The International Journal of Multi-Disciplinary Research

ISSN: 3471-7102, ISBN: 978-9982-70-318-5

DESIGN AND DEVELOPMENT OF A FARMER'S SUPPORT PROGRESSIVE WEB APPLICATION (PWA) SYSTEM

(Paper ID: CFP/1253/2019)

Robbin Mchinzi,
Dept. of Information, Communication and Technology
School of Engineering,
Information and Communications University
Lusaka, Zambia,
robbinmchinzi@gmail.com

Mr. Lameck Nsama
Dept. of Information, Communication Technology
School of Engineering,
Information and Communications University
Lusaka, Zambia
Lamecknsama64@gmail.com

ABSTRACT

The use of technologies is found in every field from education to health, research, agriculture and many more fields. Most of the businesses have now adopted Internet as their business platform and the trend seems to be increasing. Thus, there is a need for developing fast, reliable, engaging and robust applications.

Native and Web are two major types of applications. Native apps are platform dependent for instance (Android, iOS and Windows) built using specific programming languages and Software development

Kit; on the other hand, web apps are platform independent. Websites which in many ways look and feel like native applications are run by a browser and typically written in HTML5. Native apps have full access to the device hardware while web apps still lack some of the access.

During the past few years, the use of native mobile applications has shown growth as compared to mobile web. Web apps are behind native apps in terms of performance, reliability and engagement. Businesses and developers often see the need to develop native mobile applications to overcome the limitations that the web as a platform imposes on mobile devices. When it comes to user's reachability, native apps are behind web apps. Native apps take more resources and time to develop, maintain and distribute applications for specific platforms. Also, native app publishing is a tedious process. In addition, to use the native app, users need to go through many steps which include signing up to the respective store, checking the memory, downloading, finalizing download by installing and finally opening to use it. A study revealed that, on average, an app loses about 20 % of its users for every step between the user's first contact and start to use. (Gabor Cselle, "Tales of Creation," 23 10 2012) Many users also find this process difficult and daunting.

I. INTRODUCTION

Progressive Web Application (PWA) is a catchphrase in the technology sector of the web. Many blogs are talking about it, and different companies like Ali Express (Ali Express, 2017), Twitter, and Housing.com are adopting this new type of web applications. PWA's gives the user a reliable, fast and engaging experience, much like that of native application. A PWA is a web application that is enhanced with some technologies that allow for native-like behavior 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, a JavaScript Worker that allows caching content for offline access and for push notifications. 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, 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. Service workers are as of now supported by Firefox, Opera and Chrome browsers, and both Edge and Safari have shown hints of supporting them. Web applications are often described as being cross platform. They are accessible from a multitude of 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 (Mehlhorn, Nils. 2016/2017). 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 has a platform imposes on mobile devices. In many cases, they have to develop their application for the web, iOS, and Android. Developers have used web technologies to develop cross-platform mobile applications with tools like Cordova (Google Chrome Developers. Opening Keynote (Progressive Web App Summit 2016). And Phone Gap (comScore Inc, 2016). 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 (Google Inc2016**,** Beverloo, Peter, et al, 2017).

Motivation and Significance of the study

Web development is moving forward at a tremendous pace all the time. Mobile access of the web has already surpassed desktop access (Mehlhorn, Nils. 2017) and making web applications that works seamlessly on mobile devices is more important than ever. Many companies are facing a challenge when developing applications (Osmani, Addy. 2017) 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 one application that worked on all these devices was developed? 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.

Scope and delimitations

This research will focus on design and development of an application that will try to close up a gap between a web and a mobile application by using a single codebase. To demonstrate an application farmer's support progressive web application will be designed and developed for providing weather and climate information so that farmers could take appropriate actions depending on the weather depicted by the application. The report will not focus on providing solutions to any other technological challenge's farmers in Zambia or any other part of the World are facing this is due to limited time.

Problem statement

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 (Google Inc. 2016, Beverloo, Peter, et al.2017). Web applications are somewhat limited today when compared to native mobile applications, but are moving forward all the time. We want to design and develop an application that can work as a web at the same time as a mobile app, and that application is a Progressive Web Application which can compete with a native application when it comes to performance, (Rebecca Fransson VT, 2017). When it comes to user's reachability, native apps are behind web apps. Native apps take more resources and time to develop, maintain and distribute for specific platforms. Also, native app publishing is a tedious process. In addition, to use the native app, users need to go through many steps which include signing up to the respective store, checking the memory, downloading, finalizing download by installing and finally opening to use it. A study revealed that, on average, an app loses about 20 % of its users for every step between the user's first contacts and start to use (Mehlhorn, Nils, 2017). Many users also find this process difficult and daunting. This is a huge disadvantage for both companies and developers. Both the web and native platforms have their own shortcomings, so there was a need for a platform which can combines the capabilities and experiences of native apps with the reach of web. Progressive Web App could simply be that platform.

Specific Objectives

The substantive specific objectives of this project were as follows;

1. To illustrate the characteristics and development of a Progressive web application (PWA)
2. To describe Native and web applications as we try to combine the beauty of both platforms.
3. To contrasted the results from objective number 2 with challenges from the field of mobile cross-platform development.

Research Questions

RQ1 what are the characteristics of a progressive web application?

RQ2 what are the beauties of a web and a mobile application?

RQ3 what are the challenges of web and a mobile application?

Review of the Literature

The literature on cross platform app development is broad. Leaving out papers in which cross-platform considerations are a side topic, there are several particularly notable categories of papers. First, papers present work on the creation or improvement of a specific framework (e.g. (Google Inc. 2016), (Henry, Alan.2015), (Russell, Alex.2016)). Typically, the frameworks represent one specific approach towards cross platform development or seek to advance the technological possibilities into a particular direction. Second, papers address one specific framework, often in a case study like fashion (e.g. (Savkin, Victor.2016), (Progress Software Corporation.2017)). Such work typically seeks to explore certain aspects of a framework or to assess its feasibility in predefined contexts. Third, papers combine work with several frameworks, usually including a comparison (e.g. (Lync, May.2016), (Google Inc.2017), (Progress Software Corporation.2017)), focusing on giving advice (e.g. Google Inc.2017), or looking at particularities (e.g. Narayanan, Anant.2013). These works can help with decision making and they may also contain such contributions to theory that enable an understanding of which approaches excel under which circumstances. Fourth, there are meta studies, which make use of the first two kinds of papers. Especially notable are studies that propose categorization and comparison frameworks. As part of such, exemplary comparisons have been carried out. Therefore, the fourth kind of papers provides the gateway to most other relevant work. A number of papers provide a comprehensive overview of cross-platform technology and how existing frameworks can be assessed. The most widely cited paper by Heitkotter ¨ et al. is already from 2013 (Verbruggen, Eddy.2017). The authors propose an evaluation framework. They have also exemplarily applied it to several development approaches, including Web apps and native apps as benchmarks. An extension of this original idea has been presented in 2016. (Google Chrome Developers.2016). This work does not only provide a more extensive catalogue of evaluation criteria but also tries to include weighted evaluation that cater for novel device categories such as wearables. Several authors have conducted work that takes similar aims (Google Inc.2017), (VideoSpike LLC. 2017), (Lee, Andrew.2013), (Ross, David, Shepherd, Eric and Mills, Chris.2016), (Google Inc.2017), (Firbase Inc.2017). All of these papers support a deepened understanding of cross-platform app development as a whole.

Related Works

A lack of literature covering Progressive Web Apps has already been identified by a recent position paper (Popescu, Andrei.2016), which brought the concept of Progressive Web Apps to the academic communities. The paper gives a holistic introduction to PWAs, as well as some initial research findings and thoughts. Research was based on three apps: one hybrid app, an interpreted app, and a PWA. Some initial performance measurements were conducted and the generated app sizes were compared. This paper additionally presented a list of possible further research areas and topics to be explored. While this paper can be seen as a kind of shorter predecessor to our article, it must be ascertained that to a large extent the current body of knowledge is made up of literature published by developers, practitioners, and the industry in general. Practitioners and the industry continue to put efforts in implementing PWA characteristics into their Web sites, as discussed in detail in Section 3.4. Little progress within academia can so far be recorded. The academic contributions identified are few in number. Malavolta et al. (Knight, Robert, Mattisson, Jonas and Blackburn, Nathaniel.2017) makes an interesting contribution discussing energy efficiency of Progressive Web Apps, and the energy impact of Service Workers. Their research includes measuring energy consumption using different devices and scenarios. Except from the previously discussed literature, little else has been identified directly on the topic of PWAs within academia. Outside of academia, in the Fas paced world of JavaScript and the (mobile) Web, articles and discussions on PWA proliferate. As discussed in Section 3.4, the Google I/O 2017 conference featured seven PWA-related talks, some highly technical, some rather business-related. In 2016, Google also hosted the first Progressive Web App Developers Summit (Progress Software Corporation.2017), as an effort to further advocate the concept to developers and the industry. While practitioners' enthusiasm must of cause be weighted carefully as Google seems to rally for PWAs, the spread of interest is nonetheless remarkable. Multiple books on PWAs either have been published or are in the process of being written, e.g. Hume's early-access book, a technical step-by-step textbook for building PWAs. Published on O'Reilly also as an early-access book, Ater Tal aims to teach various technical aspects of PWAs from a mobile native point of view, suggesting he is Bringing the Power of Native to the Browser. The literature situation suggests numerous research possibilities, as discussed in Section 4.2. Quite notable is the lack of discussion of PWA development in the realm of iOS, owed to the current lack of support. While missing support by Apple might be a major hindrance for the spread of PWAs, practitioners began to question whether Apple can retain its position (Tsonev, Nikolay and Vakrilov, Alexander.2017). The adoption of new standards in browsers, especially the specification for service workers, allowed for a further evolution of web development (Russel, Alex, Song, Jungkee and Archibald, Jake.2017). In 2015, the Google Chrome engineer Alex Russell took a closer look at these trending characteristics and labeled the corresponding web applications as progressive. According to Alex Russell, this new class of applications delivers an even better user experience than traditional web apps are able to" (Russell, Alex.2017). This improved user experience it is achieved by the usage of capabilities which ordinary web applications lack. Compared to the mobile web, users spent a considerably high amount of time on mobile apps (see Figure 1). The engineers behind Progressive web applications substantiate this fact with the inferior capabilities of web applications on mobile. Without these, they would just not be able to be as engaging as native mobile apps were (Google Chrome Developers. Sumit.2016). after the native app is installed, it enjoys several benefits over its web-based complement. An icon on the user's home screen and push notifications allow them to reconnect with users in convenient ways. So far, web applications would not have these chances for making the user come back (Ater, Tal. pp. 12-13).

II. METHODOLOGY

Rapid Application Development (RAD) methodology was used in the designing and development of this system. Software design is the process by which an agent creates a specification of a software artefact, intended to accomplish goals, using a set of primitive components and subject to constraints. It refers to either all activity involved in conceptualizing, framing, implementing, commissioning and ultimately modifying complex systems or the activity following requirements specification and before programming. In RAD model the components or functions are developed in parallel as if they were mini projects. The developments are time boxed, delivered and then assembled into a working prototype. Advantages of the RAD method:

Integration from very beginning solves a lot of integration issues;

1. Reduced development time.
2. Quick initial reviews occur
3. Increases reusability of components
4. Encourages customer feedback

Disadvantages of RAD method:

Only system that can be modularized can be built using RAD

1. Depends on strong team and individual performances for identifying business requirements
2. Requires highly skilled developers/designers.
3. Inapplicable to cheaper projects as cost of modeling and automated code generation is very high
4. High dependency on mode*ll*ing skills

DESIGN

The design was implemented by using the technologies and languages listed below:

i. PHP this is a general-purpose scripting language that is suitable for server-side web Development. PHP generally runs on a web server. The PHP code is embedded into the HTML source document. Any PHP code in a requested file is executed by the PHP runtime, usually to create dynamic web page content. It can also be used for command-line scripting and client-side GUI applications. PHP can be deployed on many web servers and operating systems, and can be used with many relational database management systems (RDBMS) and it is available for free - (Mwape, 2017).

ii. HTML5 HTML stands for hypertext mark-up language, and the hypertext refers to the fact that HTML makes it so that you can click on links in web pages. That's the hypertext. The words mark-up language just means that it is something that you use to mark-up normal English to indicate things. Each page contains a series of connections to other pages called hyperlinks. Every web page you see on the Internet is written using one version of HTML code or another. HTML code ensures the proper formatting of text and images so that your Internet browser may display them as they are intended to look. Without HTML, a browser would not know how to display text as elements or load images or other elements. Hypertext Mark-up Language was first developed by Tim Berners-Lee in 1990

iii.. iii. MySQL MySQL is a freely available open source Relational Database Management System (RDBMS) that uses Structured Query Language (SQL). SQL is the most popular language for adding, accessing and managing content in a database. It is most noted for its quick processing, proven reliability, ease and flexibility of use. MySQL is an essential part of almost every open source PHP application. Good examples for PHP & MySQL-based scripts are WordPress, Joomla, Magento and Drupal. One of the most important things about using MySQL is to have a MySQL specialized host.

iv. JAVA SCRIPT JavaScript is considered to be one of the most famous scripting languages of all time. JavaScript by definition is a Scripting Language for the World Wide Web. The main usage of JavaScript is to add various Web functionalities, Web form validations, browser detections, creation of cookies and so on. JavaScript is one of the most popular scripting languages and that is why it is supported by almost all web browsers available today like Firefox, Brave or Google Chrome.

Progressiveness (Characteristic: Progressive)

For native implementations the barrier of standardized APIs is non-existent and therefore any device capability provided may be used in an application. Yet, this also leaves the problem of handling different feature sets to the application code itself. Backwards compatibility has to be ensured so that the target audience is not limited to a certain system version or specific device. On the Android platform, this aspect is generally covered by the usage of designated support libraries. These allow for backward-compatible implementations of import, core platform features" (Google Inc.2017) and may be used to create more modern app interfaces on earlier devices" (Google Inc.2017). Ideally, this ought to be leveraged by a NativeScript application. Alternatively, different approaches for providing progressive behavior may be applied.

Progressive web application
Progressive Web Application is not a new technology or framework rather it is best practices which have been adopted by the web to give a native app like feeling to the user. Components such as Service worker, App Shell, Web App Manifest and Push Notification, which are discussed in the following section, work together to give native like feeling to PWA. It can be installed on the user's home screen with one tap. A user can enjoy full-screen experience without going through the hassle of the downloading process. It loads faster even on a flaky network and works offline. It gradually develops with inter-action and sends relevant push notification thus

Increasing user engagement
The popularity of PWA is growing at a very fast pace in the field of e-commerce, business, online news portal and other fields because of its peculiar characteristics listed below as follows:

Progressive
PWA works for all user on all browser and builds up continuously; taking the benefits of features found in user's device and browser.
Responsive

PWA's UI fit on all devices forms, factor and size: mobile, desktop and tablet. Responsive feature is achieved using the material design, fluid grid concepts, CSS3 media queries and flexible images. (Google Inc.2016).

Home screen Shortcut
Due to W3C web app manifest and service worker registration scope, search engines identify PWA as an application. It also increases the probability to be found easily on search engines compared to the native app.

Native App-like
Implementation of design concept App-shell architecture makes PWA unique and divide application functionality from its content with least possible page refreshes to give native app look. PWA, when launched from the home screen, has an entirely native app like look with a splash screen.

Connectivity-independent
Implementation of service worker makes it able to work offline and give good performance even on a flaky network. PWA does not treat loss of connectivity as an error, but as an eventuality, which can be planned for, and handled with grace.

Fresh
New content published gets an update once the user is connected to the Internet due to the service worker update process. Application shell and content, after it is cached al-ways load from the local storage.

Safe
Implementation of HTTPS connection and SSL certificate to serve the page is a must to prevent man-in-the-middle attacks, password intruding and making sure content is not manipulated.

Push Notifications
Push API and Notifications API make the user more likely to revisit PWA by the user using push notifications. PWA can receive messages pushed from the server, which can be shown as a notification, and the user is notified about the updates. This helps for reengagement and bringing the user back to the application. Progressive Web Apps' notifications have a completely native feel and are like those of native app notifications. (Lync, May.2016)
PWAs have low friction as they are a product of the web, i.e. faster and cheaper to develop lowering the user acquisition costs and always up to date. However, PWA faces some challenges. It

has issue with cross browser support. Google Chrome, Chrome for Android, Samsung Internet has good support for PWA while Firefox and Safari still lack good support. PWAs also have limited functionality as they cannot access all device-specific hardware. They can only support hardware that is supported by HTML5. (Ater, Tal. 2016). It also lacks the cross-application login support.

Native apps

Native apps are platform dependent (Android, iOS, Windows) built using the specific programming languages, SDK. Developers should stick with Java for Android, C# for windows and swift for iOS. Native apps have full access and take full advantage of device features such as access to the accelerometer, camera, compass, GPS, incorporate gestures and APIs, thus provide great UI, UX, performance and reliability. Native application is installed on the user's device via specific app stores (Apple's app store or Google play store).

Native apps are platform dependent thus cannot be deployed on multiple platforms. The cost is high and development time is longer. A user must install the application via respective store on device to use thus set drawbacks if the user wants to use the application just one single time or periodically. Also, updating native app is tedious.

Hybrid apps

Hybrid apps are platform independent built using web technologies HTML5, CSS 3 and JavaScript wrapped inside a native container Cordova. Once the app is developed, it can be deployed on multiple platforms. Hybrid apps have access to most device features like native but when it comes to 3D, fluid animations, multi-touch, graphics, transition and gaming, their performance decreases. However, the cost and time of development is lower than native. Hybrid apps should also be installed on the device and need to be updated from time to time Hybrid apps are deployed in a native container that uses a mobile Web View object. When the app is used, this object displays web content thanks to the use of web technologies (CSS, JavaScript, HTML, HTML5). It is in fact displaying web pages from a desktop website that are adapted to a WebView display. The web content can either be displayed as soon as the app is opened or for certain parts of the app only i.e. for the purchase funnel. In order to access the devices' hardware features (accelerometer, camera, contacts) for which the native apps are installed, it is possible to include native elements of each platform's user interfaces (iOS, Android): native code will be used to access the specific features in order to create a seamless user experience. Hybrid apps can also rely on platforms that offer JavaScript APIs if those functionalities are called within a Web View.

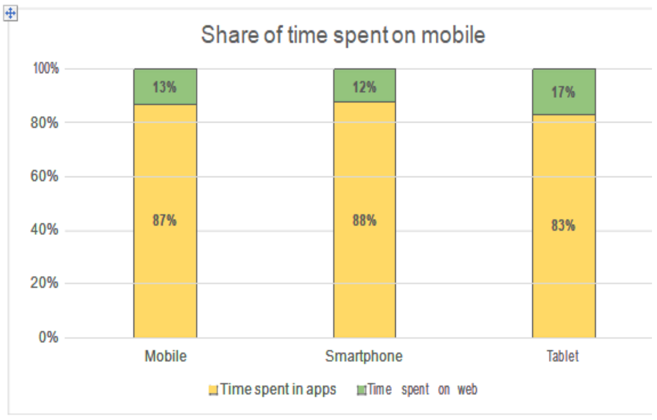


Figure 1 Share of time spent on mobile apps and web, data source: 9 p. 12

PWAs are meant to provide remedy. In the right environment they can perform in an app-like fashion. They are able to send out push notifications, launch in hardware-accelerated full-screen and use a wide range of sensors. Various innovative interfaces like the Push API or Geolocation API are making this possible (Beverloo, Peter, et al. 2017) (Popescu, Andrei. 2016). Hereby, features which were previously reserved for native applications are made accessible in a standardized way. In concept, PWAs are meant to be on a par with native apps regarding their capabilities (see Figure 2).

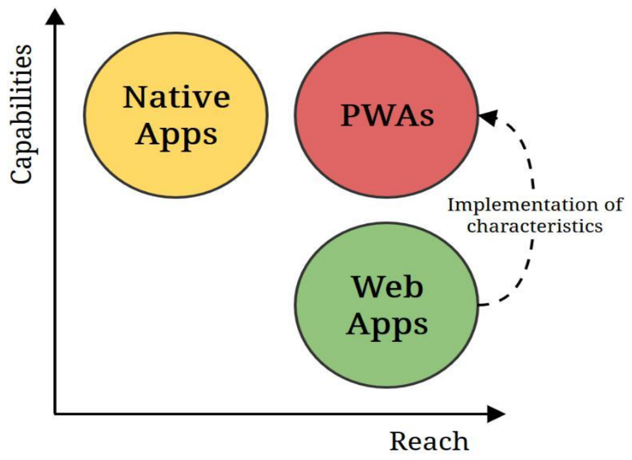


Figure 2 Envisaged classifications of PWAs regarding c apability and reach (Google Chrome Developers. 2016)

The average monthly audiences of mobile web properties are about three times the size of the ones of comparable mobile apps. In addition, the mobile web audiences grow with twice the speed of their counterpart (comScore Inc. p. 15). This means that web apps have a major advantage regarding reach on mobile systems. A PWA is meant to leverage this fact by being just that eventually: a web application.
"Establishing app audiences is harder, but their real value is in their loyalty." (comScore Inc. p. 19)
Although the web predominates in terms of audience size, the engagement of their audiences needs to be considered, too. As already stated, the time spent on apps is overtaking the mobile web by far. Users show far more engagement for them. Yet, this engagement is mostly limited to a handful of apps (comScore Inc. p. 30). PWAs are supposed to combine the loyalty for native applications with the reachability of the web. Several approaches in the field of cross-platform development stem from the incentive for building mobile applications with web technologies. Solutions like Apache Cordova bundles web resources into a native application wrapper. This workaround allows for building applications with the means of web development while leveraging native features (Mehlhorn, Nils. pp. 19-21). In theory, with PWAs it should no longer be necessary to deliver a web application in any kind of proprietary native wrapper. Implementing the defined technical requirements makes "good old web sites exhibit super-powers" (Mills, Chris. 2016). Thus, the mobile browser itself takes care of filling the gap to the system. The result is a standardized solution for building web applications which may feel like native apps.

Characteristics

It becomes clearer which role PWAs are meant to play in the mobile landscape when looking at their characteristics one by one. Originally named by Russell, a PWA should be all of the following:
Table 1

Responsive	Fresh	Re-engage able
Installable	Safe	Discoverable
Linkable	Connectivity	App-like
	independent	interactions

These are the dictated characteristics a web application needs to have in order to certify as progressive. Every characteristic is represented by certain technical properties. A subset of these properties forms a baseline for web applications to be detected as a PWA by a browser (Google Inc. 2017) (Russell, Alex. 2016). Google also provides a way for asserting many of the baseline properties automatically with a tool called Lighthouse (Google Inc.Lighthouse.2017). For the following elaboration the characteristics are organized according to the definitions on the Mozilla Developer Network as these arguably make up for a more suitable separation in this case (Mills, Chris.2016). The associated classifications by Russel are listed accordingly.

Network independent (Fresh, Connectivity independent)

A central component of a PWA is the service worker. It represents a previously mostly non-existent instance between a web page and the corresponding server. The service worker is defined and registered via JavaScript for a single or multiple page. After its registration it listens to events broadcasted by the web page (Ater, Tal. p. 15-17). With this new instance in place, the offline state for web applications is meant to be improved. Instead of ending up with no functionality at all in a situation with no network or low transmission rate, a reasonable offline experience can be delivered. The service worker is able to offer cached assets and data independent of network availability. In the best case the application can allow for browsing previously visited pages while displaying cached content. The bare minimum for the characteristic of being network independent would be displaying a custom offline page (Russell, Alex. 2016).

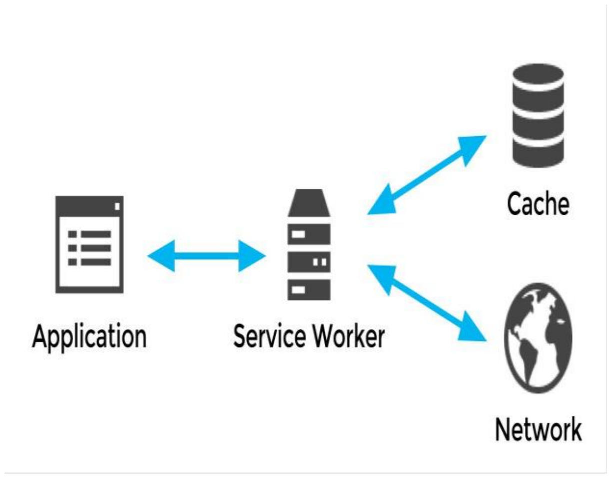


Figure 3 UML activity diagram for the workflow of an offline capable app implemented by service work ers (Shepherd, Eric, Mills, Chris and Sabiwara. 2016)

As service workers guaranty a response for requests made by the application, be it populated with cached data, they need to intercept HTTP communications for being able to alter or replace their contents. To allow for this process to be carried out in a secure environment, web pages registering a service worker have to be served via secure HTTP (HTTPS). Otherwise man in the middle attacks can take place (Ater, Tal. 27-28).

Safe

So PWAs have to be safe, thus use HTTPS, for service workers to work. But there are more reasons justifying a standalone characteristic of safeness. HTTPS is based on the successor of the cryptographic specification for the Secure Sockets Layer (SSL), called Transport Layer Security (TLS). It prevents tampering of web communications. Without the protocol in place, intruders may be able to access sensitive information or exploit the connection to insert advertisements, for example. But it is not just useful to secure sensitive connections with HTTPS. Even web pages which seem irrelevant from a security perspective can be violated for gathering usage data illegitimately (Basques, Kayce. 2017). Implementing HTTPS even favors how a web page is ranked in search engines (Bahajji, Zineb Ait and Illyes, Gary. 2014). Moreover, some web technologies may work even better on HTTPS connections (Ater, Tal.p. 28).

Discoverable

Unlike native mobile applications, web applications do not have central points like app stores or market places for being discovered. Instead they may be discoverable via search engines or social media links. This plays into the aspect of reach. A central place for app distribution is limited in the number of apps it can represent efficiently and thus a new application "can seem like a grain of sand on a beach" *(Hume, Dean Alan. p. 6).*

> *"These apps aren't packaged and deployed through stores; they are just websites that took all the right vitamins."* (Russell, Alex. 2015)

However, PWAs may be market in a way more dynamic way. Any platform complying with the standards is able to handle them. A fundamental artifact for this characteristic is the web app manifest. This file contains metadata in the JavaScript object notation (JSON) format with essential information about the application. Assets like app icon or splash screens are defined and identification data like the application name or author is provided. Moreover, an entry point for the application is to be specified in the manifest. The manifest compares very well to something like the application manifest for native Android applications. Just like it describes how the Android system should handle a packaged application, the web app manifest describes the PWA in a way any modern browser may be able to understand (Hume, Dean Alan**.** pp. 3,6) (Caceres, Marcos, et al. 2017) (Knight, Robert, Mattisson, Jonas and Blackburn, Nathaniel. 2017) (Google Inc.2017).

Installable (App-like-interactions)

The web app manifest is also required for making the web application installable. In this context installable refers to being able to add an icon to the user's home screen. This way the PWA can be started in a similar manner to native mobile applications. For this to work within Google Chrome on an Android system, a couple of requirements have to be met. Firstly, the manifest has to provide basic information about the app. Its name, a shortened name, an icon image and the already mentioned entry point in form of a Uniform Resource Locator (URL) are required. Furthermore, an appropriate display mode for a screen filling presentation has to be specified. Secondly, the application has to be able to start while offline. As already stated, this is to be ensured via service workers. Lastly, implicated by the service worker, the web site has to be served over HTTPS for being installable. Eventually, if the requirements are met and a certain degree of engagement is determined, the browser prompts for adding the PWA to the home screen (Russell, Alex.2016) (Gaunt, Matt and Kinlan, Paul.2016). A PWA launched from the home screen should execute in an application shell providing a fast startup. The shell may consist of static user interface elements which are cached during the installation. This guaranties a network independent provision of the application infrastructure which can then be filled with content (see Figure 4). With this separation of infrastructure from content the perceived performance and thus the user experience are enhanced (Hume, Dean Alan. pp. 18-23.2016) (Osmani, Addy.2017). Installable basically provides users the ability to add a home screen icon on their mobile device. There's no software or files to download on your mobile device (or desktop/laptop for that matter). With a native mobile-app, once you've gone thru the App Store download process, the app's icon will show up on your mobile device's home screen (See image below).

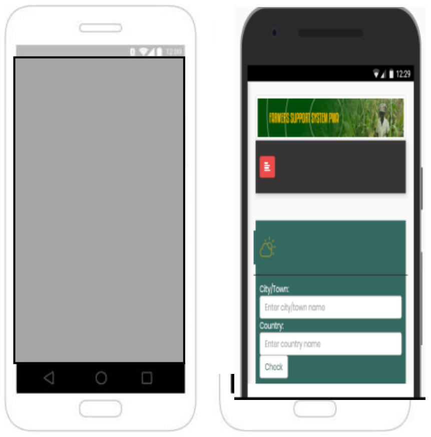


Figure 4 Illustration of a PWA's applic ation shell without content (left) and filled with content (right)

Linkable

Having a discoverable PWA means it is approachable in terms of technical handling. It allows for accessing the app similar to how a native mobile app would be accessed. In contrast, the fact that PWAs are linkable refers to approachability from the web perspective. To use a specific feature of an ordinary mobile app it is required to be installed first. And even then, the ways for accessing the feature might be limited. Web applications can be accessed almost at any point without preliminary work by just having the right URL. The high reachability of the web thrives from this simplicity. A high amount of traffic originates from assignable sources resulting in a phenomenon called "dark social" (Madrigal, Alexis C. 2012). While its extent might be controversial, it shows how there may be access channels to an application which might not be seen in advance. PWAs leverage this aspect by working without installation and therefore being easily shareable (Russell, Alex.2015). Having a web-app be Linkable is simple and makes it considerably easier for users to access web-app compared to getting a mobile native app on an App Store for Customers to access the app, they simply click on the link. This makes it very convenient for you to share a web-app by adding the web-app's link in an email, a tweet, add to blog, etc. Then simply click the link, a browser opens and there's your web-app.

Re-engage able (App-like-interactions)

As pointed out, the engagement for web applications is rather low. PWAs are meant to overcome this. With an icon sitting on the home screen and push notifications they are supposed to re-engage the user just like native apps can. Google's model example for this characteristic is the e-commerce site Flipkart. When their web presence was launched as a PWA, the time that users spent on the site tripled. More than half of their users now visited the site via the home screen icon. Among them the conversion rate would be 70% higher compared to the average user (Google Inc.2017). Flipkart substantiates these numbers with the use of the new technologies. Based on the Push API they were able to send messages to their clients' service workers, which in turn would then notify the user via the Notifications API. As the service workers "live beyond the lifetime of the browser" (Nagaram, Amar.2017), such ways of interacting with the user would now be possible for web applications (Nagaram, Amar.2017).

Responsive

"PWAs are quickly becoming a set of best practices. The steps you take to build a Progressive Web app will benefit anyone who visits your website, regardless of what device they choose to use." (Hume, Dean Alan. p. 4)

One of the best practices brought together by PWAs is the one of responsive design. Mobile-first approaches changed the way websites are designed substantially, and one might say rightfully so. Today, two thirds of the time spent on digital media takes place on mobile systems (comScore Inc. p. 6). This grants valid reason for making an effort to deliver highly mobile friendly websites. With a heterogeneous landscape regarding device specifications, the goal has to be a proper display regardless of form factor. This is commonly achieved by using media queries and advanced features of CSS like device adaption and the flexible box layout (Ross, David, Shepherd, Eric and Mills, Chris.2016).

Progressive

Just like a PWA should display properly on any device regardless of its form factor or design, it should also work properly regardless of the browser in use. Not everyone is able to keep pace with the mentioned emergence of advanced interfaces. For example, service workers are not yet overall supported by common browser vendors (Archibald, Jake.2017). The established web development principle of progressive enhancement is used to deal with this circumstance. Similar to mobile-first, it relates to building a web application by starting with a small but working foundation and then layering more functionality on top. The usage of advanced technologies (e.g. low-level APIs such as the ones for push notifications) would be acceptable as long as some kind of fallback is provided. Thereby, the website is made "more accessible to all audiences".

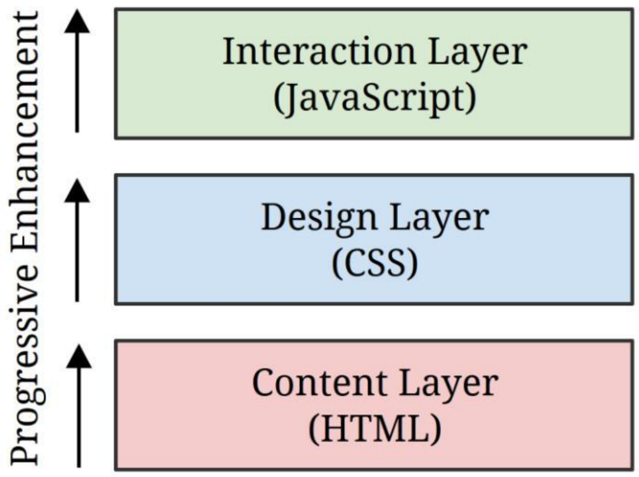


Figure 5 Illustration of layered view o n progressive enhancement

Source: (4 p. 27)

The principle is often described as being layered. The base layer would be made up by the site's content with subsequent layers for design and interaction. This way, even clients that are not supporting JavaScript or CSS may get access to a site's content. The metaphor can also be applied within these layers where different levels of support may be present (see Figure 5). For example, some browsers may be proficient in JavaScript but may not implement the most recent standards

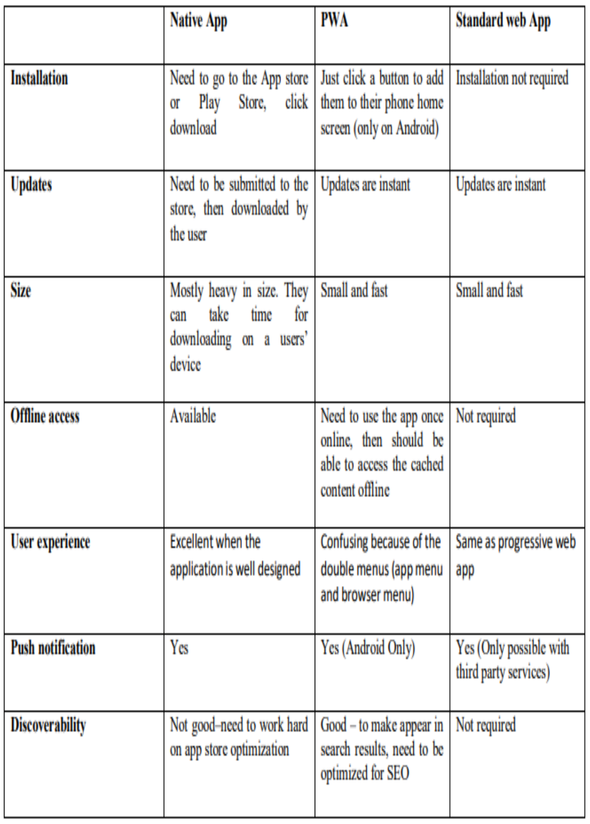
(Ater, Tal. p. 27). With progressive enhancement these clients will still be able to deliver less yet reasonable functionality. Regardless of a client's current connectivity, the website may not be completely broken.

Hereinafter, all characteristics of PWAs are listed collectively with a certain described intent and the technical properties used to instantiate them. The characteristics Linkable and Progressive lack specific technical properties as their implementation are essential to web development itself. With them, it is rather about the definition of their concept than the emergence of a new technology.

Web Performance

Web performance can be divided into two parts. The performance on the server-side is commonly referred to as the backend performance and the client-side performance is referred to as the front-end performance. The client in this case means the web browser that is used to access a webpage. The client-side performance can be further divided into the performance of a webpage on mobile devices and on desktop computers. This thesis focuses on the client-side performance and optimizing it on mobile devices and mobile network connection speeds in order to achieve fast user experiences in varying conditions. On a fully responsive website, the content will automatically adapt to the device screen size and no separate mobile web page is usually needed. The technologies and the content are similar when accessed using different devices, such as mobile phones or desktop computers. However, the performance of a webpage on different devices may vary. These differences in the performance are caused by other factors according to the study; mobile web reach is way higher than native app reach. It was 11.4 million unique visitors per month compared to 4 million visitors. Whereas the states of user engagement with services, showed that users tend to spend more time on native mobile apps compared to the standard web app. It was an average of 188.6 minutes on app against 9.3 minutes on the web. So, the idea was clear. They wanted to provide a native app like engaging experience to users on the mobile web. In this way, Progressive Web Apps were developed to deliver amazing user experience on the web. Below is the comparison between Native App, PWA and Standard web app on various important parameters:

Table 2

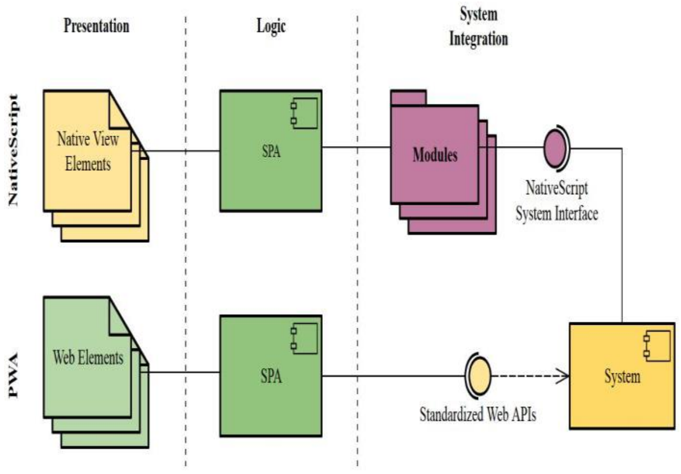


	Native App	PWA	Standard web App
Installation	Need to go to the App store or Play Store, click download	Just click a button to add them to their phone home screen (only on Android)	Installation not required
Updates	Need to be submitted to the store, then downloaded by the user	Updates are instant	Updates are instant
Size	Mostly heavy in size. They can take time for downloading on a users' device	Small and fast	Small and fast
Offline access	Available	Need to use the app once online, then should be able to access the cached content offline	Not required
User experience	Excellent when the application is well designed	Confusing because of the double menus (app menu and browser menu)	Same as progressive web app
Push notification	Yes	Yes (Android Only)	Yes (Only possible with third party services)
Discoverability	Not good–need to work hard on app store optimization	Good – to make appear in search results, need to be optimized for SEO	Not required

Source: (4 p. 27)

Context Diagram

The Context Diagram shows the system under consideration as a single high-level process and then shows the relationship that the system has with other external entities (systems, organizational groups, external data stores, etc.). The following are the context diagrams for PWA and Native apps



(Progress Software Corporation. 2016). Figure 6:

UML component diagram illustrating differences and similarities between progressive web and Nati ve Script application architecture layers

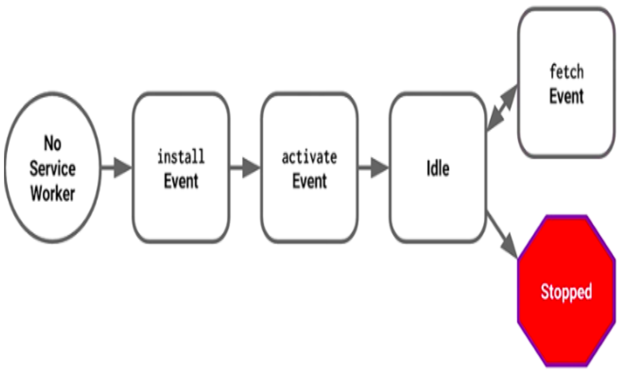


Figure 7:
UML activity diagram for the workflow of an offl ine capable app

3.2.3 System Software Level Architectural Design

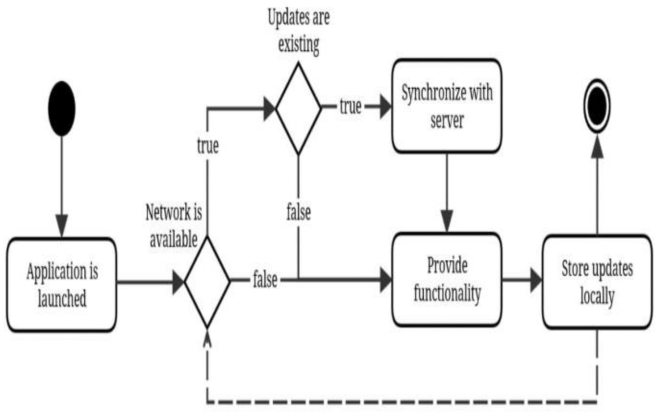


Figure 7. Source: (4 pp. 27-28)
Figure 7 UML activity diagram System Softwa re Level Architectural Design
It is based on diagrammatic representations of software components. As the old proverb say s “a picture is worth a thousand words” By u sing visual representations, we are able to bet ter understand possible flaws or errors in soft ware or business processes.

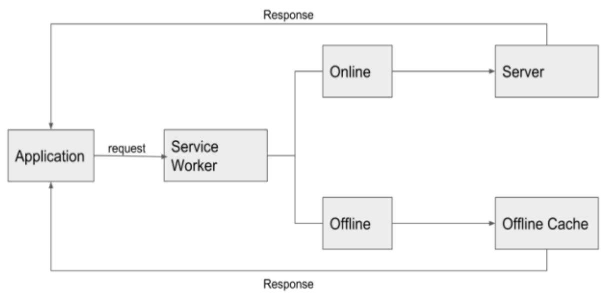
UML was created as a result of the chaos re volving around software development and doc umentation. In the 1990s, there were several different ways to represent and document soft ware systems. The need arose for a more unified way to visually represent those systems an d as a result, in 1994-
1996, the UML was developed by three software engineers working at Rational Software. It was lat er adopted as the standard in 1997 and has remain ed the standard ever since, receiving only a few u pdates. Mainly, UML has been used as a general-purpose modeling language in the field of softwar e engineering. However, it has now found its way into the documentation of several business proces ses or workflows. For example, activity diagrams, a type of UML diagram, can be used as a replace ment for flowcharts. They provide both a more sta ndardized way of modeling workflows as well as a wider range of features to improve readability a nd efficacy. UML itself finds different uses in soft

ware development and business process document ation: UML diagrams, in this case, are used to co mmunicate different aspects and characteristics of a system. However, this is only a top-level view of the system and will most probably n ot include all the necessary details to execute the project until the very end. Forward Design – The design of the sketch is done before coding t he application. This is done to get a better view of the system or workflow that you are trying to cre ate. Many design issues or flaws can be revealed, thus improving the overall project health and well -being. Backward Design – After writing the code, the UML diagrams are dr awn as a form of documentation for the different activities, roles, actors, and workflows.

Modular Design for the System Function

Modular design is basically a method that divides a system or project into smaller segments, which can then be brought together to function as one. The main thing to note about modular design is that it refers to small parts that are create independently yet can still be used in different systems to power manifold functionalities. The diagram below shows the modular design for this project.

Figure: 8



RESULTS

This chapter describes the implementation of the Farmer's support progressive web application system, the prototype developed in this study demonstrate the features and benefits of PWA. The application thus developed has all the components of PWA such as Service Worker, Web App Manifest, App shell and Web push notification. Farmer's support progressive web application, the user can browse the weather report and get the report for the next four days and get notified for the latest news via push notification, install app on the home screen with just one tap and use offline with UI and UX like a native app.

4.2 Baseline Study Results

Building progressive web apps and meeting all the requirements based on Performance, Accessibility, Best Practices and SEO was not easy to be archived. All that makes it happen, all the components of PWA, i.e. Service Worker, Web App Manifest, Application Shell model, and Web Push notification need to be implemented with great care and work hand in hand. All these components are described in the following section.

Web App Manifest

Web App Manifest is a simple JSON file containing information: name, short name, description, icons for different device resolution, starts URL, display mode, theme color of the application. The use of Web App Manifest installs the web app in the user home screen between the native apps. As a result, the user can get quick access and enjoy full-screen display like with the native app. (Google Inc.2016) Listing 1 shows the manifest json file for a web application.

For the web app to be able to appear as install banner on sites and provide an app-like experience, the web app must fulfill the following criteria:

1. The site needs to be served over HTTPS.
2. The site needs to have a service worker registered.
3. The web app manifest file of the site should have at least the four mandatory fields name

or short name (preferably both, start URL, icons and display). (Google Chrome Developers.2017)

As illustrated in Figure 15, a web app install banner is triggered by the browser when manifest file meets above mentioned criteria so the user can install app on home screen and enjoy full screen experience like a native app. The web app installs banner user prompt that Chrome will trigger to indicate that the user can add your web app to the users' home screen. It will only prompt when a number of criteria have been met:

1. The app uses a service worker
2. The site is using HTTPS
3. The app has a manifest declared
4. The manifest has a short name, 144-pixel icon and a type of 'image/png'

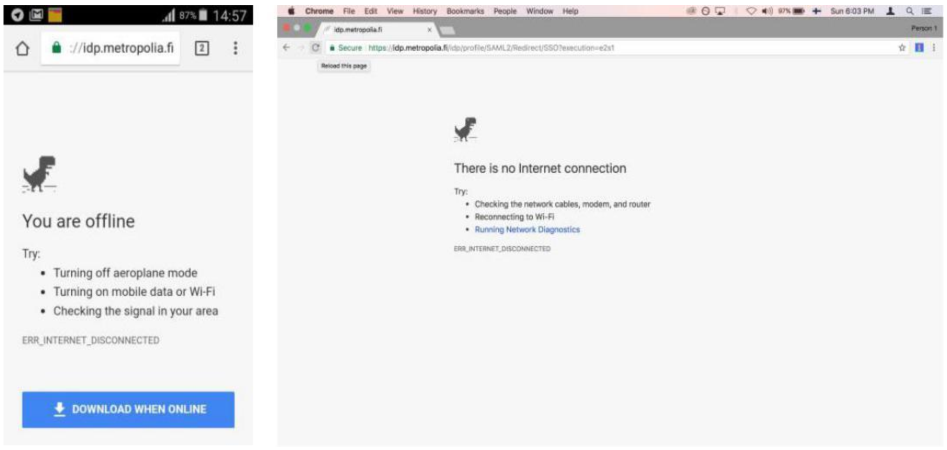
A web app manifest file can be verified manually by using the Manifest tab on the Application panel of Chrome DevTools, as shown in figure 9.

Web App Manifest in Chrome Developer Tools A pplication panel
Web app manifest has compatibility issue with browsers. Not all browser support manifest file. The compatibility of web app manifest with different browser can be checked by using online validation tool as shown in figure 29.

As illustrated in figure 4, Web App Manifest works best for the Chrome, Chrome for An-droid, UC Browser for Android, Safari, iOS Safari have recently started supporting the manifest file whereas, Firefox and MS Edge have no support till date.

Service Workers

The web as it is today is rich, beautiful and useful but when users visit a web app; when they have poor connection or have lost their connectivity, they are shown the page 'there is no internet connection' as shown in figure 9.


You are offline

Try:
- Turning off aeroplane mode
- Turning on mobile data or Wi-Fi
- Checking the signal in your area

ERR_INTERNET_DISCONNECTED

There is no Internet connection

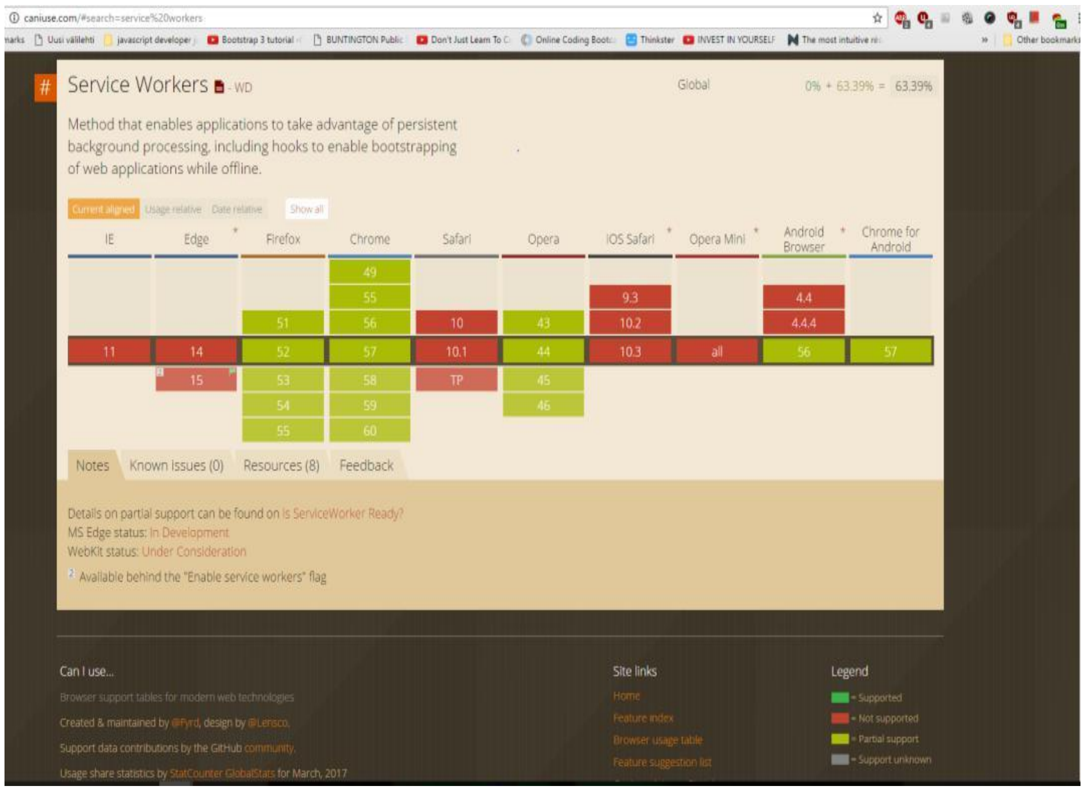
Try:
- Checking the network cables, modem, and router
- Reconnecting to Wi-Fi
- Running Network Diagnostics

ERR_INTERNET_DISCONNECTED
://idp.metropolia.fi
DOWNLOAD WHEN ONLINE
Chrome File Edit View History Bookmarks People Window Help
idp.metropolia.fi
Secure https://idp.metropolia.fi/idp/profile/SAML2/Redirect/SSO?execution=e2s1
Reload this page

Figure 9 A web app in offli ne state in mobile and desk top view. Source: Author, 20 19.

As illustrated in figure 18, offline state of web app could not provide any useful information to users. But the introduction of a service worker has turned this error as something that can be handled with grace.
A Service Worker is an event-driven script that runs in the background separately from the webpage, reacts to events and intercepts network requests of application or website with server and resources. It works as proxy between the network and the browser. It can run even when the application is closed, thus it serves to trigger events even when the site is closed. Features like push notifications and background sync are possible in web today because of Service Workers. In the future, the service worker shall back other cool features like periodic sync. (Archibald, Jake.2017) Since the service worker can intercept the request, modify content or even completely replace with new responses, only pages served over secure connections (HTTPS) can register a service worker. This is to protect

users and prevent man-in-the-middle attacks. ((Archibald, Jake.2017). Figure 6 shows the caching strategy of service worker.



Service Workers - WD

Global 0% + 63.39% = 63.39%

Method that enables applications to take advantage of persistent background processing, including hooks to enable bootstrapping of web applications while offline.

Current aligned Usage relative Date relative Show all

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
			49						
			55			9.3		4.4	
		51	56	10	43	10.2		4.4.4	
11	14	52	57	10.1	44	10.3	all	56	57
	15	53	58	TP	45				
		54	59		46				
		55	60						

Notes Known Issues (0) Resources (8) Feedback

Details on partial support can be found on Is ServiceWorker Ready?
MS Edge status: In Development
WebKit status: Under Consideration
[2] Available behind the "Enable service workers" flag

Can I use...
Browser support tables for modern web technologies
Created & maintained by @Fyrd, design by @Lensco.
Support data contributions by the GitHub community.
Usage share statistics by StatCounter GlobalStats for March, 2017

Site links
Home
Feature index
Browser usage table
Feature suggestion list

Legend
🟩 = Supported
🟥 = Not supported
🟨 = Partial support
⬜ = Support unknown

Figure 10 Compatibility of browsers for service workers *(comScore Inc.2016)*

As shown in figure 10, most of the browsers such as Chrome, Firefox, Opera, Chrome for Android Safari, iOS Safari, Edge support Service Worker. Opera Mini, and Internet Explorer do not support service worker till date.

Service Worker Life Cycle

As service worker is an event-driven script, it has a short life span. It wakes up with events and runs only if it needs to process the event. Through Service Worker the developer can treat the network as enhancement, control caching of resources on a proper way. Control over the cached resources plays an important role in developing offline application, which is one of the key features of PWA. Via service worker the

webpage is available even offline and with cached data loads faster even on a flaky network. (Russell, Alex.2016). The lifecycle of the service worker is its most complicated part. If you don't know what it's trying to do and what the benefits are, it can feel like it's fighting you. But once you know how it works, you can deliver seamless, unobtrusive updates to users, mixing the best of web and native patterns. This is a deep dive, but the bullets at the start of each section cover most of what you need to know.

The intent of the lifecycle is to:

Make offline-first possible.

Allow a new service worker to get it ready without disrupting the current one. Ensure an in-scope page is controlled by the same service worker (or no service worker) throughout.

Ensure there's only one version of your site running at once.

That last one is pretty important. Without service workers, users can load one tab to your site, and then later open another. This can result in two versions of your site running at the same time. Sometimes this is ok, but if you're dealing with storage you can easily end up with two tabs having very different opinions on how their shared storage should be managed. This can result in errors, or worse, data loss.

The install event is the first event a service worker gets, and it only happens once.
A promise passed to install Event. Wait Until signals the duration and success or failure of your install.
A service worker won't receive events like fetch and push until it successfully finishes installing and becomes "active".
By default, a page's fetches won't go through a service worker unless the page request itself went through a service worker. So, you'll need to refresh the page to see the effects of the service worker.
A service worker has a totally separated lifecycle from a web page. Figure 11 illustrates a simplified version of a service worker lifecycle on its first installation.

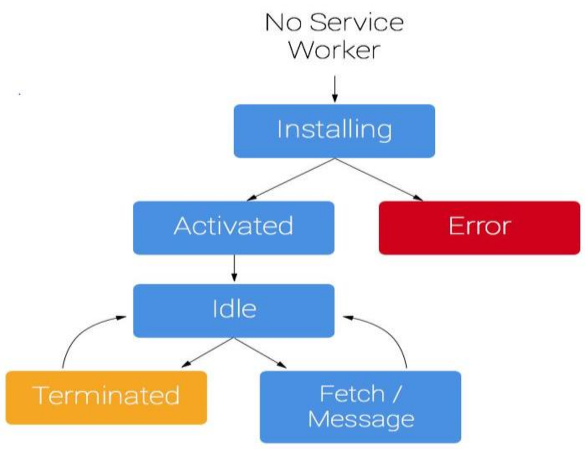


Figure 11 illustrates a simplified version of a service worker lifecycle on its first installation.
Figure 11 UML App Manifest
Source: *By Matt Gaunt (2019)*

Survey Results and Discussion

The web app manifest is a simple JSON file that tells the browser about a web application and how it should behave when 'installed' on the user's mobile device or desktop. Having a manifest is required by Chrome and other supporting browsers to show the Add to Home Screen prompt.
A typical manifest file includes information about the app name, icons it should use, the start url it should start at when launched, and more.
The web app manifest was also required for making the web application installable. In this context installable refers to being able to add an icon to the user's home screen. It was completed and worked ok. Below was the status of the Manifest. The App manifest was successfully created check and the following images show the results. To manually verify my manifest was setup correctly, I used the Manifest tab on the Application panel of Chrome DevTools. This tab provides a human-readable version of many of the manifest's properties. I could also simulate Add to Home Screen events from there. When I wanted an automated approach towards validating my web app manifest, I checked out Lighthouse. Lighthouse is a web app auditing tool. It's built into the Audits tab of Chrome DevTools, or can be run as an NPM module. I also provided Lighthouse with a URL, it was able to run a suite of audits against my page, and then displayed the results in a report.

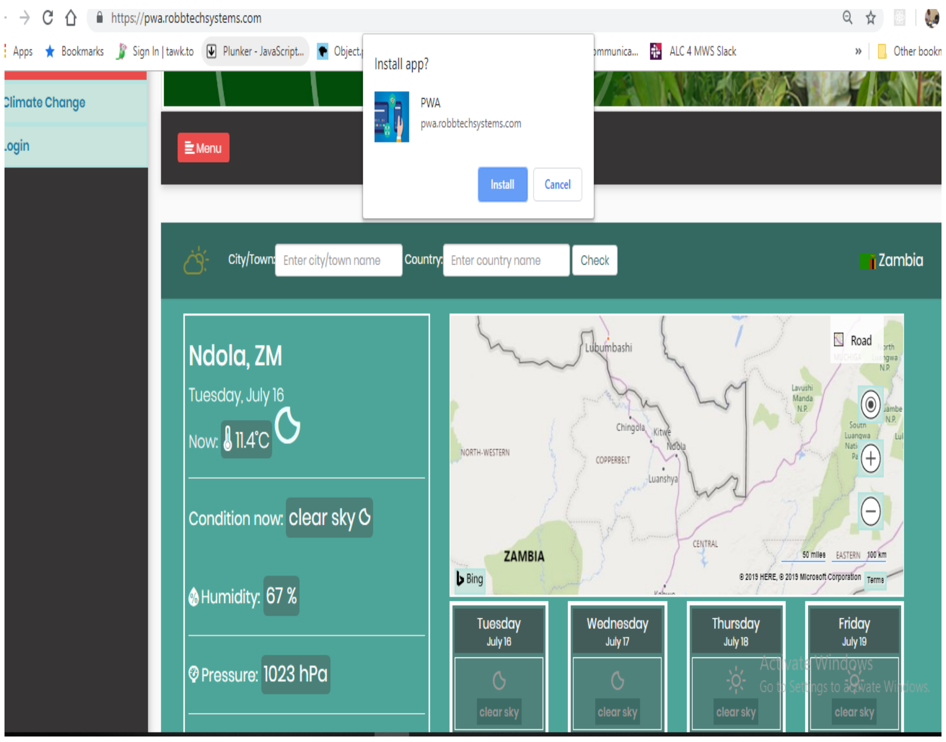
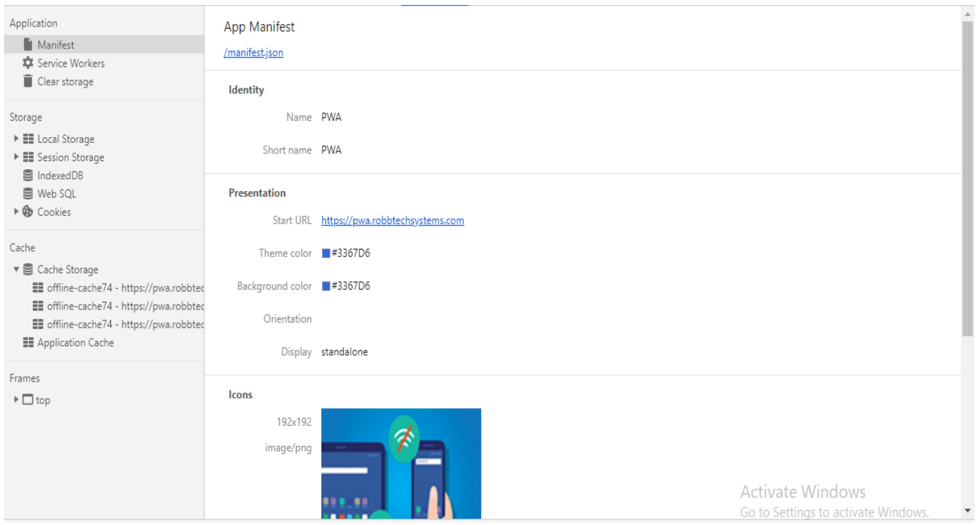


Figure 12 UML App Manifest prompting to install an app on the desktop computer.
Source: Author

The service worker was created and it was able to run smoothly below is the status of the service worker with an active status. On the left part of an image is cache storage.

Push Notification

The push notification was created and test and it worked well as shown in the Figure 13 below. A notification is a message that pops up on the user's device. Notifications can be triggered locally by an open application, or they can be "pushed" from the server to the user even when the app is not running.


Application
Manifest
Service Workers
Clear storage
Storage
Local Storage
Session Storage
IndexedDB
Web SQL
Cookies
Cache
Cache Storage
offline-cache74 - https://pwa.robbtec
offline-cache74 - https://pwa.robbtec
offline-cache74 - https://pwa.robbtec
Application Cache
Frames
top

App Manifest
/manifest.json

Identity

Name	PWA
Short name	PWA

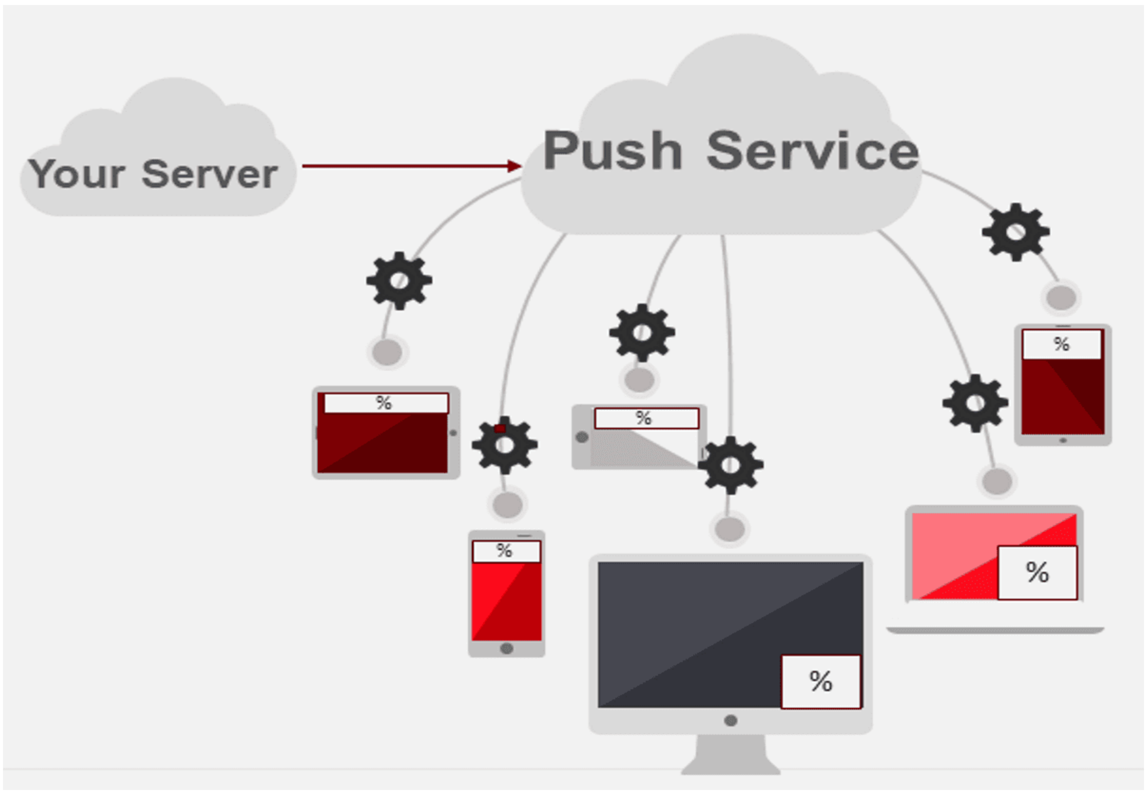
Presentation

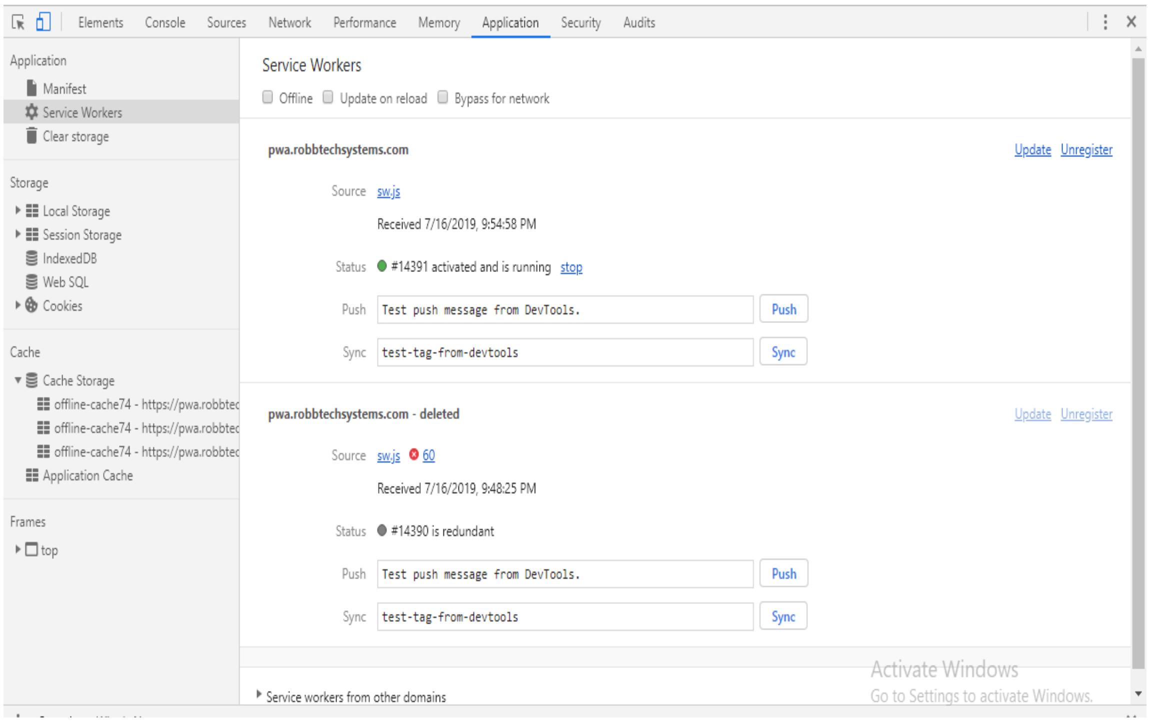
Start URL	https://pwa.robbtechsystems.com
Theme color	#3367D6
Background color	#3367D6
Orientation	
Display	standalone

Icons
192x192
image/png

In my case the notification comes from the server, they allow users to opt-in to timely updates and allow you to effectively re-engage users with customized content. Push Notifications are assembled using two APIs: the Notifications API and the Push API. The Notifications API lets the app display system notifications to the user. The Push API allows a service worker to handle Push Messages from a server, even while the app is not active. The Notification and Push API's are built on top of the Service Worker API, which responds to push message events in the background and relays them to the application.

The following Figure 14 shows how the push message moves





Application
Manifest
Service Workers
Clear storage

Storage
Local Storage
Session Storage
IndexedDB
Web SQL
Cookies

Cache
Cache Storage
offline-cache74 - https://pwa.robbtec
offline-cache74 - https://pwa.robbtec
offline-cache74 - https://pwa.robbtec
Application Cache

Frames
top

Service Workers

Offline   Update on reload   Bypass for network

pwa.robbtechsystems.com   Update   Unregister

Source	sw.js
	Received 7/16/2019, 9:54:58 PM
Status	● #14391 activated and is running stop
Push	Test push message from DevTools. Push
Sync	test-tag-from-devtools Sync

pwa.robbtechsystems.com - deleted   Update   Unregister

Source	sw.js ⊗ 60
	Received 7/16/2019, 9:48:25 PM
Status	● #14390 is redundant
Push	Test push message from DevTools. Push
Sync	test-tag-from-devtools Sync

Service workers from other domains

Figure 15 UML Push Notification Architecture

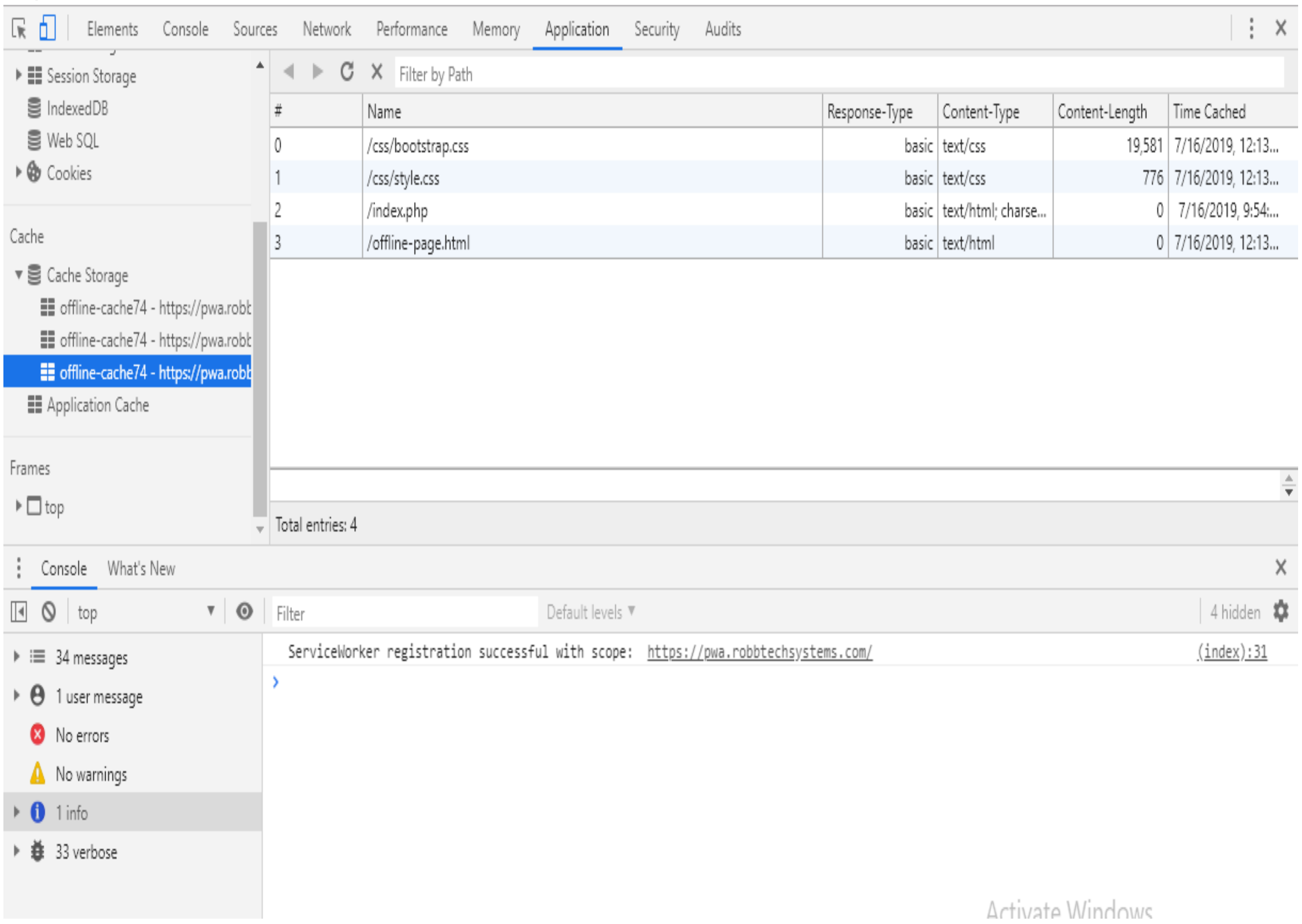
Figure 14 Service worker

Source: Author, 2019

The service worker was created and it was able to run effectively as shown in figure 14. Files are not the only content that can be cached. localStorage is great to persist key value pairs where the values are strings. IndexedDB is more robust and can store many more types of data efficiently. I think of it as a light weight document database in the

browser. appCache and service worker cache persist files, URL addressable resources to be technically correct. But service worker cache is not the only browser storage medium you need to monitor.
IndexedDB is a low-level API for client-side storage of significant amounts of structured data, including files/blobs. This API uses indexes to enable high-performance searches of this data. While Web Storage is useful for storing smaller amounts of data, it is less useful for storing larger amounts of structured data. IndexedDB provides a solution IndexedDB is a transactional database system, like an SQL-based RDBMS. However, unlike SQL-based RDBMSes, which use fixed-column tables, IndexedDB is a JavaScript-based object-oriented database. IndexedDB lets you store and retrieve objects that are indexed with a key; any objects supported by the structured clone algorithm can be stored. You need to specify the database schema, open a connection to your database, and then retrieve and update data within a series of transaction. In this project an IndexedDB was use because of the nature of the project. Data from the weather API was cached for future use.

Figure 16 Service worker



#	Name	Response-Type	Content-Type	Content-Length	Time Cached
0	/css/bootstrap.css	basic	text/css	19,581	7/16/2019, 12:13...
1	/css/style.css	basic	text/css	776	7/16/2019, 12:13...
2	/index.php	basic	text/html; charse...	0	7/16/2019, 9:54:...
3	/offline-page.html	basic	text/html	0	7/16/2019, 12:13...

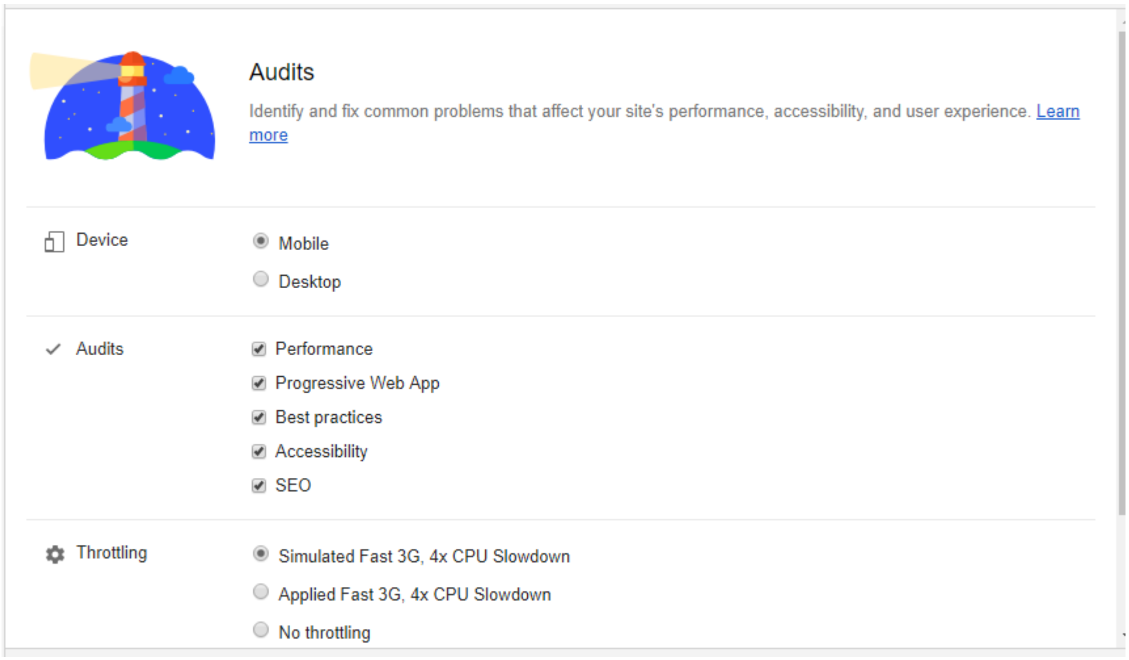
Total entries: 4

```
ServiceWorker registration successful with scope:  https://pwa.robbtechsystems.com/          (index):31
```

Source: Author, 2019

After creating a service worker and an App Manifest to a web page it was impotant to test the performance of the web by runing Lighthouse. This is an open source, automated tool for improving the quality of web pages. Lighthouse is an open-source, automated tool for improving the performance, quality, and correctness of your web apps.

When auditing a page, Lighthouse runs a barrage of tests against the page, and then generates a report on how well the page did. From here you can use the failing tests as indicators on what you can do to improve your app. You can run it against any web page, public or requiring authentication. It has audits for performance, accessibility, progressive web apps, and more. You can run Lighthouse in Chrome DevTools, from the command line, or as a Node module. You give Lighthouse a URL to audit, it runs a series of audits against the page, and then it generates a report on how well the page did. From there, use the failing audits as indicated on how to improve the page each audit has a reference.



Audits

Identify and fix common problems that affect your site's performance, accessibility, and user experience. Learn more

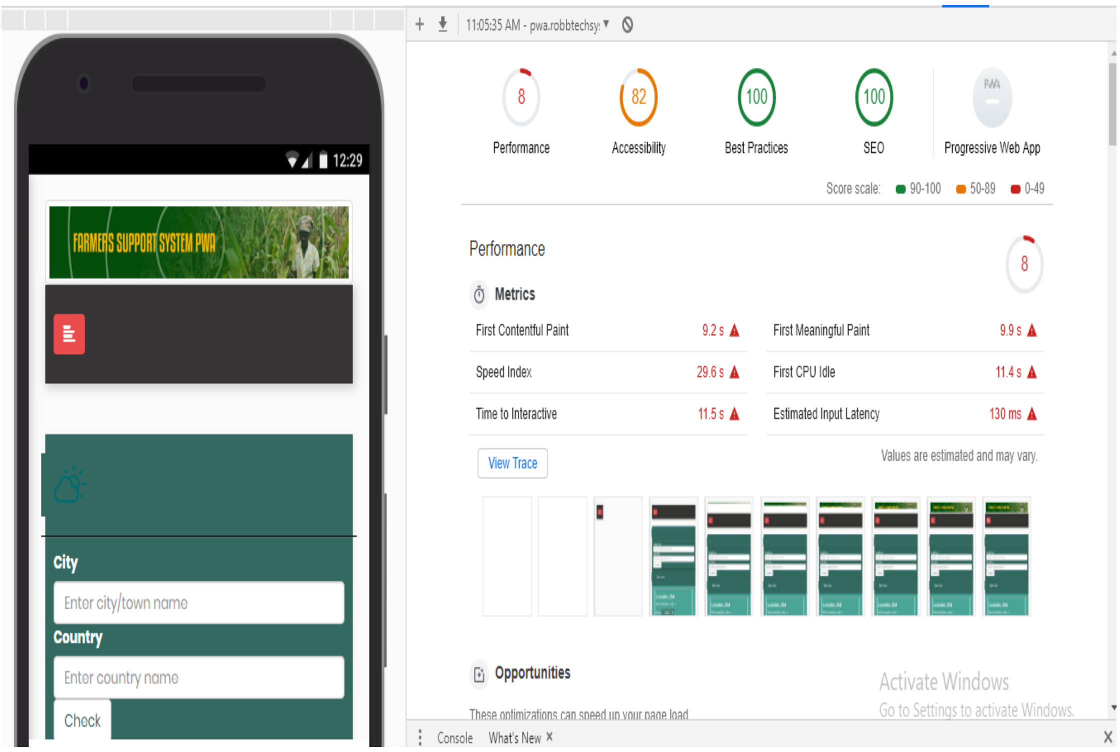
Device: [✓ Mobile] Desktop

Audits:
- ✓ Performance
- ✓ Progressive Web App
- ✓ Best practices
- ✓ Accessibility
- ✓ SEO

Throttling: [✓ Simulated Fast 3G, 4x CPU Slowdown] Applied Fast 3G, 4x CPU Slowdown; No throttling

Figure 17 Light house for testi ng the web Source: Author, 2019

Figure 18 First Audit results: Source: Author, 2019



11:05:35 AM - pwa.robbtechsys

Performance	Accessibility	Best Practices	SEO	Progressive Web App
8	82	100	100	PWA

Score scale: 🟩 90-100 🟧 50-89 🟥 0-49

Performance 8

Metrics

First Contentful Paint	9.2 s	First Meaningful Paint	9.9 s
Speed Index	29.6 s	First CPU Idle	11.4 s
Time to Interactive	11.5 s	Estimated Input Latency	130 ms

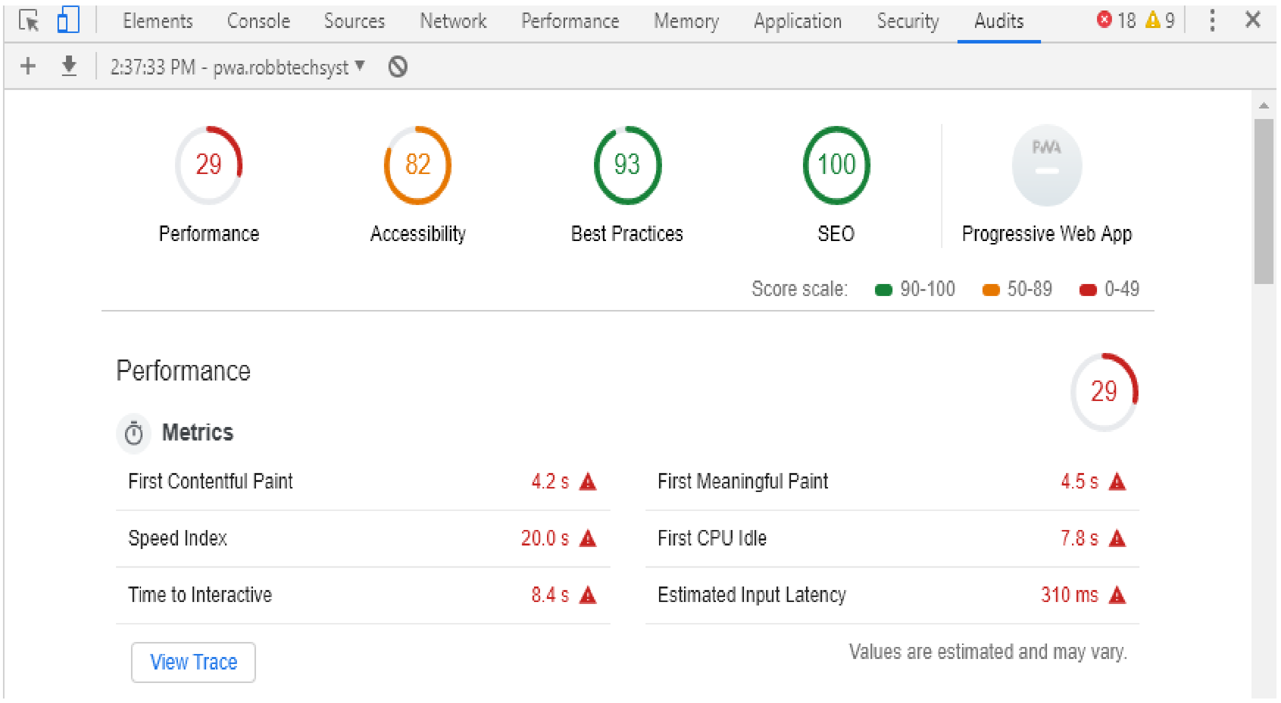
Values are estimated and may vary.

Opportunities

These optimizations can speed up your page load

The following results were obtained after run at test for the first time

The results indicate very low performance that was beforee optimising images and javascripts.



Performance 29

Metrics

First Contentful Paint	4.2 s ▲	First Meaningful Paint	4.5 s ▲
Speed Index	20.0 s ▲	First CPU Idle	7.8 s ▲
Time to Interactive	8.4 s ▲	Estimated Input Latency	310 ms ▲

Values are estimated and may vary.

Figure 19 shows second Audit results: Source: Author, 2019

The performance increased from 8 to 29 after optimizing images, the warning sign shows areas that need improvements.

4.3 System Implementation Results

The application was developed on HP Probook as the hardware and Windows 10 as an operating system. Wampp server was downloaded and installed and it was used as a local hosting server. The open weather API was also incorporated in order for the system to access the weather data. Visual Studio Code editor was also installed. Visual studio code is a free, lightweight and robust source code editor, which leverage the development of application through its features like embedded CLI and built-in git support. It is available for all the platforms: macOS. Windows and Linux. It has intelligent code completion, streamlined debugging and In-Product Source Control. It also comes with built-in support for Typescript, JavaScript and Node.js and extension for C#, C++, Java, Python, PHP and Go. Moreover, it also has runtime such as Unity and .NET.

HTTPS

In order for PWA to work properly it has to run on a secure protocol, certificate has to be installed SSL here a cloud hosting server was used to test the prototype application.

CONCLUSION

The Farmer's support system progressive web application will be used to help farmers with information regarding weather pattern and monitor the crops remotely. The system is able to display weather report for the next four day or more, that will help farmers to plan for the season. The application is capable of running even without internet connection or with a slow network without displaying a connection error page even when the network is disconnected and refreshed contents will be intact because it uses the first offline approach. The system is also able to show the weather focus in any city of any country.

Development of the System as a solution

Technology has become a key component of business in every sector and every business is trying to engage its customers by using website and mobile phones. This System will provide a solution to the challenges businesses have been having if the business adopts a progressive web application then it finds a solution for a mobile application, no need for developing other mobile applications. Progressive web application also comes as a cost saving method of developing cross platform applications. (Russell, Alex. 2015)

Comparison with other similar works

Jan von der Assen (2018) in his paper titled "A Progressive Web App (PWA)-based Mobile Wallet for Bazo" created a fund transfer Progressive web application which was able to work offline. Users could enter their transaction while offline and when they go online the transaction could be processed.
Parbat Thakur (2018) developed a PWA news app which had a reachability of the web and capabilities like a native application. PWA News App could work offline, install on home screen, launch in full screen mode without any challenge also reengaged users by sending relevant push notification and load fast even on flaky networks. "The prototype application was eventually functioning regardless of network state to the greatest possible extent.

Anna Zacharuk (2018) carried a research on both PWA and Native app and she states that the competition between PWA and native applications is slightly uneven because these are two different technologies even if we try to explain in simple terms that PWA is a technology that adjusts to all devices and works with all operating systems, while native apps – in order to give the same effect – must be created twice: separately for each system (iOS and Android), it becomes clear the absolute winner is PWA. But we all know the devil is in the detail of PWAs are progressive internet apps, or websites with additional functions, which make them look and work the same on various systems and various screens – systems treat them as native system applications. The user is unable to tell the difference (it is technically a web application). The website A native app, on the other hand, is created for a specific platform, that is for iOS (in Swift or Objective-C) or Android (Java). What's important and different from PWA, is that a native app can fully communicate with the device on which it's going to be opened. PWA and native apps are comparable, because PWA was designed as a "native like experience", which means it should work like a native application. So, we can investigate their usability, speed, opening time, offline mode and several other functions. Before we get to a detailed comparison, it is worth explaining the controversies around PWA and Apple. This turbulent story is slowly getting closer to a happy ending. As PWA started to draw companies' attention and was celebrating its first successes, Apple saw it as problematic for the simple reason that with the fast development of PWA, App Store wouldn't have any reason to exist. Initially, PWA worked on iOS in such a way that many solutions on the websites were available, but the maximum capabilities of PWA were not available (e.g. no implementation of Service Worker API in Safari browser or no Web App Manifest files). PWA has had some fat years and is still on top. Under such circumstances, Apple had to be reasonable. First, they pulled in their horns by making more functions available, and currently we can see a growth in aggressive removal from iOS App Store apps which were built with the use of commercialized iOS application templates. Apple is also cleaning App Store from applications thought to be copies, clones, false, abandoned or incompatible with 64 bits. Experts believe this step means that Apple has begun to promote PWA technology creation and is following its expanding concept. There are also experts who claim that Apple has never been against PWA.

Conclusion and Possible application

The research focused on building a progressive web application and assessing whether it is time to abandon native application and move on with progressive web app. PWAs are meant to close the existing gap between the web and native mobile applications. Due to the emergence of sophisticated yet standardized technologies they are able to implement characteristics which eventually allow them to deliver a significantly enhanced user experience. Consequently, PWAs may offer exciting opportunities by elevating the web to native spheres. Several of the leveraged technologies may be still evolving and it might take some time until they are widely supported, however, with concepts like progressive enhancement this does not constitute a problem. Rather it fits in perfectly with the overall idea. The established characteristics may be implemented incrementally and subsequently improve the experience for everyone in the application's audience. Meanwhile, NativeScript demonstrates how technologies, previously only used on the web, can be used today to develop highly native applications on a conveniently abstracted development layer. The starting point of PWAs poses interesting environment for looking at NativeScript applications. Though not all characteristics are directly applicable, they still offer a solid foundation for knowing what may be expected from a mobile application. All of these requirements could be practicably implemented or otherwise certified through the presence of certain properties, concepts or features of NativeScript to a far extent. Of course, certain issues occurred during the development and shortcomings were identified, yet NativeScript may be able to provide some characteristics earlier than it would be possible on the web. At the same time, however, development can be conducted very much like for the web. Indeed, with the right architecture to ensure proper encapsulation it may even be possible to seamlessly share large parts between a PWA and a NativeScript application.

The differences between both solutions in relation to individual features are rather minor. They both seem to allow for the creation of more than decent user experiences while mitigating the need for cumbersome native developments. However, NativeScript might not be able to offer the same progressive nature as PWAs. The successive elevation of web applications presents innovative ways to think about user funnel optimization. NativeScript applications just cannot compete here as they are inevitably working in the same realms as plain native ones. On the other hand, the framework may allow the use of advanced features on systems which are not yet compatible with the most recent web standards.

Therefore, as often is the case, the individual priorities have to be evaluated when deciding between the approaches as they both seem to offer interesting opportunities but might not fit everyone's needs. It may well be that PWAs eventually gain great general attention in the future, yet, one may implement a NativeScript application today and transform it into a PWA at a later point. This may be accomplished with arguably little effort due to the technical and architectural overlaps. Looking forward, web technologies, especially JavaScript, seem to grow increasingly powerful and extend to more and more areas of application. As shown, their usage is not restricted to simply making web content slightly more dynamic but rather be the basis for full-featured applications delivering immersive user experiences to various platforms. Furthermore, during the creation of this report, the third major version of NativeScript was released. With improvements in performance, the elimination of several issues and new features it poses even better abilities to implement the criteria laid down in this report. Hence, the prospects for the future development of web and mobile development are exciting as they seem to converge in certain aspects and conflict in others, consequently sparking progress and remarkable innovations.

In the course of this report, PWAs and the characteristics of their development could be described in detail after the conditions for the later conducted analysis were defined. Moreover, the thematic backgrounds regarding mobile cross-platform development with NativeScript were provided in order to subsequently prepare the analysis by laying down verifiable criteria. The afterwards derived criterion was comprehensively applied through prototypic implementations and sound research to assess NativeScript and applications developed with the framework. The findings of these examinations were then summarized and weighed accordingly in a discussion. Lastly, the relevant solutions were comparatively debated against the established background resulting in concluding considerations and an appropriate outlook.

Future Works

Based on the results, it is recommended to continue improving the performance of the PWA Farmer support system by focusing on the performance and by doing small improvements where ever possible. It is also important to measure the impact of these improvements in order to know whether the actions are actually improving the performance. As the results from the lighthouse performance evaluation suggests, focusing on the bottlenecks in the page loading and rendering may enable improving the page loading speed and winning some valuable time. Furthermore, it is recommended to set up continuous performance monitoring in order to notice if new performance bottlenecks are caused by new features before they go to production. This continuous focusing on performance would allow rewarding developers or teams who are improving the performance. Page loading performance budgets could be defined and alarms when exceeding the budgets could be utilized for notifying about downgraded performance. However, the loading performance of the websites varies based on the content and advertisement campaigns on the site. Therefore, a short-term performance monitoring solution will not provide reliable results in a changing environment like this. A solution to this problem may be creating a static test environment that is as close to the production version as possible. The test environment should contain as little changing elements as possible in order to compare the page loading results over time. The advertisements and other changing content should be disabled on this environment to achieve this goal. The static test site may be enough for evaluating the performance impact of the changes that are going to be released before releasing them. Additionally, a longer-term monitoring solution on the actual site may provide useful information about the trends in the performance of the site. In addition to the above-mentioned actions, there are a few actions that are not recommended at the time being. Re-implementing the PWA Farmer's support system front end as a JavaScript single-page application may have a positive impact on the performance. However, if not done properly, it could even slow down the page load times due to large scale JavaScript applications having to download large bundles of application code before rendering the content. This is possible to avoid by utilizing server-side-rendering in combination with client-side application logic. On the other hand, if done correctly it could reduce the page load times and especially the navigation between articles. The enormous effort required for the rewrite may be hard to justify, especially from the performance point of view. Another option would be implementing AMP support for the PWA farmers' support system. This would allow Google to serve the content rapidly from the AMP cache and improve the visibility in the mobile search results. There is nothing wrong with AMP as an idea for improving web performance on mobile devices. However, it is heavily controlled by Google and restricts the possibilities of the developers in addition to moving traffic to Google's servers where the AMP cache is stored. It would also be difficult to make sure that the content stays always up-to-date if the content is served from the AMP cache. Implementing AMP support in the SNDP is therefore not recommended. During the writing of the report, even wider support for service workers was introduced. Now, all the major browsers support service workers, including Microsoft Edge and Safari both on MacOS and on iOS as seen in Figure 6. This may change the situation when considering whether building a PWA is worth it. Wider browser support means reaching larger audience with the possible performance improvements. However, in the current situation it is not recommended to continue to develop the prototype PWA further unless features such as push notifications are needed. From the performance point of view a PWA is not the best option for improving the performance of the Farmers support system at the moment. By refactoring the front-end structure in other ways, either into a JavaScript SPA or static HTML page with dynamic content loading strategy, a faster loading experience may be achieved. After improving the performance in other ways, the PWA option should be reconsidered. By that time, the PWA implementation differences between the different platforms have probably been worked on and decreased making a PWA a more suitable solution for multi-platform web application development. Progressive web apps have benefits for everyone involved. The user will be able to instantly install the "app" without a visit to the app store and a large download, which can be an unpleasant experience on a slow connection. Organizations can go back to developing web apps without requiring the requisite separate Android and iOS teams. They can update and "release" their app without going through the app store approval process. Releases and defect fixes can be deployed immediately. Web design elements are immediately picked up by the progressive web app. A progressive web app is a website that combines the best experiences of the web and an app. They don't require any installation. The app loads quickly, even when the user is on bad networks. It can send relevant push notifications to the user and has an icon on the home screen and loads as top level, full screen experience. Application shell architectures come with several benefits but only make sense for some classes of applications. The model is still young and it will be worth evaluating the effort and overall performance benefits of this architecture. Progressive web apps are an interesting forward look into the future of mobile apps. It will become a key factor in the world of apps.

The enhancement of a traditional webpage with Progressive Web Application features will make it fast, reliable and engaging. The webpage is added with the tools like Service Workers which make it work offline. The App Shell architecture helps in caching the contents separately thus making the process of retrieving faster. The Service Worker also helps in setting up the push notification and makes the webpage to be made as an icon on desktop thus provide app like experience

Acknowledgement

First and foremost, I would like to thank God almighty for the good health he has always provided for me even during the period I was working on the research. My wife Sylvia C. Mchinzi for the encouragement, Mr. Lameck Nsama and Mrs. Sylvia Nanyangwe and Mr. Kaela Kamweneshe (IJMDR-Editor), my project supervisors for the appropriate guide provided during my research. I would like also to appreciate Open weather for allowing me to use their weather API, also Bing cooperation for allowing me to use their map API
.

REFERENCES

[1] Ali Express Showcase," in *developers.google.com*. [Online]. Available:

[2] Anderson, David J. *Kanban: Evolutionäres Change Management für IT-Organisationen.* [trans.] Arne Roock and Henning Wolf. 1. Heidelberg: dpunkt. Verlag, 2011. ISBN 978-3-86491-027-2.

[3] Anderson, Nathanael J. *Getting Started with NativeScript.* Birmingham: Packet Publishing Ltd., 2016. ISBN 978-1-78588-865-6.

[4] April 4, 2017. [Cited: May 2, 2017.] https://github.com/VideoSpike/nativescript-web-image-cache.

[5] Bundling Script Code with Webpack. *NativeScript.* [Online] April 11, 2017. [Cited: May 3, 2017.] http://docs.nativescript.org/angular/tooling/bundling-with-webpack.html.

[6] Creating Launch Screen and App Icons for Android. *NativeScript.* [Online] September 22, 2016. [Cited: May 5, 2017.] https://docs.nativescript.org/publishing/creating-launch-screens-android.

[7] Enabling Deep Links for App Content. *Android Developers.* [Online] [Cited: April 5, 2017.] https://developer.android.com/training/app-indexing/deep-linking.html.

[8] Firebase Cloud Messaging. *Firebase.* [Online] April 3, 2017. [Cited: April 6, 2017.] https://firebase.google.com/docs/cloud-messaging/.

[9] Launch screens. *Material design guidelines.* [Online] [Cited: April 4, 2017.] https://material.io/guidelines/patterns/launch-screens.html.

[10] Layouts. *NativeScript.* [Online] March 17, 2017. [Cited: May 4, 2017.] https://docs.nativescript.org/angular/ui/layouts.html.

[11] Making Your App Content Searchable by Google. *Android Developers.* [Online] [Cited: April 5, 2017.] https://developer.android.com/training/app-indexing/index.html.

[12] Mehlhorn, Nils. *Modern Cross-Platform Development for Mobile Applications.* Faculty of Media, Hochschule Düsseldorf University of Applied Sciences. Düsseldorf, 2016/2017. Scientific Consolidation. Publication revision.

[13] Multithreading Model. *NativeScript.* [Online] November 7, 2016. [Cited: April 4, 2017.] https://docs.nativescript.org/core-concepts/multithreading-model.

[14] NativeScript 3.0 Available Today. *NativeScript.* [Online] May 3, 2017. [Cited: May 8, 2017.]

https://www.nativescript.org/blog/nativescript-3.0-available-today.

[15] NativeScript/android-runtime: Android runtime for NativeScript (based on V8). *GitHub.* [Online] April 24, 2017. [Cited: May 4, 2017.] https://github.com/NativeScript/android-runtime.

[16] Notifications. *Android Developers.* [Online] [Cited: April 6, 2017.] https://developer.android.com/guide/topics/ui/notifiers/notifications.html.

[17] Publishing for Android. *NativeScript.* [Online] December 9, 2016. [Cited: May 2, 2017.] https://docs.nativescript.org/publishing/publishing-android-apps.

[18] Styling. *NativeScript.* [Online] May 2, 2017. [Cited: May 4, 2017.] https://docs.nativescript.org/angular/ui/styling.html.

[19] What is Android Runtime for NativeScript? *NativeScript.* [Online] [Cited: November 25, 2016.] http://docs.nativescript.org/runtimes/android/overview.