.com*, Google Developers, 2017. [Online]. Available: https://developers.google.com/web/fundamentals/getting-started/primers/service-workers
Accessed: Feb. 13, 2017.

[4] Ali Express Showcase," in *developers.google.com*. [Online]. Available: https://developers.google.com/web/showcase/2016/aliexpres
Accessed: Feb. 13, 2017.

[5] "Housing.com increases conversions and lowers bounce rate by 40% with new PWA," in *developers.google.com*, Google Developers, 2017. [Online]. Available: https://developers.google.com/web/showcase/2016/housing
Accessed: Feb. 13, 2017.

[6] E. Bidelman, "The basics of web workers," in *html5rocks.com*, HTML5 Rocks - A resource for open web HTML5 developers. [Online]. Available: https://www.html5rocks.com/en/tutorials/workers/basics
Accessed: Feb. 13, 2017.

[7] M. Rouse, "What is push notification? - definition from WhatIs.com," in *Search Mobile Computing*, SearchMobileComputing, 2015. [Online]. Available:
http://searchmobilecomputing.techtarget.com/definition/push-notification
Accessed: Feb. 13, 2017.

[8] "Apache Cordova," in *Apache Cordova*, 2012. [Online]. Available:
https://cordova.apache.org/
Accessed: Feb. 13, 2017.

[9] Nexercise, "Get started now," in *Phonegap*, 2016. [Online]. Available:
http://phonegap.com/
Accessed: Feb. 13, 2017.

[10] J. Parsons, "What are progressive web Apps?," in *Ionic*, The Official Ionic Blog, 2016. [Online]. Available:
http://blog.ionic.io/what-is-a-progressive-web-app/
Accessed: Feb. 27, 2017.

[11]"Service Worker Support," in *Can I Use*, 2016. [Online]. Available:
http://caniuse.com/#feat=serviceworkers
Accessed: Feb. 28, 2017.

[12]"The Status Of Service Workers In Microsoft Edge," in *https://developer.microsoft.com*. [Online]. Available:
https://developer.microsoft.com/en-us/microsoft-edge/platform/status/serviceworker/
Accessed: Feb. 28, 2017.

[13] "FiveYearPlanFall2015 – WebKit," in *https://trac.webkit.org*. [Online]. Available: https://trac.webkit.org/wiki/FiveYearPlanFall2015

Accessed: Feb. 28, 2017.

[14] M. Gaunt, "Push notifications on the open web," in *https://developers.google.com*, Google Developers, 2017. [Online]. Available: https://developers.google.com/web/updates/2015/03/push-notifications-on-the-open-web
Accessed: Feb. 28, 2017.

[15] P. Kinlan, "Installable web Apps with the web App manifest in chrome for Android," in *https://developers.google.com*, Google Developers, 2017. [Online]. Available: https://developers.google.com/web/updates/2014/11/Support-for-installable-web-apps-with-webapp-manifest-in-chrome-38-for-Android
Accessed: Feb. 28, 2017.

[16] “What Web Can Do Today,” *WhatWebCanDo*. [Online]. Available: https://whatwebcando.today/
Accessed: Mars 13, 2017

[17] R.Eneman, “Improving load time of SPAs - An evaluation of three performance techniques”, 2016

[18] Charkaoui, S., Adraoui, Z. and Benlahmar, E. (2014). “Cross-platform mobile development approaches”. 2014 Third IEEE International Colloquium in Information Science and Technology (CIST).

[19] Charland, A. and LeRoux, B. (2011). “Mobile Application Development: Web vs. Native”. Queue, 9(4), p.20.

[20] Jan Steczko. “Analysis of companies’ experience with cross-platform development compared to native development for mobile devices” in *Lnu.diva-portal.org*, Lnu Diva portal, 2016. [Online] Available:

http://lnu.diva-portal.org/smash/get/diva2:945760/FULLTEXT01.pdf
Accessed: Mars 13, 2017

[21] Mozilla Foundation. "Introduction to the DOM" in *https://developer.mozilla.org,* Developer Mozilla, 2017. [Online] Available: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction
Accessed: Mars 24, 2017

[22] Mozilla Foundation. "Window.performance" in *https://developer.mozilla.org,* Developer Mozilla, 2017. [Online] Available: https://developer.mozilla.org/en-US/docs/Web/API/Window/performance
Accessed: April 11, 2017

[23] Andrew Gazdecki. "2016 Is the Year Small Businesses Must Develop Mobile Apps" in *https://smallbiztrends.com*, Smallbiztrends, 2016. [Online] Available:
https://smallbiztrends.com/2016/03/2016-year-small-businesses-must-develop-mobile-apps.html
Accessed: April 25, 2017

[24] Stackoverflow. "Developer Survey Results 2016" in *http://stackoverflow.com,* Stackoverflow, 2016. [Online] Available: http://stackoverflow.com/insights/survey/2016
Accessed: April 24, 2017

[25] w3c. "MediaStream Image Capture" in *https://w3c.github.io*, w3c, 2016. [Online] Available: https://w3c.github.io/mediacapture-image/
Accessed: April 24, 2017

[26] Guava, "Google Core Libraries for Java" in *https://github.com,* Google 2017. [Online] Available: *https://github.com/google/guava*

Accessed: April 25, 2017

[27] “Twitter Lite PWA Significantly Increases Engagement and Reduces Data Usage
“ in *developers.google.com,* Google Developers, 2017. [Online]. Available: https://developers.google.com/web/showcase/2017/twitter. Accessed: May 20, 2017.

[EXEMPEL PÅ webplats/blogg] Författare. “namn på artiklen” in

Hemsida.com, Hemsida-namnet 2099. [Online] Available:

http://hemsida.com/saker/andra

Accessed: April 24, 2017