**UNIVERSITY OF WATERLOO**

Faculty of Mathematics

**A BRIEF ANALYSIS OF PROGRESSIVE WEB APPLICATIONS**

Royal Bank of Canada

Toronto, Ontario

Prepared by

Yingzao Li

3A Computer Science

ID 20688077

April 2019

**Memorandum**

To: Mr. Anand Supner

From: Yingzao Li

Date: April 10, 2019

RE: Work Term Report: A Brief Analysis of Progressive Web Applications

I have prepared this report, titled “A Brief Analysis of Progressive Web Applications”, for my 3A work term. This is the third report that I have completed for my Bachelor of Computer Science degree in the Co-operative Education Program at the University of Waterloo. It has not received any prior academic credit.

In a team lead by Anand Supner at RBC, a dashboard application for investment advisors is built to provide a better user experience. My responsibility as a software developer intern is to add widgets to the dashboard, build backend services and maintain the codebase. The report is written inspired by the possibility to make this Angular web application a Progressive Web App (PWA).

I would like to thank my coworkers, teammates, as well as supervisors for helping me with coding and finishing this report.

Thank you for your attention and assistance.

Yingzao Li

**Table of Contents**

Executive Summary……………………………………………………………….………3

1 - Introduction …………………………………………………………………….……...4

2 - Analysis............................................…………………………………………........…...5

2.1 - The Characteristics of a PWA …………...………………….…............…………5

2.2 - The Components of a PWA ……...……………………………….............…...…7

2.2.1 - App Shell…………...………………………...……….…............…………7

2.2.2 - Service Worker...…...………………………...……….…............…………8

2.2.3 - Manifest…….....…...………………………...……….…............…………8

2.3 – Comparison between PWA and web app….………………..............................…9

3 - Conclusion…………………………………………………………….……………...13

4 - Recommendation……………………………………………......................................14

References……………………………………………………………………..…………15

**Executive Summary**

The rapid growth of the internet and the popularity of smart devices make web applications more and more important in people’s everyday lives. Nevertheless, traditional web applications are constrained to browsers. To access the application, the users have to open up a browser and get to the link through bookmark folders. In order to solve this problem, Progressive Web Application (PWA) is introduced to run web applications outside of the browser and provide a near-native app experience.

The report aims to analyze the advantages and disadvantages of PWA, based on its design, components, and performance.

It is concluded that PWAs are better at rendering speed and data usage. However, it is not widely supported in all browsers. It offers native app experience using much less device storage but doesn't have access to several device hardware due to the restriction of the browser.

**1 – Introduction**

The ubiquity of the internet gives rise to the popularity of web apps. Traditionally, web apps are run in the browser, either on a computer or on a mobile device. However, the new technology introduced by Google frees the application from the browser and provides a smoother user experience similar to a native app. The technology mentioned above is called a Progressive Web App (PWA).

Just as its name implies, a Progressive Web App is progressive in the ideal case. That is, the web app works for every user, no matter what device or browser the user uses. Besides, according to the official document (“Progressive Web Apps”, n.d.), there are three main advantages PWA has over traditional apps: reliable, fast and engaging.

This work report will introduce these advantages and discuss component-wise the way of achieving the goals. What’s more, analysis and comparison between PWAs and traditional apps will be conducted.

**2 - Analysis**

In this section, we will discuss some of the main characteristics of a PWA: progressive, reliable, fast and engaging, and how its components achieve these goals.

**2.1 - The Characteristics of a PWA**

- Progressive

One can consider a PWA as a web app out of the browser. Or more generally, a standalone app for the computer/tablet/mobile phone platform.

With that in mind, one can easily figure out the main characteristic of a PWA, which is progressive. That’s because a PWA is essentially a web app that makes HTTP calls and relies on web services, but without the restriction of the browser. Just like google.com is accessible for all platforms, a PWA is also easy to access, regardless of the device or browser. (However, this is just the ideal case, as not all browsers support PWA. This will be discussed further in the comparison section.)

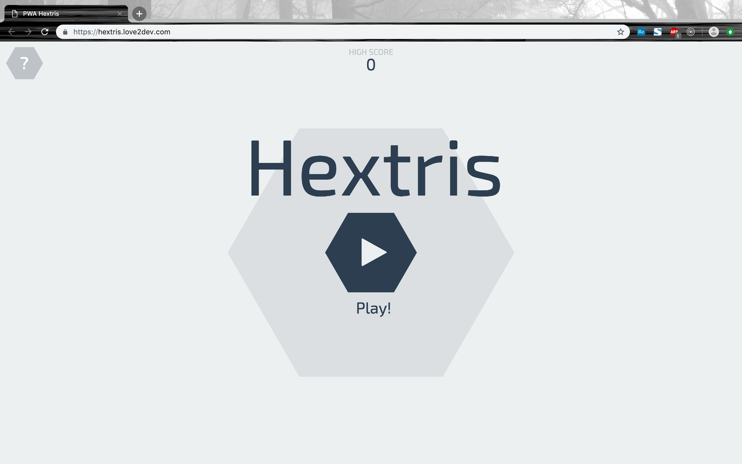
- Reliable

A PWA also has a special web worker, namely, the “Service Worker”. We will take a closer look at it in the next section as well. In brief, the Service Worker sits between the app and the server, and its logic can handle the app's HTTP requests in all kinds of network condition. In other words, a PWA will always open and operate without any problem, even if there is no network connection.

- Fast

As one can imagine, another advantage of web apps is their speed. Once the network calls are made and the connection is established, the webpage and the app itself will load. There is little or no local setting up time, as all data will come from the internet servers. In this regard, PWAs are even better, as the service worker can return the cached data when applicable.

- Engaging



Take the PWA mini-game “Hextris” as an example.

First of all, the game is accessible through its URL: https://hextris.love2dev.com/. Other than that, the user can also choose to “Install Hextris” in the browser page, after which an app will show up on the user’s computer, as shown in the screenshot on the right.

The app can be launched outside of the browser. It has its own window, without any browser-like interface. Besides, the “installation” is finished in one click, and it doesn’t require any “uninstall”, as it is essentially just a web app, like a bookmark. Therefore, deleting the icon suffices to remove the app from the computer.

In the next section, we will discuss how PWAs are designed to achieve these goals.

**2.2 - The Components of a PWA**

In order to make a web app work, it should have the backend service and the frontend interface. What’s more, in order to be progressive, reliable, fast and easy to launch, as discussed in the previous section, the PWA also introduces some special components, namely App Shell, Service Worker, and Manifest file.

**2.2.1 - App Shell**

App Shell is like the framework of the PWA. Let’s take the popular phone app Instagram as an example: no matter what the images or contents would be, the basic structure of the app is all the same. The images fill in the grid template, and the template is created following the layout design. Other than that, the same set of icons (like the gear icon) and the other static assets also remain the same for all users and all sessions.

Therefore, instead of sending requests to the server and get the same set of template and icons, we can separate the framework (App Shell) and the content. After the separation, whenever the user reloads the PWA, only the real content is fetched from the server, while the App Shell itself is loaded instantly from the local storage managed by the Service Worker. The idea here is similar to the browser cache, and the difference will be discussed in the “Comparison” section.

With the help of App Shell, a PWA can be fast.

**2.2.2 - Service Worker**

A Service Worker is an intermediate level between the app itself and the server. It monitors the network calls and manages the local storage. Note that there is a limit on the local storage (Osmani, 2019).

With the help of the Service Worker, the app will load instantly once launched, regardless of the network condition. That's because instead of making direct calls to the servers, a PWA dispatch network request through the Service Worker, and the latter will provide the optimal solution. Even when the device has no network connection at all, the Service Worker can still response the app with the cached data.

The PWA is hence reliable because of this mechanism.

In terms of progressiveness, let's take a look at this piece of code:

if ('serviceWorker' in navigator) {

navigator.serviceWorker

.register('../service-worker.js')

.then(function() { console.log('Service Worker Registered'); });

}

The concept of progressive enhancement is used here in the code. No matter it is registering the service worker, or fetching data from cache or network based on different connection situation, the code will always consider the fall back plan if the method won't work.

That is how a PWA become progressive.

**2.2.3 - Manifest**

A manifest is a JSON object that specifies the name, icon, and other metadata of a PWA. It provides information to the browser and tells it what the app should look like when it is installed on the device i.e. defines the look and structure of the app for offline experiences.

A sample Manifest file looks like this:

{

"name": "My Progressive Web App",

"icons": [

{

"src": "/images/icons-512.png",

"type": "image/png",

"sizes": "512x512"

}

],

"start\_url": "/test/?source=pwa",

"display": "standalone",

"scope": "/test/",

}

The manifest makes a PWA engaging.

**2.3 - A comparison between PWAs and native apps**

Essentially, PWAs are not new technologies. It is still based on the standards of web applications while presenting a better practice on how to make the application progressive.

The most important concept that PWA introduces is the Service Worker. One may think that the role a Service Worker plays is similar to the cache of the browser. However, they are not exactly the same. To be more specific, a Service Worker enhances the browser cache by making it possible to take control of all network calls the app makes.

Consider this "fetch" method for Service Worker:

self.addEventListener('fetch', function(event) {

event.respondWith(

caches.match(event.request)

.then(function(response) {

if (response) {

// Cache hit (or do whatever you want)

return response;

}

// Cache miss: pass the request to the server (or do whatever you want)

return fetch(event.request);

})

);

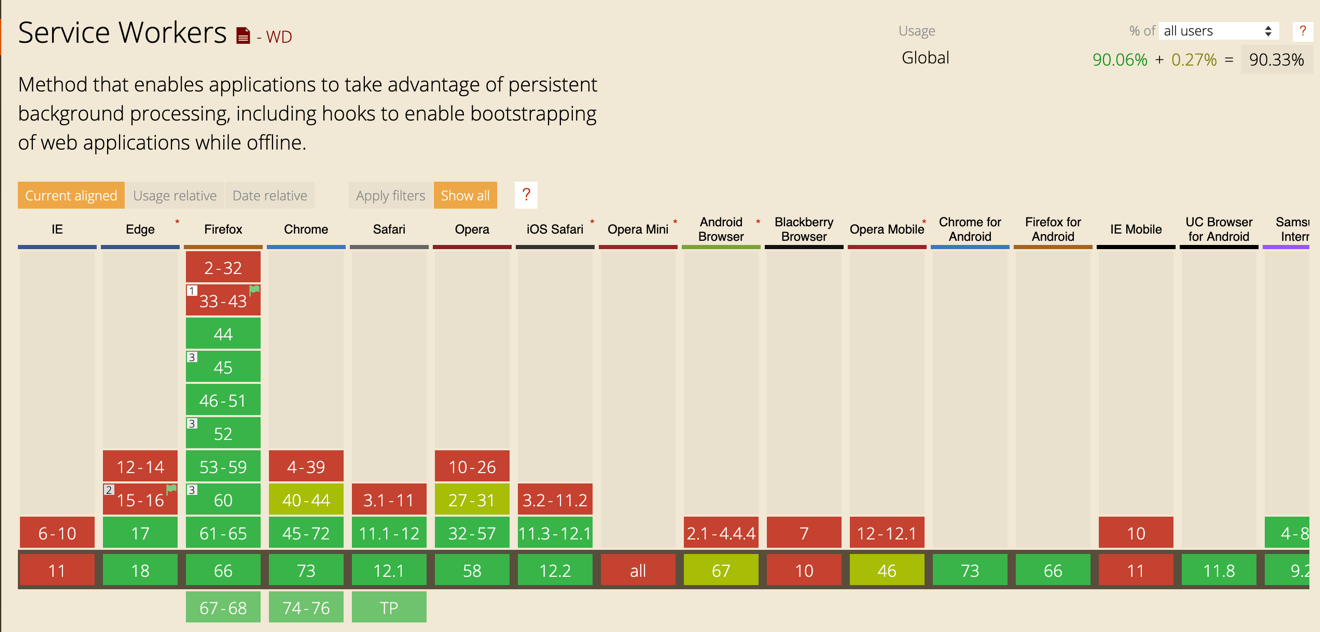
});

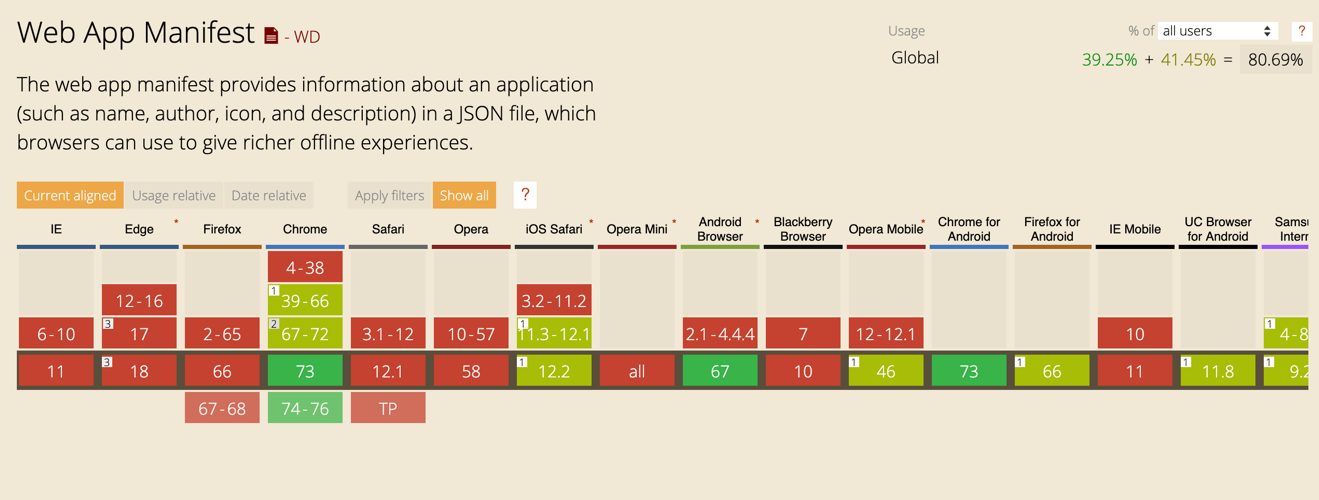
In this code snippet, the Service Worker return the response from cache on "fetch" event if the local storage holds the cached data. As per the code shown above, the Service Worker is acting the same as a regular browser cache. However, the Service Worker can also do something else that a regular cache can't do - for example, it is able to modify the request header and re-send it to somewhere else, or it can also hardcode a dummy response and send it back when the PWA is offline.

In a word, with the help of a Service Worker, we can actively respond to the requests made by the app. It gives the developer more freedom to handle different conditions.

On the other hand, while web apps work on all devices, let's also take a look at how different browsers support PWA, in terms of Service Worker and Manifest file. We will focus on these two, because the Service Worker provides a progressive rendering experience, and the Manifest makes the app installable.

As shown in the figure on the top (“Can I use Service Worker?”, n.d.), Service Worker is now widely supported by almost all mainstream browsers, except for IE (as always). That is, one can enjoy the PWA experience in almost all browsers. Even in IE, the code shown in section 2.2.2 makes it possible for the app to run like a regular web application.

According to the figure on the bottom (“Can I use Manifest?”, n.d.), the Manifest file is not fully supported by many browsers. Therefore, the user cannot get the one-click install feature unless they're using certain browsers.



As for the performance of PWA vs native app, let's take a look at Pinterest's statistics on their PWA (Osmani, 2017):

|  |  |  |
| --- | --- | --- |
|  | Mobile Site | PWA |
| First Paint | 5.6s | 1.8s |
| Time to Interactive | 3.9s | 0.6s |
| Size | Android 9.6MB  iOS 56MB | 150KB |

One can see in the chart that the performance is greatly improved when we get from the app to PWA.

**3 - Conclusion**

As written above, a Progressive Web Application is progressive, fast, reliable and engaging. The mechanism behind it is that a PWA separates the real-time content with the App Shell, so the size of the data needed from the server is reduced. The Service Worker controls all the HTTP requests, which makes the rendering flow fast and reliable. Last but not least, the Manifest provides the metadata of the PWA and makes it possible to install and run out of the browser.

The drawback of the PWA is that some of the components are not supported in all browsers. Besides, the size of the local storage space that a PWA can take is limited, which makes it not as free as a native application.

**4 - Recommendation**

This whole report is inspired by the dashboard web app we build in our team. Now, let’s apply the conclusion we drew in this work report to this app.

Since our app is a dashboard app that shows the real-time account information, it should guarantee that all data is valid and up to date. When there is slow or even no internet connection, instead of showing cached data or mock data returned by the Service Worker, it’s better that the app shows nothing, because stale data may confuse the advisors who are using our application.

In terms of caching the static assets, the browser cache should suffice to handle the loading process of images and icons we used in the model.

Therefore, my opinion is that our app doesn’t need to be a PWA.

**References:**

Can I use Manifest? (n.d.).

Retrieved April 18, 2019, from <https://caniuse.com/#search=manifest>

Can I use Service Worker? (n.d.).

Retrieved April 18, 2019, from <https://caniuse.com/#search=service%20worker>

Firtman, F. (2018). Progressive Web Apps on iOS are here.

Retrieved April 18, 2019, from <https://medium.com/@firt/progressive-web-apps-on-ios-are-here-d00430dee3a7>

Osmani, A. (2017). A Pinterest Progressive Web App Performance Case Study.

Retrieved April 18, 2019, from <https://medium.com/dev-channel/a-pinterest-progressive-web-app-performance-case-study-3bd6ed2e6154>

Osmani, A. (2019). Offline Storage for Progressive Web Apps. Retrieved April 18, 2019,

from <https://developers.google.com/web/fundamentals/instant-and-offline/web-storage/offline-for-pwa>

Progressive Web Apps. (n.d.). Retrieved April 18, 2019, from

<https://developers.google.com/web/progressive-web-apps/>