Bachelor's thesis

Degree programme in Information and Communications Technology

2019

Archana Karki

PROGRESSIVE WEB APPLICATIONS

- Powerful websites, functional as native mobile apps

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Degree programme in Information and Communication Technology

2019| 60 pages

Archana Karki

PROGRESSIVE WEB APPS

- Powerful websites, usable as native mobile apps

Current web platform and browsers are adequately powerful to support mobile and desktop applications. Application Program Interfaces (APIs) that supports the integration of mobile device and web browsers can execute features such as notifications, push messages, home screen icon, device camera and so on. The concept of progressive web apps is to make regular websites functional as mobile and desktop app, without compromising the user experience of traditionally used native apps. In order to understand the concept of Progressive Web Apps (PWAs), a functional app was created using HTML, CSS, JavaScript, along with the manifest file, service workers, and web APIs.

The primary objective of the thesis was to understand and implement the technologies of Progressive Web Applications (PWAs). Hence, a detailed study of the topic and development of a PWA was also carried out. The study showed that one PWA could serve the function of a website, mobile app and a desktop app efficiently. On one hand, the technology used for making a PWA is not too complicated for web developers using HTML, CSS, and JavaScript, and on the other hand, simple files can turn an existing HTTPS website into a fully functional app, saving the cost of developing a new native app.

KEYWORDS:

Web apps, web, native apps, hybrid app, service workers, app shell, cache API, device integration and notification API.

CONTENTS

FIGURES

PROGRAMS

LIST OF ABBREVIATIONS (OR) SYMBOLS

A2HS	Add to Home Screen
API	Application Program Interface
CSS	Cascading Style Sheets
DOM	Document Object Model
HTML	HyperText Markup Language
HTTPS	HyperText Transfer Protocol Secure
iOS	Operating system for mobile devices manufactured by Apple Inc
JSON	JavaScript Object Notations
NFC	Near Field Communication
OS	Operating System
PWA	Progressive Web Applications
SMS	Short Message Service
Src	Source
UC	It is a web browser developed by the Chinese mobile internet company UCWeb.
UI	User Interface
URI	Unique Resource Identifier
URL	Unique Resource Locator
W3C	World Wide Web Consortium
WebGL	Web Graphics Library
XML	eXtensible Markup Language

1 OVERVIEW

Progressive Web Apps (PWAs) are comparatively a newer technology than hybrid apps that aims to bridge the differences created in app development, due to mobile operating systems, such as Android from Google and iOS from Apple. This thesis aims to research technologies and implementation of PWAs. Three questions are the main focus of this thesis, and they are as follows:

1) What are PWAs?
2) What are the core technologies of PWAs?
3) How are these core technologies of PWAs implemented?

In today's context, the global market share of Android devices is approximately 85.9%, and that of iOS is 14%. Therefore, there is always a need for creating at least two versions of the same app to reach users of both operating system; one for Android and another for iOS. Platform difference has been a reason for an extra cost to customers and developers. (Statista, 2019a). Therefore, alternatives such as PWA, React Native, Flutter are being introduced to replace the issues of platform-difference. However, this thesis focuses on only PWA and its possible benefits. PWA is not compared with either React Native or Flutter in this thesis.

Alex Russell initially drafted the term and concept for PWAs in 2015. Hence, the idea of PWA’s relatively new. The resources and information used for the research are from Mozilla Developers Network's web docs, Google Developers' official documentation, documentation of Vaadin, Google I/O event's insights, a course on PWAs by Maximilian Schwarzmüller and other supplementary references such as statistics from Statista.

For this thesis, a PWA named ‘Share Moments’ is created and deployed on the internet. ‘Share Moments' is capable of operating in offline states, installing a home screen icon, sending push notifications, and using a camera to capture an image. Tools used in the process are as follows:

- Firebase from Google provides the real-time database, backend and HTTPS (HyperText Transfer Protocol Secure) services for the app.
- The browsers used for testing the app are Chrome for Android, Safari for iOS, Firefox for Android and Chrome for the desktop version.

- The responsive UI (User Interface) template of the app is from Google's Material design lite package. The starter template contained a floating
- action button, a sidebar, a navigation pane, and a header image container, which helped to focus more on learning concepts of PWAs than adjusting designs.

Additionally, Google Chrome currently supports desktop PWAs which allows a webpage to function as a desktop app as well. However, the details of desktop PWAs are beyond the scope of this thesis.

To sum up, this thesis has been divided into eight main chapters. Chapter 1 provides an overview of PWAs, chapter 2 includes a general introduction of PWAs and chapter 3 includes a brief description of various types of applications, other than PWAs. Similarly, Chapter 4, 5, 6 and 7 includes features of PWAS, core technologies of PWAs and its implementation respectively, and Chapter 8 concludes all those concepts learnt about PWAs.

2 INTRODUCTION OF PWAS

PWAs utilize the power of web technologies to deliver mobile and desktop apps. This chapter aims to address the first question mentioned in the overview section, so it covers the history, needs, and current possibilities of PWAs.

The terms “Progressive Open Web Apps” and “Progressive Apps” were introduced by Frances Berriman and Alex Russell in 2015. The word ‘app' is an abbreviated form of ‘applications,' hence, the terms are used interchangeably. (American Dialect Society, 2014a). The main features of PWAs are responsive design, safe data usage, the ability of installation from web browsers, secure sharing and use of native device features such as camera, geolocation, and push notifications.

In a simple definition, PWAs are regular websites at their core, that use few additional technologies such as service workers, web app manifest and web APIs (Application Program Interfaces) and progressively provide native-apps-like features. (Russell, 2015). The word "native" is synonymous to local, and it refers to someone or something that belongs to its place of origination. In computing, a native app is an application that can also access the device hardware such as cameras, memory or the geolocating sensors. Before the introduction of PWAs, native and hybrid apps were in use. PWAs is the enhanced concept that combines the best of these applications and the web as well.

2.1 Need for PWAs

The report of Statista for 2018 mentions that currently there are at least 2.1 billion smartphone users in the world, which is approximately 32.3% of the global population. (Statista, 2019). Similarly, each year approximately 1.56 billion units of smartphones are sold worldwide. (Statista, 2019b). Operating systems, specially designed for mobiles, are responsible for making hardware components of these smartphones functional. Mobile OS such as Android by Google and iOS by Apple are dominating the market share with 85.9% and 14% of the total smartphones in the world, respectively. Although a vast difference of such dominance maintains a fierce competition between the tech giants in innovations, it creates a substantial platform-specific difference while developing mobile apps or purchasing a smartphone as a developer and end-user.

Figure 1 offers support to the fact that the users of Android devices are massively high, but the brand loyalty of iOS users is also equally consistent over the years. Therefore, developing separate versions of an app for Android and iOS devices has been a challenge for developers because different mobile OS demands different programming language for development. Similarly, such apps are deployable in the app store of their OS only, either the Google Play store or iOS app store. Therefore, PWAs have been considered a new alternative for this issue since 2015.

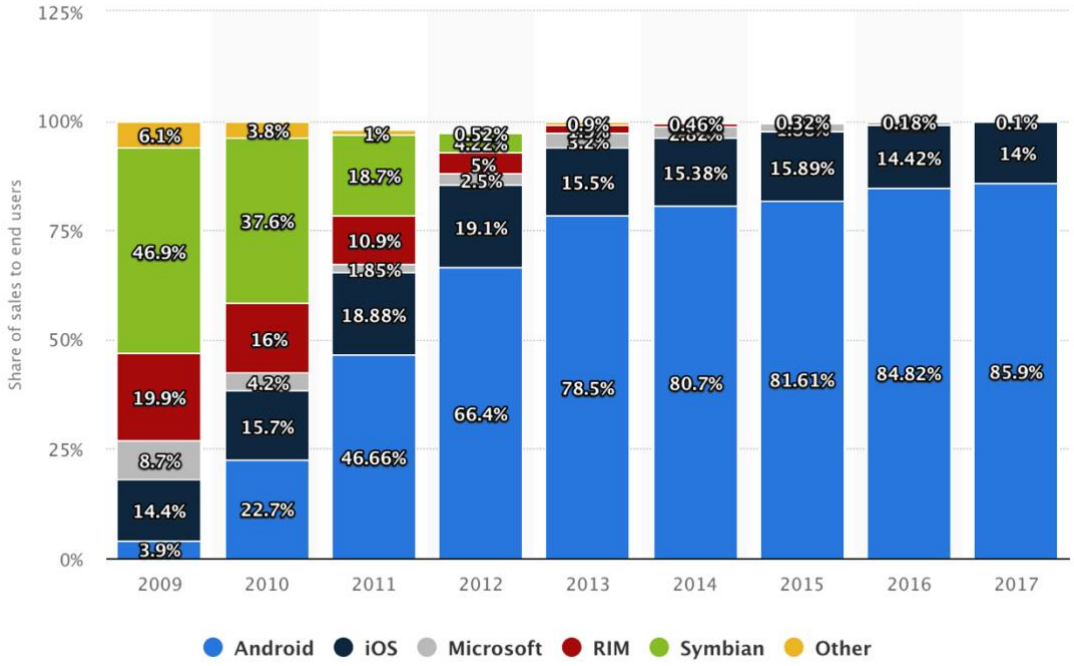

Figure 1. Global market share of various mobile operating system.

Unlike native app stores, PWAs are installable from the web browsers through an app install banners or Add to Home Screen (A2HS) feature. PWAs can also be developed with languages of the web, such as HTML (HyperText Markup Language), CSS (Cascading Style Sheets), and JavaScript. These languages are the basis of website development; hence, the web is a core fundamental of PWAs for development, deployment, and sharing.

Figure 2 depicts that currently, a web browser can support functions such as interactions with cameras, microphones, wireless surroundings, storage, sensors, inter-app communications, and payments. Previously, technologies used in native apps or separate plugins could only access the hardware of the device. However, in a modern browser, web APIs simplify the process and help to create custom functions. Generally, JavaScript handles the APIs. More information about web APIs can be found in Chapter 5.

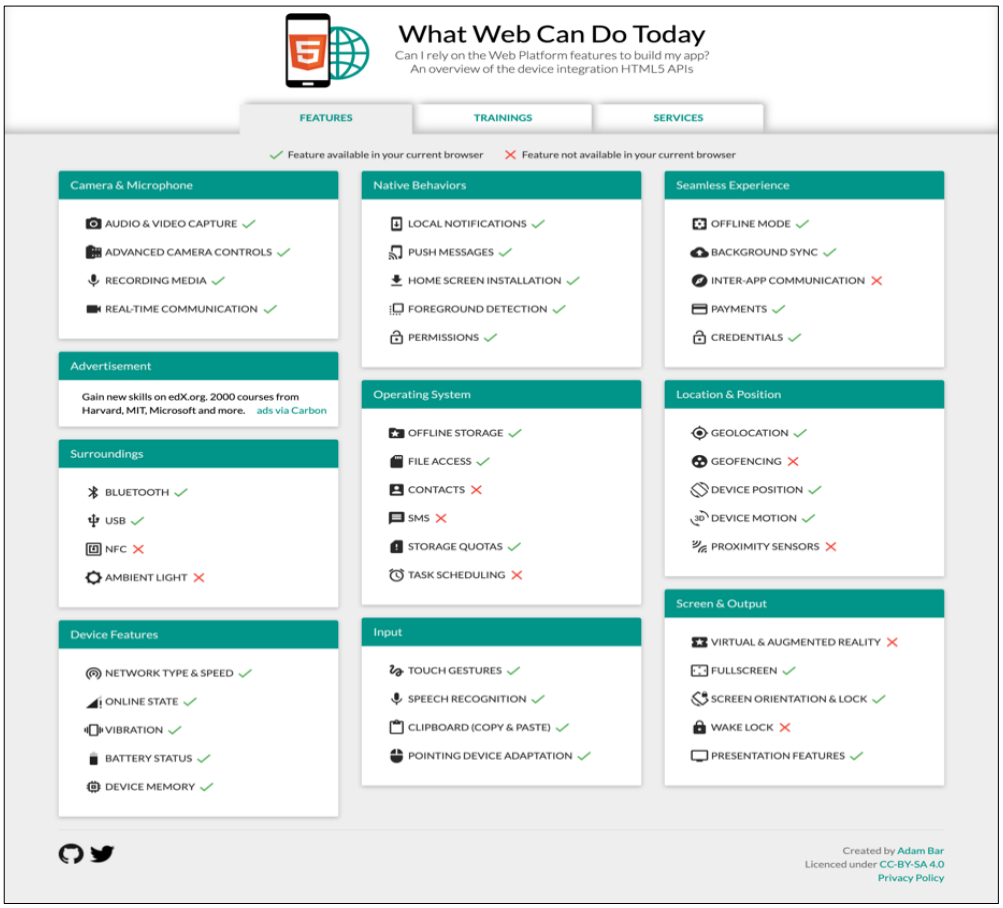

Figure 2. Features supported by web (as in 2019)

Source: https://whatwebcando.today/

PWAs are not yet the ultimate alternative or replacement for native apps. Previously, hybrid apps also started with features of web combinedly with native technologies. However, PWAs are more focused on making maximum utilization of the web. The following chapter contains a brief description of different types of apps.

3 TYPES OF APPLICATIONS

This chapter further justifies the need for PWAs by elaborating on the types of apps before PWA was introduced. Currently, there are three types of apps; native, hybrid and PWAs.

3.1 Native Applications

Native apps are platform-specific apps designed for one specific operating system such as Android, iOS and Windows. They are meant to utilize the device features such as cameras, memory, and geolocating sensors. Hence, functions such as push notifications, geolocations, offline contents, background synchronizations, and having an icon in the home screen work in native apps. Different operating systems have different requirements for an app to be native to their operating system. In terms of programming languages, Java is the preferred language for Android whereas iOS apps can be created using Swift and Objective-C. As a result, separate versions of apps are developed for Android, iOS, Windows, MacOS, and so on. (Dashtaki, 2017).

3.2 Hybrid Applications

Hybrid apps are aimed for cross-platform usage by combining the capabilities of native and web apps. Hybrid apps are programmed using HTML, CSS, and JavaScript and their deployment occurs through the native app store. Because of this, hybrid apps have to pass the review process of Apple, Google or similar app stores. Hybrid apps have a web view which is a simple browser within the app, in which the app runs. Although hybrid apps run on their web view, they are deployed through app stores only and not from the browser. (Dashtaki, 2017). Hence, hybrid apps do not comply with all requirements of web technology, so they can not be a part of the web ecosystem entirely.

3.3 PWAs

Contrary to hybrid apps, PWAs comply with the main specifications of the web in both development and deployment process. According to the definition of World Wide Web Consortium (W3C), "The web is an information space in which the items of interest, referred to as resources are identified by global identifiers called Uniform Resource Identifiers (URI)." (World Wide Web Consortium, 2004a).

According to this definition of W3C, the use of HTML, HTTPS, and URIs are the three main specifications for any web technology and PWAs fulfill these specifications. (World Wide Web Consortium, 2004b).

Web browsers, such as Google Chrome and Mozilla Firefox helps users to access this ecosystem of web and locate web pages through URIs, and as a rule, PWAs uses web browsers for deployment. (World Wide Web Consortium, 2004c). Due to this fact, PWAs are linkable, discoverable and installable. (Ingerbrigtsen, 2019). Moreover, the next chapter elaborates on more features of PWAs.

4 FEATURES OF PWAS

This chapter describes the essential elements of any PWAs, while it also points to features implemented in Share Moments app mentioned in chapter 1. PWAs should be reliable, responsive, engaging and secure for users, despite the type of device, network or browser support. (Vaadin, 2019b). Therefore, basic PWAs should pose the qualities mentioned below:

4.1 Reliability of app in sudden network disconnection

A PWA should be able to serve the basic and minimum content to the user regardless of network condition. A PWA can be entirely or partially offline-capable; however, it should not show an error page suddenly if the connection is lost. Hence, service workers, cache APIs, IndexedDB, are used to prevent a sudden breakdown of an app in ideal situations, and background synchronization is used to save the data to send it later, once the network connection is re-established. Share Moments app, uses cache API, indexedDB system and background synchronization.

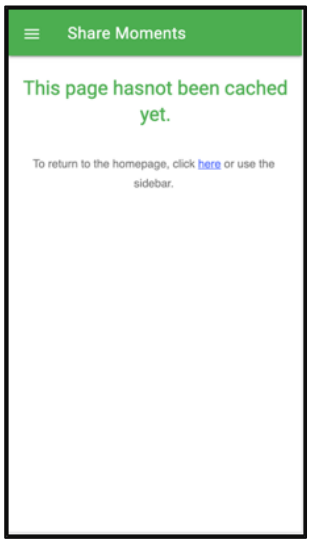

Figure 3. An example of a generic fallback page to prevent sudden disconnection

4.2 Responsive design for seamless user experience

A PWA needs to render native-app-like user experience; therefore an app must have a responsive design. For responsive web design, proper use of HTML and CSS is a must that automatically resizes the website according to the device's screen. The webpage should be able to adapt current screen sizes, but it should also be able to scale to future devices. Proper viewport, scalable images, media queries, and responsive web design frameworks can also simplify the process. Share Moments app uses Google's Material Design Lite template because it includes ready-to-use basic HTML, CSS and JavaScript file with responsive design.

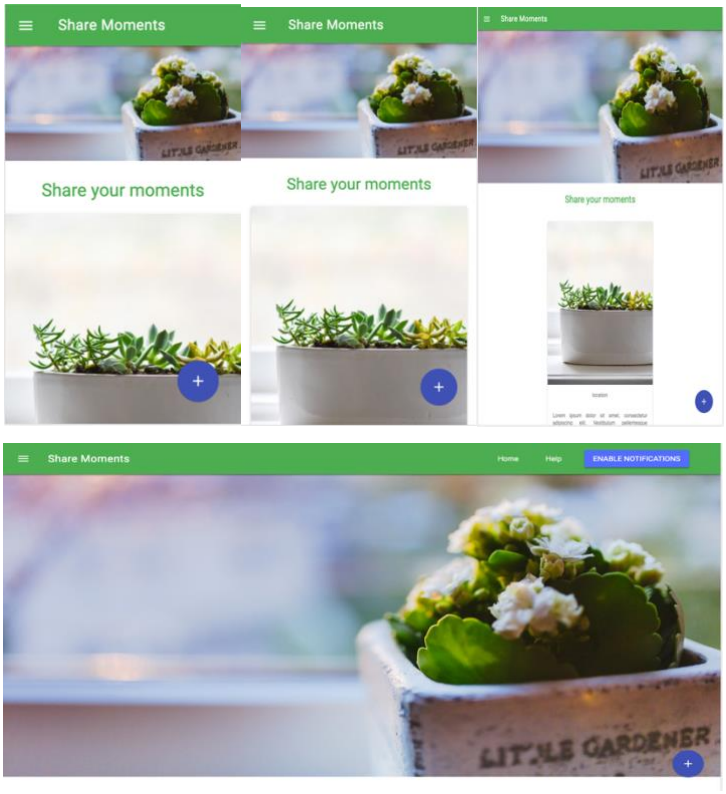

Figure 4. Responsive view of PWA in iPhone 5/SE, Galaxy S5, iPad and desktop.

4.3 The capability of notifying users by push notification and home screen icon

Push notifications and home screen icon are one of the primary tools of native apps that brings a user back to the app. A home screen icon can make the app accessible and push notification can alert the user. Simple web app manifest.json file generates a home screen icon and push API handles the push notifications.

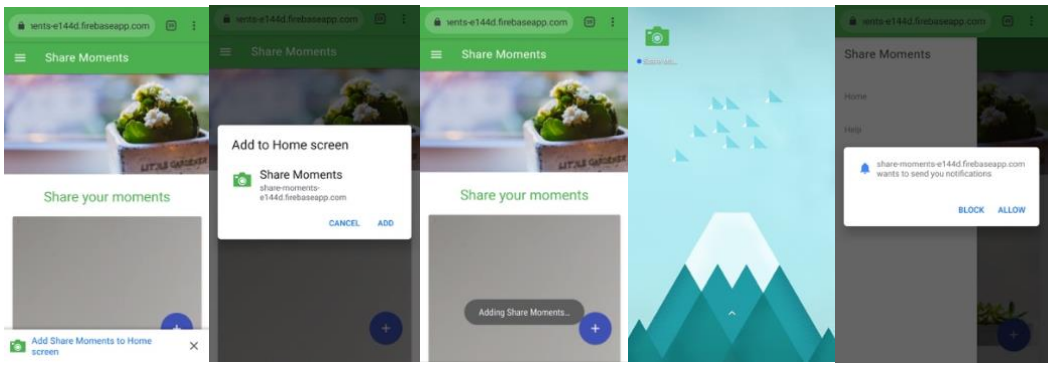

Figure 5. Examples of notification in PWA

4.4 Use of HTTPS for secure data exchange

PWAs are made safe with the use of HTTPS. Although, PWAs do not undergo the security check of a native app store they should comply with safety standards of HTTPS, which encrypts data before its transmission over the web. HTTPS is the basic necessity of a website to be usable as PWAs, so it is secure and safe by default. (Vaadin, 2019).

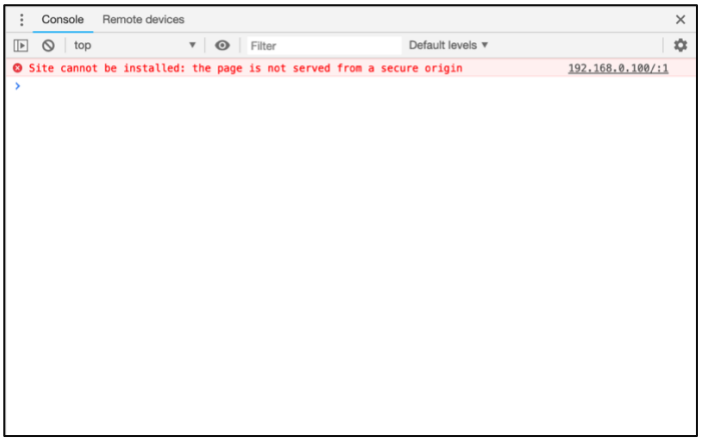

Figure 6. An example showing PWA cannot run on a website not using HTTPS

The browser-based requirement for PWAs

PWAs operates through web browsers so browsers can have additional requirements for a PWA. (Google Developers, 2019a). Google Chrome is used as the primary browser for Share Moments app to debug and to check its performance. Google Chrome's Lighthouse tool is used to monitor criteria met by the PWA.

Lighthouse is an automated tool of Google that audits performance, accessibility and PWAs qualities of the webpage. It generates a report and also provides feedback on possible improvements.

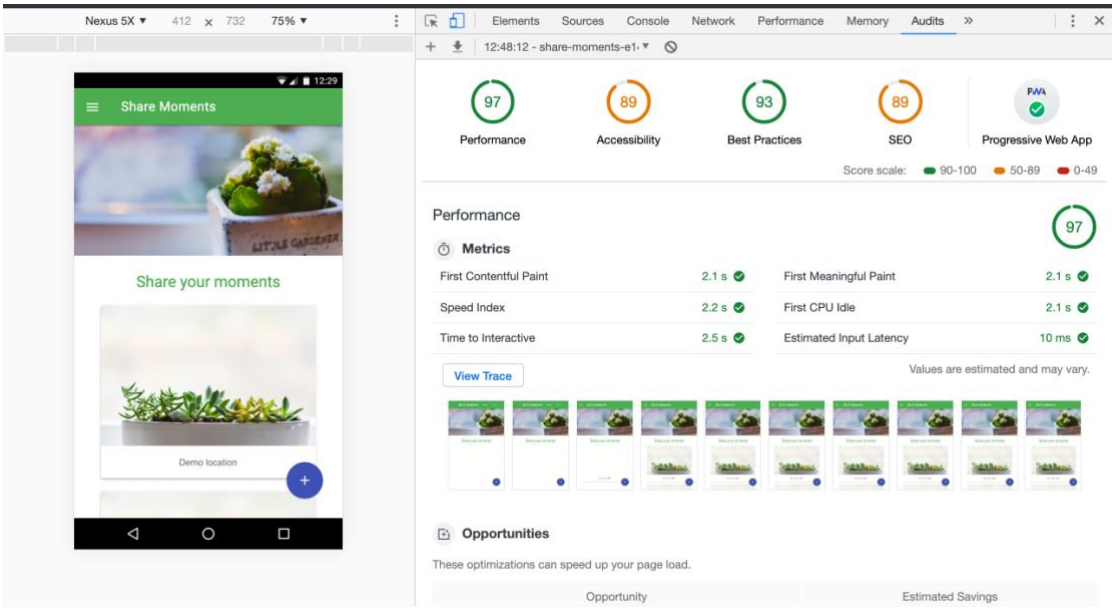

Figure 7. Audit report of Share moment app with Lighthouse

Source: https://share-moments-e144d.firebaseapp.com/

In conclusion, this chapter solely concentrated on the features of PWAs, so the next section highlights the technologies used in PWAs that helps to provide experience as mentioned earlier.

5 CORE TECHNOLOGIES OF PWAS

This chapter covers the technologies used in PWAs that helps to provide services mentioned in the previous chapter. All browsers might not support these technologies; so, if the browser or device does not support such features, the browser ignores unsupportive technologies, and the website runs as usual. On the other hand, if the browser supports the technology, then the intended user interface executes. This chapter includes a brief description of three core technologies of PWAs, which are listed as follows:

- Web app manifest file,
- Service workers and
- Web APIs

5.1 Web App Manifest file

Web app manifest file is a JSON (JavaScript Object Notations) file that makes an app installable from the browser to the device's home screen. Unlike the traditional way of installing apps from the app store, it provides similar capabilities through the "Add to Home screen" option of a browser. A manifest file requires information such as name, short name, icons, background-color, display, scope and theme color to customize the installation. A detailed implementation of the manifest file for the Share moment app is in chapter 5.

Program 1. An example of code from the manifest.json file

```
{   "name" : "Share Moments",
    "short_name" : "Share Moments",
    "start_url" : "/index.html",
    "theme_color" : "#4CAF50",
    "icons" : [{
        "src":"/src/images/icons/app-icon-48x48.png",
        "type":"image/png",
        "sizes" : "48x48"
}]
```

Figure 8 depicts that web app manifest is currently functional with Chrome browser, iOS Safari, Opera Mobile, Chrome for Android, Firefox for Android, UC browser for Android, Samsung Internet, QQ Browser and Baidu Browser while it is in-development in FireFox and WebKit. However, due to the large market share of Google Chrome, still 77.64% of global users can use it, and the numbers of users can increase in the coming years. It is also worth noting that, browsers which do not support web app manifest ignores this information and the website runs without a problem. Hence, the apps are "Great for all, and awesome for most." (Vaadin, 2019c).

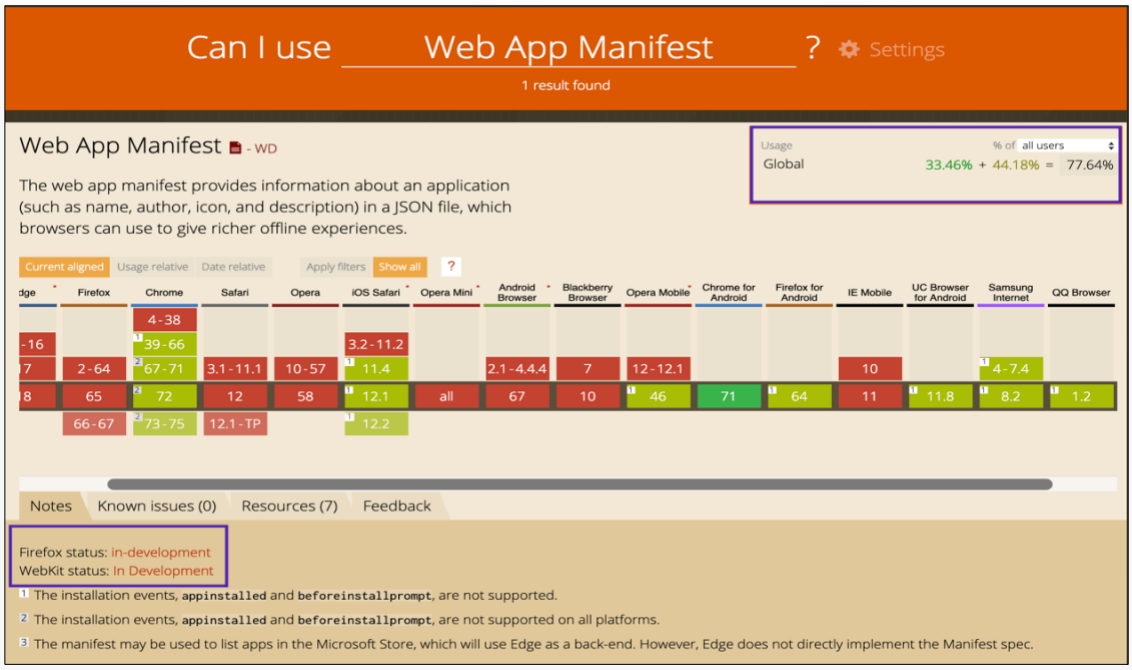

Figure 8. Different browsers supporting Web App Manifest file and its global reach.

Source: https://caniuse.com/#search=Web%20App%20Manifest

5.2 Service Workers

Service workers are the second essential building blocks of PWAs that work behind the scene. It is a JavaScript file that runs in the background of an app, makes the app offline-ready, allows push notifications and handles background synchronization. Service workers carry much power to intercept requests; hence service workers only work on the website that is served over HTTPS, so that the request and response are encrypted and uses a safe host. (Vaadin, 2019d).

Service workers are at work, even if the app closes, browser tab changes or the device is not in use. It handles the caching of a file to use in times of poor network connection. Likewise, with background synchronization, service workers update the webpage contents as soon as the internet connection is re-established. Service workers are independent of both the browser and the app, which makes push notifications possible with service workers.

Regular JavaScript file is attached to an HTML page, and it executes in a single thread. Even if an HTML page contains several JavaScript files, the thread of execution can be only one. Such regular JavaScript files can access the DOM (Document Object Model) of the website and change styles and divisions of the webpage based on mouse clicks, keypress, change or hover events. DOM is an interface that connects web pages and scripts so that web pages are modifiable with programming languages.

Service workers are also a JavaScript file, and it runs in a single thread; however, its thread of execution is different from the standard thread of execution. It is separate from the regular thread, and instead of DOM, it accesses background events. Hence, although service worker is linked to an HTML page as a standard JavaScript file, it is not coupled with that HTML page. It only manages pages within its scope. In this example, the service worker is in the root folder of Share Moments app, so it applies to all page of the app. (Schwarzmüller, Maximilian, 2019).

Service workers work by reacting to different sets of events than mouse clicks, keypress, and hover. Events can occur from JavaScript files, HTML codes or the servers. Service workers stay in the background, waits for any incoming events and reacts accordingly.

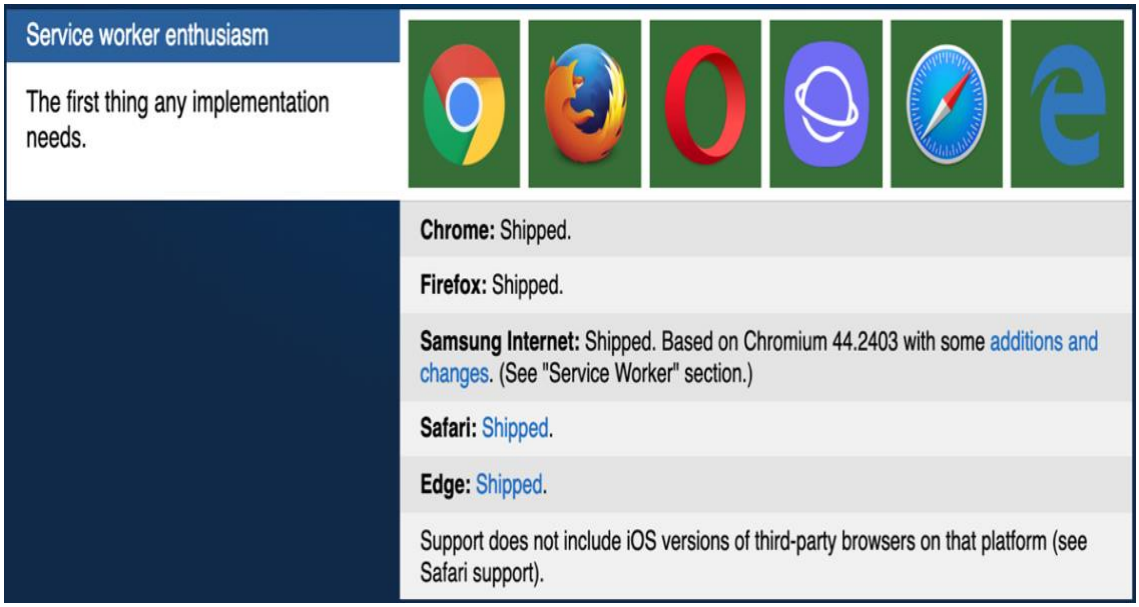

Figure 9. List of browsers that support Service Workers

Source: https://jakearchibald.github.io/isserviceworkerready/index.html#moar

5.3 Web APIs

APIs are the syntax that allows one application to communicate with other applications through programming languages. (MDN web docs, 2019a). Web APIs are designed for the server and the browser. It allows a developer to extract complex information or functionality from other apps and helps to create custom functions, by using fewer syntax. Web APIs use JavaScript for communication. The structure of data and functionality of web APIs are similar to the structure of properties and methods in JavaScript objects. Hence, the functions of an API highly resembles JavaScript objects. Some notable web APIs used in Share Moments app are listed below, as per their functions:

- **Cache API**
  Caching refers to storing (something) in a hidden or secure place for safety or concealment. For a PWA, Cache API provides a mechanism to store the minimum set of required assets from the website so that the app can be offline-ready.

- **Canvas and WebGL (Web Graphics Library) API**
  Canvas and WebGL API is used to draw 2d and 3d shapes, capture images and manipulate graphics.

- **DOM API**
  DOM API is capable of manipulating data rendered in the browser which includes manipulating styles of existing elements, adding new divisions, appending child element for containers and adding interactive components in the webpage.

- **Google Maps API**
  Google Maps API is an example of third party API that helps to send a specific query to Google or other sites and return information to customize maps in a website by using events, controls, and style. Other examples of third-party APIs are Twitter API, Facebook API, and Youtube API.

- **HTMLMediaElement and Media Streams API**
  Example of HTMLMediaElement are audio, video, source, track element that combines with media streams API to provides access to the hardware of the device. As a result, media stream API allows customization of multimedia such as audio and video. It also displays caption along with video and grabs an image from the web camera.

- **Notification API and Push API**
  Notification API or Push API is another example of device APIs that helps to sends a notification to the user as soon as any messages are detected from the server.

- **Promise API**
  Promise API is used to handle the callbacks and errors of asynchronous or unsequential operations. Promises are used to prevent the confusion created by additional callback parameters in complex functions. (Lindesay, 2019). For a PWA, many functions tend to be asynchronous, and Promise API makes the process more manageable.

- **Web Storage or IndexedDB API**
  Web Storage or IndexedDB API interacts and manages the client-side storage of the browser, to avoid sudden breakdown of the app in an uncertain network connection. In a PWA, these API offers additional support to cache API and makes an app offline ready.

- **XMLHttpRequest (eXtensible Markup Language) and Fetch API**
  XMLHttpRequest and Fetch API helps in fetching data from the server and instantly update the webpage without a need of reloading the entire page.

In conclusion, web app manifest file, service workers and web APIs are the most crucial technologies of PWA that provides easy access, background processing capabilities and offline storage respectively. Next chapter includes its implementation thoroughly.

6 IMPLEMENTATION OF A PWA

Any core technologies mentioned in the previous chapter can be used or ignored in a PWA as per the convenience. Using all core technologies enhance the user experience, whereas not including either of the manifest file, service workers or web APIs does not break the web page, and the general functionality of the web page works without any hindrance. This chapter aims to cover the implementation of technologies for a PWA and Share moments app also follows a similar process. The steps are covered in the sub-chapters as follows:

- Adding a web app manifest file
- Adding service workers file
- Adding required web APIs in service workers

6.1 Adding a web app manifest file

Firstly, a web app manifest file is added to the root directory of an app as manifest.json or filename.webmanifest.(MDN Web Docs, 2019a). Usually, .json is the appropriate extensions for the browsers.

Figure 10. Root web folder of Share Moments app

Secondly, the app manifest file needs to be linked to all HTML pages of an app, by using the HTML5 link tag. The link tag notifies browser about the associated manifest file and its location. (Google Developers, 2019a).

Program 2. An example of code for linking web app manifest file in the HTML page

```
<link rel = "manifest" href="/manifest.json">
```

Finally, the properties need to be added in a manifest.json file in a format of a key/value pair. (W3Schools, 2019). Other properties used in web app manifest file are explained as follows:

- **Name**

Name property refers to the name of an app. It is written in a plain text and is visible on the splash screen of an app along with the logo.

Program 3. An example of code for defining the name of a PWA

```
"name": "Share Moments."
```

- **Short name**

The short name refers to the shorter name of an app. It is visible below the icon on the user's home screen, launcher or other limited areas of the device.

Program 4. An example of code for defining the short name of a PWA

```
"short_name": "Share Moments”
```

- **Start URL (Unique Resource Locator)**

Start URL defines a pathname to the first page; a user can see when s/he taps the home screen icon. Defining a specific path can prevent users from landing in an unrelated page. In Share Moments app, “/index.html” is the pathname to the index file.

Program 5. An example of code for defining the index page for a PWA

```
"start_url": "/index.html"
```

- **Scope**

Scope refers to pages that can render PWA experiences to users. Hence, it is possible to include or skip some pages from PWA capabilities. However, the starting URL of the previous point must be within the scope to be defined.

Program 6. An example of code for defining the scope of a PWA

```
"scope":"."
```

- **Display**

Display property detects how an app should look like in an app-mode. There are four different modes for display which are fullscreen, standalone, minimal-UI and browser. The app takes entire display area in standalone mode except for the icons bar for network status, battery and hours.

Program 7. An example of code for defining the display mode for a PWA

```
```

```
    "display": "standalone"
```

- **Background color**

The value of background color is visible in the splash screen of the app. Generally, the splash screen appears after the app is launched but before the first page is loaded. It generally consists of logos, color or text that represents the app.

Program 8. An example of code for defining the background color for a PWA

```
    "background_color": "#4CAF50"
```

Figure 11. Use of background color on the splash screen.

- **Theme color**

This color is displayed on the toolbar and in task switchers. Theme color also enhances the native-like app experience in the standalone display mode.

Program 9. An example of code for defining the theme color of a PWA

```
    "theme_color": "#4285f4"
```

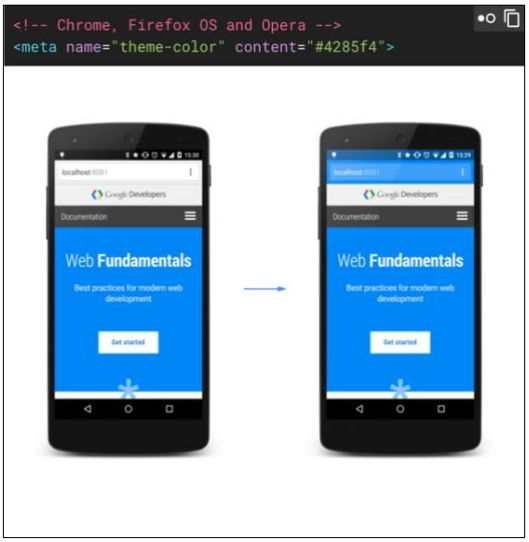

Figure 12. Meta-theme color tag and effect of theme color in an app

Source: https://developers.google.com/web/fundamentals/design-and-ux/browser-customization/imgs/theme-color.png

- **Description**

The description provides additional information about the app, and it is used by the browser while pinning the website or adding it to favorites.

Program 10. An example of code for describing a PWA

```
"description": "A photo-sharing app."
```

- **Dir**

Dir is a shortened form of direction. It represents the direction of name, short_name, description, and language. Distinguishing the display of left-to-right or right-to-left languages is helpful. Value of this property can be ltr(left-to-right), RTL(right-to-left) and auto.

Program 11. An example of code for specifying the direction of letters for a PWA

```
"dir": "ltr"
```

Left-to-right languages	Right-to-left languages
Latin	Arabic
Modern Greek	Aramaic
Cyrillic	Azeri
Indic and Southeast Asian	Hebrew
Europe	Persian/Farsi
North and South America	Urdu

Figure 13. List of languages by the direction of letters.

Source: https://www.andiamo.co.uk/resources/right-to-left-languages/

- **Lang**

This property specifies the primary or default language of an app. It does not carry a more profound level of function, apart from providing information. However, it is one of the critical metadata.

Program 12. An example of code defining the language of a PWA

```
"lang": "en-US"
```

- **Orientation**

Orientation refers to the position or direction of an object in respect to its surrounding. This property should be considered to suit the comfort values of the user. Natural, landscape, landscape-primary, landscape-secondary, portrait, portrait-primary and portrait-secondary are the values for orientation.

Program 13. An example of code defining the orientation for a PWA

```
      "orientation": "portrait-primary"
```

- **Icons**

Icons include several JSON objects where each of the objects specifies an src (source), type and sizes of an icon. Browsers itself chooses the size of the most appropriate icons for a given device.

Program 14. An example of code for defining icons for the app

```
"icons": [{
  "src": "/images/icons-192.png",
  "type": "image/png",
  "sizes": "192x192"
}]
```

At least, browsers require 192 x 192 pixels and 512 x 512 pixels icon. For more precise scaling, icons of 48 x 48, 72 x72, 96x96, 128x128, 144x144, 256 x256 and 512 x 512 pixels are suitable.

- **Related applications**

This property lists any alternative native apps developed for the website such as Android or iOS app. However, this is not for listing any alternative web app. This property requires a platform, URL, and id of each app.

Program 15. An example of code for defining related native apps of a PWA

```
"related_apps": [{
"platform": "play",
"url":"https://play.google.com/store/apps/details?id=com.example
.app1",
}]
```

6.2 Adding service workers file

A file for service workers is JavaScript file that resides in the root directory of an app; however, it has to pass three different phases before being able to function successfully. These three stages formulate a lifecycle of a service worker, and they are as follows:

- Registration
- Installation and
- Activation

The first stage of service workers' lifecycle is its registration in the browser.

In program 16, the word navigator refers to the browser and the code evaluates if the browser supports service worker API. The code checks if the browser supports service worker API. If the response is accurate, the browser registers the service worker file.

Program 16. An example of code for registering the service worker.

```
//Registration of service workers
if("serviceWorker" in navigator){
navigator.serviceWorker.register("/sw.js").then(function(){
console.log("Service worker is registered");
});}
```

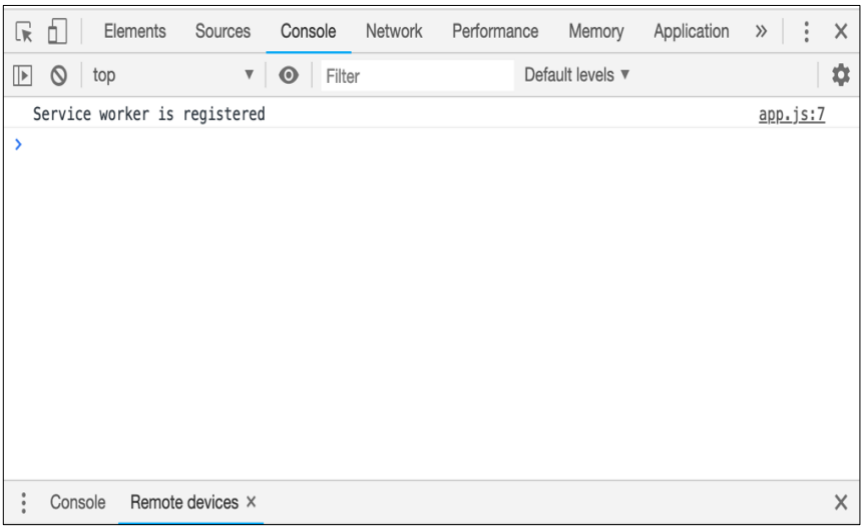

Figure 14. An example of service worker registration

Secondly, the browser needs to install the service worker. Registration of service worker does not necessarily mean that it is ready to work.

In program 17, the word ‘self' refers to the request of service worker API to access background processes. A service worker cannot access to DOM, so events such as click, keypress are not usable in it, instead a separate set of events are used, such as ‘install,' ‘activate' and ‘fetch.' These events can succeed, fail or remain pending, so a callback function is also executed. The callback function for installation can return promises that can execute cache API. Chapter 4 deals on core technologies of PWAs includes the details about promises and cache APIs.

Program 17. An example of code for installing the service worker.

```
//Installing and activating the service worker
self.addEventListener("install", function(event){
console.log("[Service Worker] Installing Service Worker ....",
event);
/*---- In later point, Cache APIs will be included here to provide
offline access of the app ------ */
});
```

Lastly, the service worker should be activated for performing any action.

Program 18. An example of code for activating the service worker.

```
//Activating the service worker
self.addEventListener("activate", function(event){
console.log("[Service Worker] Activating Service Worker .....",
event);
/*--In later point waitUntil method will be used to remove old
cache before updating new service worker--*/
return self.clients.claim();
});
```

However, the service worker might not necessarily activate after this function. It activates, only after every open tab of an app is closed, to avoid new updates while the app is in use. Therefore, closing all tabs of that app is vital for an updated service worker. Service workers impact pages defined in the scope property of the manifest.json file. The syntax for activation is similar to the syntax of installing service workers; however additional information included in the return statement ensures that service worker activates correctly.

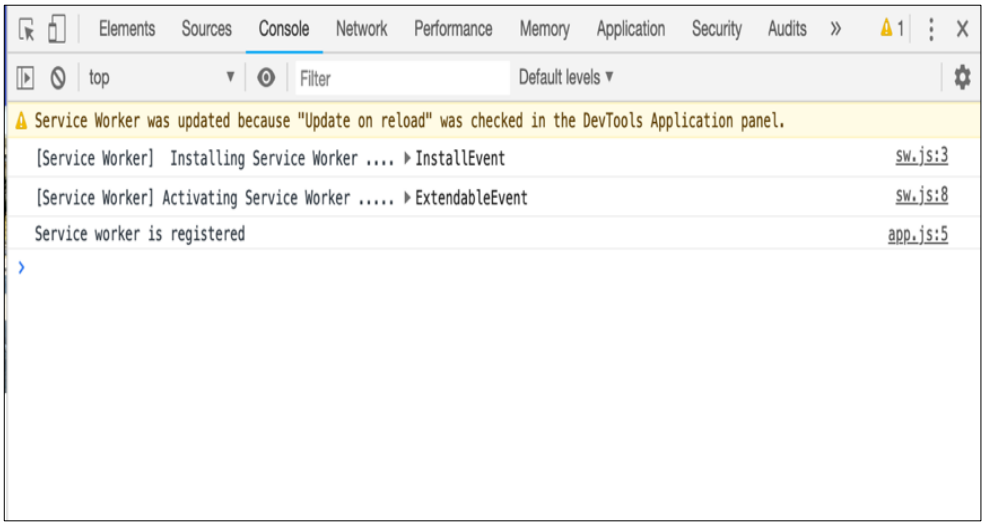

Figure 15. An example of a log of service worker installation

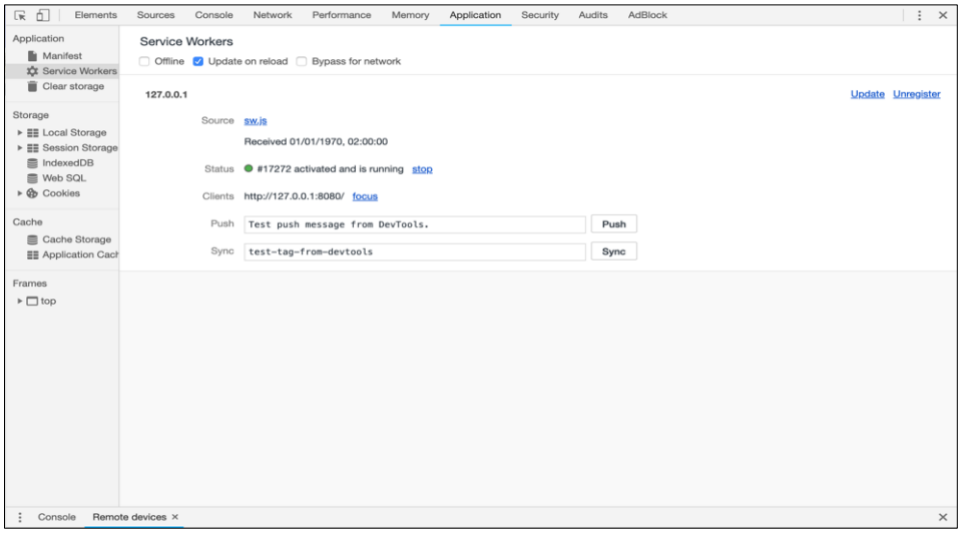

Figure 16. An example of a service worker, activated and running

After the service workers are registered, installed and activated, service workers turn into an idle mode or terminate, unless it detects any events. Such idle state does not refer to uninstallation or cancellation of service workers. Service workers get active and perform defined actions, only if it detects any appropriate events from the server. The figure mentioned below demonstrates the lifecycle of a service worker.

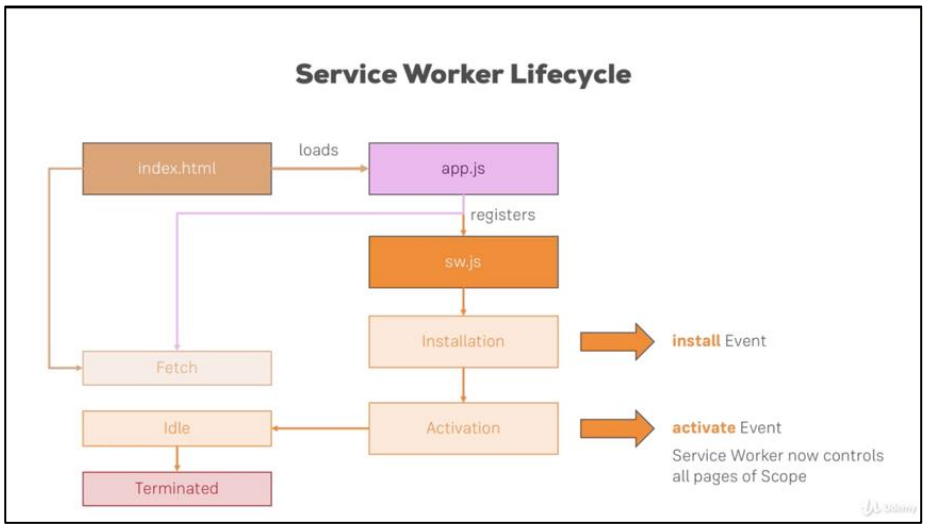

Figure 17. Lifecycle fo service workers

Source: https://www.udemy.com/progressive-web-app-pwa-the-complete-guide/learn/v4/t/lecture/7810136?start=0

6.3 Adding required web APIs in service workers

Modern day web APIs allows a developer to communicate with other apps and use them to make custom functionalities. Chapter 4 thoroughly discusses the concept of web APIs, whereas the following paragraphs show its implementation only, in any PWA or Share Moments app for an example.

- DOM API
- Promise API
- XMLHttpRequest and Fetch API
- Cache API
- Web Storage or IndexedDB API
- Notification API or Vibration API
- Canvas and WebGL API
- HTMLMediaElement and Web Audio API
- Google Maps API

DOM API

DOM API has been used to create HTML division, append sub-division to it and to add style for new posts by using JavaScript syntax instead of HTML tags. It is added in the JavaScript file handling client-side functions.

Program 19. An example of code that structures each new post division.

```
var shareImageButton = document.querySelector('#share-image-
button');
var createPostArea = document.querySelector('#create-post');
function createCard(data) {
//Card div created
var cardWrapper = document.createElement('div');
// class name given to card div
cardWrapper.className = 'shared-moment-card mdl-card mdl-shadow-
-2dp';
//CardTitle div created
var cardTitle = document.createElement('div');
```

```
//CardTitle div is added in cadWrapper division
cardWrapper.appendChild(cardTitle);
```

Promise API

Promise API has been used in Share Moments app to handle unsequential operations, that can have an uncertain result. Promise API was used to prevent callback hell, in case several functions need to be executed inside of one another. Promise uses a chain method with .then() method and .catch() method to simplify the process.

Program 20. An example code to create a new promise and use it (Archibald, 2019a).

```
//Creates Promise
var promise = new Promise(function(resolve, reject) {
if (/* Execution successful*/) {
resolve("Succesful");
}
else {
reject("Error occurred");
}
});
//Uses promise
promise.then(function(result) {
  console.log(result); // Successfull
}, function(err) {
  console.log(err); // Error occurred
});
```

Most of the APIs such as fetch API, cache API, and MediaServices API returns a promise, so promises do not need to be created manually.

Program 21. An example of promise in media device API

```
navigator.mediaDevices.getUserMedia({video : true})
//if user has given persission for camera access
.then(function(stream){
videoPlayer.srcObject = stream;
videoPlayer.style.display = 'block';
})
.catch(function(err){
//if user have declined the persission for camera access
imagePickerArea.style.display = 'block';
});
```

XMLHttpRequest and Fetch API

Both XMLHttpRequest and Fetch API can be used for fetching resources from a server, updating a web page without reloading it and for sending data to a server in the background. However, Share moments app uses fetch API instead of new XMLHttpRequest due to its simplicity with promises.

Program 22. An example of making GET request using fetch API for Share Moments

```
var url = 'URL of a json file was used here”;
var networkDataReceived = false;
//This is a GET request that fetches the data
fetch(url).then(function(res) {
return res.json(); //This is the response received
}).then(function(data) {
//Response received is used to compare with networkData variable
networkDataReceived = true;
console.log('From web', data);
var dataArray = [];
for (var key in data) {
```

```
dataArray.push(data[key]); }
updateUI(dataArray); });
```

Cache API

Share Moments app uses cache API to store or cache the static and dynamic assets. Caching provides an offline capability to the PWA. Cache API is usable in the regular JavaScript file and service workers, however, service workers file is used for the PWA to handle the background processes.

Program 23. An example of cache API used to cache the static assets of the webpage

```
var STATIC_FILES = ['/', '/index.html', '/offline.html',
'https://fonts.googleapis.com/css?family=Roboto:400,700'];
//Installing service worker
self.addEventListener("install", function(event){
console.log("[Service Worker] Installing Service Worker ....",
event);
//Cache API
event.waitUntil(
caches.open(CACHE_STATIC_NAME)
.then(function(cache){
console.log("[Service Worker ] Precaching App Shell");
cache.addAll(STATIC_FILES);
}))
});
```

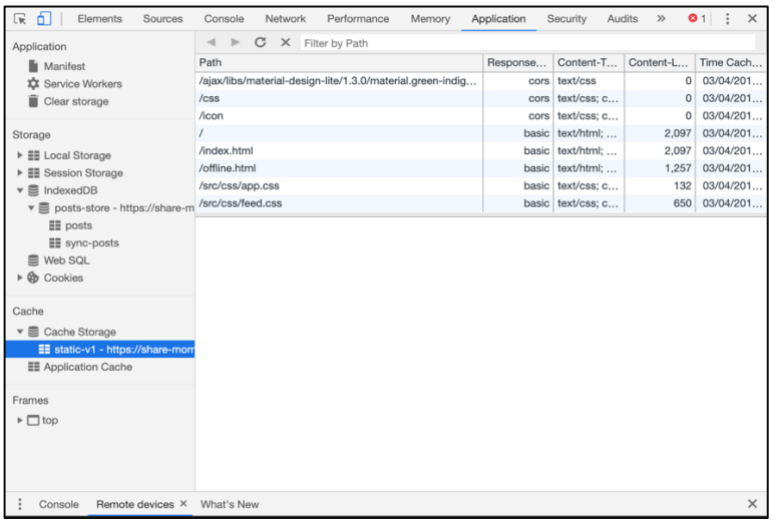

Figure 18. An example of cache storage in a browser that contains cached assets

Web Storage or IndexedDB API

IndexedDB is a NoSQL storage system that allows storing of files in the user's browser. (Google Developers, 2019b). It is useful for storing more significant amounts of structured data compared to cache storage. IndexedDB API has been implemented in Share moments to save any unsent post in an offline state, and to send it later internet connection is re-established.

Program 24. An example of code to create and populate indexedDB storage

```
//Checking support for indexedDB
if ('indexedDB' in window) {
readAllData('posts').then(function(data) {
if (!networkDataReceived) {
//remaining codes are in next page
//remaining codes are from previous page
console.log('From cache', data);
updateUI(data);
}});
}
```

```
//Opening indexedDB database
var dbPromise = idb.open('posts-store', 1, function (db) {
if (!db.objectStoreNames.contains('posts')) {
//Adding object stores or sub divisions in indexedDB
db.createObjectStore('posts', {keyPath: 'id'});}
if (!db.objectStoreNames.contains('sync-posts')) {
db.createObjectStore('sync-posts', {keyPath: 'id'});
}});

//Adding data to the object store
function readAllData(st) {
return dbPromise.then(function(db) {
var tx = db.transaction(st, 'readonly');
var store = tx.objectStore(st);
return store.getAll();
});
}
```

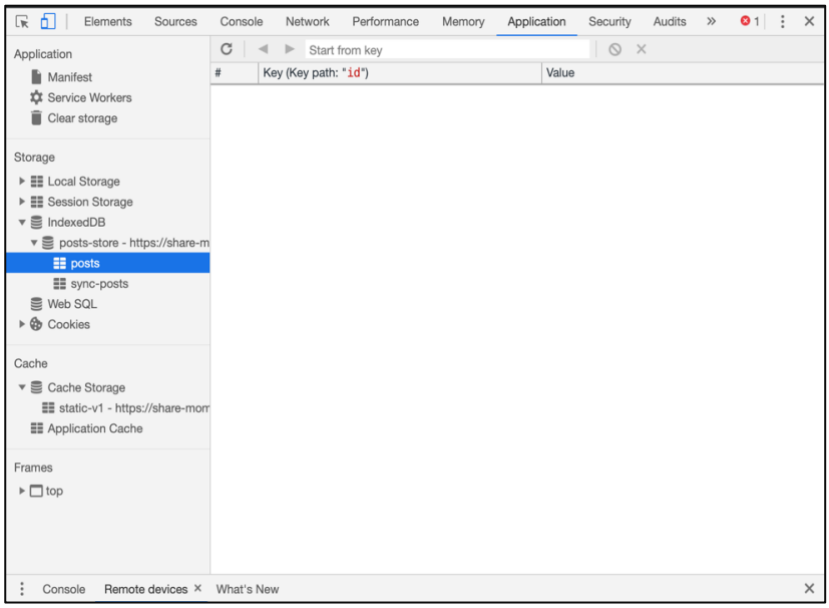

Figure 19. An example of IndexedDb storage in a browser

Notification API and Push API

The Notifications API allows web pages to send notifications to users by customizing the display of system notifications, based on the permission of the user. (MDN web docs, 2019b). Similarly, push API gives an ability to a PWA to receive messages from the server, even when the user has switched the tabs or moved to a different app. In Share Moments app, notifications API is used to ask permission for the camera, location, whereas push API is used to send push messages if the user has subscribed for notifications.

Program 25. An example of code used for sending notifications on push event

```
//Checking support for service worker
if('serviceWorker' in navigator){
//Customising notification message
var options = {
body: 'You successfully subscribed to the notification
service!',
icon : '/src/images/icons/app-icon-96x96.png',
image : 'src/images/plant1.jpg
renotify : true,
actions: [{ action : 'confirm', title : 'Okay', icon :
'/src/images/icons/app-icon-96x96.png'}, { action : 'cancel',
title : 'Cancel', icon : '/src/images/icons/app-icon-
96x96.png'}]
};

//Displaying notification
navigator.serviceWorker.ready.then(function(swregistered) {
swregistered.showNotification('Successfully subscribed!',
options);
});
}
```

Canvas, WebGL, HTMLMediaElement, and Media streams API

HTMLMediaElement, Canvas, Media Streams API and WebGL are combinedly used in Share Moments app to define the type of media, to provide an empty graphic zone for displaying photo image, to gain access to the device camera and to capture pictures respectively.

Program 26. An example of code that captures an image from device camera

```
//-------Use of HTMLMedia Element tag (video and canvas)--------
<video id="player" autoplay></video>
<canvas id="canvas" width = "320px" height="240px"></canvas>
<button “id="capture-btn">Capture</button>
//-------Initialise Media---------------------------------
function initializeMedia(){
if(!('mediaDevices' in navigator)){
navigator.mediaDevices={};
}

//-------Accesses the device camera-----------------------
navigator.mediaDevices.getUserMedia({video : true})
.then(function(stream){
videoPlayer.srcObject = stream;
videoPlayer.style.display = 'block'; //video player is displayed
}).catch(function(err){ //this will display imagepicker in error
imagePickerArea.style.display = 'block';
});
}

//-------Capture Image------------------------------------
captureButton.addEventListener('click', function(event){
canvasElement.style.display = 'block';
var context = canvasElement.getContext('2d'); //2d sketch
context.drawImage(videoPlayer, 0, 0, canvas.width,
videoPlayer.videoHeight/(videoPlayer.videoWidth/ canvas.width));
```

```
videoPlayer.srcObject.getVideoTracks().forEach(function(track){
track.stop(); //Stops the video and access to video's srcObject
});


// Turns raw data to blob(blob is file like structure that makes
easier to upload) and gets the picture to the server which is
Firebase at the moment
picture = dataURItoBlob(canvasElement.toDataURL());
});
```

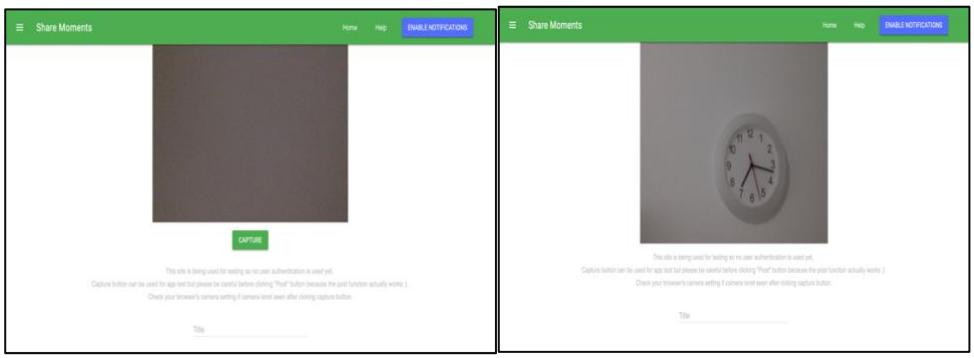

Figure 20. Empty graphics area (left) and an image captured by the camera (right).

This chapter covered the implementation of main web API used for the Share Moment app. The geolocation function with Google API is still in progress for this app, and it aims to track the location of the user to make an automatic update in location form. Next chapter discusses the additional implementation and approach for PWA to enhance the user experience.

7 ADDITIONAL IMPLEMENTATION TO ENHANCE PWA

Apart from the necessary files for PWA, some other files and method are required to provide a seamless PWA experience. This chapter describes four additional implementations. They are:

- Adding an app shell
- Adding ‘Add to Home Screen’ prompt or app install banner
- Choosing the right caching strategies
- Providing generic fall back page

7.1 Adding app shell

The app shell is the basic frame of an app which consists of pre-cached minimum HTML, CSS, and JavaScript for a UI. It ensures that the minimum UI and assets are visible to the user in an offline state, while dynamic contents are fetched when an internet connection is available. Service workers handle the storing of assets for app shell. The app shell architecture provides the following benefits:

- It creates a native-app-like experience through UI
- It caches content in the first visit, so, it loads faster in repeated visits.
- It aims to fetch a minimum amount of changeable assets, to lessen the data usage.
- The app updates automatically in each new visit.

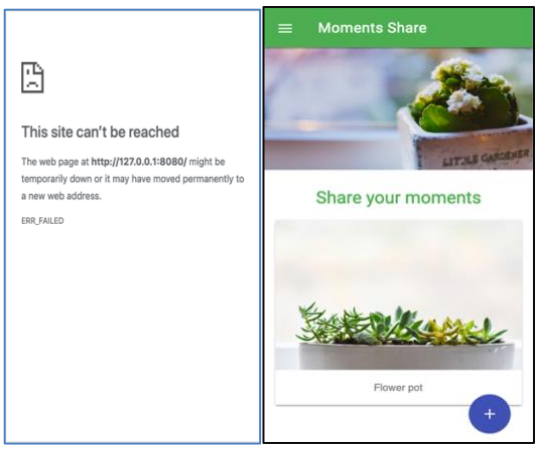

Figure 21. A PWA without an app shell (left) and with an app shell (right)

Every web domain or site gets one cache storage at the minimum. The app shell is built by storing assets in cache storage of web browsers. Hence, it requires considerations on the minimum components of an app shell. If the cache storage is overpopulated, the browser tends to clean the space to match its memory, which might not display the intended app shell. (Osmani, 2019).

There are several strategies for pre-caching an app shell. The next chapter includes a detailed description of the types of caching and proper strategy for caching.

7.2 Adding ‘Add to Home Screen’ prompt or app install banner

Add to home screen(A2HS) feature allows adding a shortcut of the web apps or sites to the home screen, which provides easy access to users. For instance, FireFox for Android has a home icon with a plus sign(+) for A2HS feature whereas Chrome for Android provides a web app install banner. According to Mozilla Developer Network web docs, currently, A2HS feature is available for Android versions of Chrome, Opera and FireFox, and Android Webview.
This chapter contains criteria for A2HS and measures for displaying the install banner.

- **Criteria for A2HS**
  The requirements for A2HS features are:

- The device should not have the same PWA installed beforehand.
- The web app manifest file should include name or short_name, icons of at least 192px and 512 px sizes, start_url, background_color and display mode of either fullscreen, standalone or minimal-UI.
- The app needs to be served over HTTPS and not in HTTP.
- Service worker should be registered with a fetch event handler.

Besides, browsers might also have additional requirements such as the user must have been using the website for at least 30 seconds. Moreover, storing assets offline are handled by Service Worker API and Web Storage/IndexedDB respectively, so A2HS does not download app files and saves it for offline use.

- **Measures for displaying the app install banner**

Firstly, if all the criteria of A2HS are met, beforeinstallprompt event fires before the actual banner displays.

Program 27. An example of code to be fired before displaying app install banner

```
var deferredPrompt;

    window.addEventListener("beforeinstallprompt",
function(event){

       //  Display of App install banner

        console.log("beforeinstallprompt fired");

        event.preventDefault();

       // Stores the event for later use.

        deferredPrompt = event;

        return false;

 })
```

Secondly, the prompt is shown as app install banner. This step also includes handling of user's permission for installing the app. It is shown in the following code.

Program 28. An example of code for displaying an app install banner

```
if (deferredPrompt) {

  // Waits for response of user

    deferredPrompt.prompt();

    deferredPrompt.userChoice.then(function(choiceResult) {
```

```
      console.log(choiceResult.outcome);
      if (choiceResult.outcome === 'dismissed') {
        console.log('User cancelled installation');
      } else {
        console.log('User added to home screen');
      }
    });
    deferredPrompt = null;
  }
```

Thirdly, the browser verifies if the app is installed or not.

Program 29. An example of code for verifying if the app is installed or not.

```
window.addEventListener('appinstalled', (evt) => {
  app.logEvent('a2hs', 'installed');
});
```

Finally, browsers such as Safari and Microsoft needs to be taken into consideration as well because these browsers might not support PWA features yet. Hence, every HTML pages should also include explicitly added meta information in the head section.

Program 30. An example of code that shows additional metatags for iOS devices

```
<meta name="apple-mobile-web-app-capable" content="yes">

<meta name ="apple-mobile-web-app-status-bar-style"
content="black">

<link rel="apple-touch-icon" href="/src/images/icons/apple-icon-
57x57.png" sizes="57x57">
```

Program 31. An example of code for additional metatags for Internet Explorer.

```
<meta name="msapplication-TileImage"
content="/src/images/icons/app-icon-144x144.png">

<meta name="msapplication.TileColor" content="#fff">

<meta name="theme-color" content="#3f51b5">
```

The A2HS banner provides better user experience for the PWA. Likewise, offline capability also enhances the experience of using an app. Hence, next chapter elaborates the right caching strategies for PWA to make it offline capable.

7.3 Choosing the right caching strategies

Caching can be divided into static caching and dynamic caching. This chapter provides details about divisions and sub-divisions of the caching approach.

- Static caching
- Dynamic caching

7.3.1 Static caching

Static caching refers to storing assets that rarely undergo changes and modification by the user. In this caching, the files or URLs are defined manually. In the Share moment app, the help page is an example of static content. The content of the help page required no modification; therefore, the offline help page was assigned to be statically cached, once the app loads.

Program 32. An example of code that caches file manually

```
var STATIC_FILES = ['/', '/index.html', '/offline.html',
'https://fonts.googleapis.com/css?family=Roboto:400,700'];
//Installing service worker
self.addEventListener("install", function(event){
console.log("[Service Worker]  Installing Service Worker ....",
event);
//Cache API
event.waitUntil(
caches.open(CACHE_STATIC_NAME)
.then(function(cache){
console.log("[Service Worker ] Precaching App Shell");
cache.addAll(STATIC_FILES); }))
});
```

7.3.2 Dynamic caching

Dynamic caching stores the changeable assets of a website such as files, images or script. Dynamic assets are user-driven, and it needs consideration in PWA because most of the website contains modifiable data from the user that requires frequent fetching from the database. In dynamic caching, old cache storage is commonly replaced by the latest version of dynamic content while the internet connection is available. (Vaadin, 2019a).

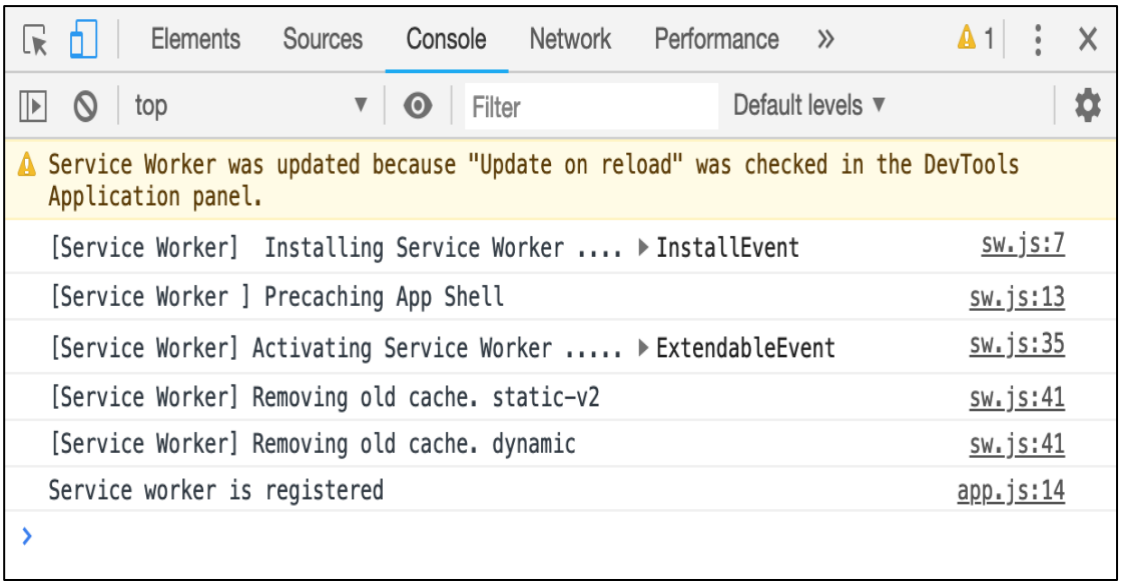

Figure 22. A logged view of the updated cache and service worker in a console.

There are several strategies to implement dynamic caching in an app. Few are listed as follows:

- **The cache then fallback to network strategy**
  In this approach, the request from the page goes through service workers that checks for the file in the cache, initially. If the files are not found in the cache, then the request to fetch the assets is redirected to the network. Then the response is sent to page from the web. This strategy is useful in apps that can operate with the offline-first approach. (Archibald, 2019b).However, cache then fallback to network shows file from the cache in initial load, so it might not be suitable for an app that requires real-time data such as monetary value, chat app, and weather app.

- **Cache and network race strategy**

  In this strategy, the request passes from the service worker to both cache and network, but only one response that returns quickly to the page is displayed. It is suitable in case of older devices with slower disk access so that response is not dependent on only one option. (Archibald, 2019b).

- **Network falling back to cache strategy**

  In this strategy, the service worker firstly fetches an updated version of assets from the server, and it uses the cached file, only if the network connection fails. It is suitable for requests that require updated information frequently such as in game-boards. However, for slower connections, this strategy can also give problems because service worker might display cached files only when the network is utterly failing. (Archibald, 2019b).

- **The cache then networks strategy**

  In this strategy, two separate requests go from service worker at the same time, one to cache and another to the network. Files found from the cache returns to the page; meanwhile, another request fetches the assets from the server, then it sends one copy of updated version to the cache and another copy to the page. However, the content should not disappear suddenly from the page due to update from the network. It is suitable for articles and social media pages. The cache then network strategy is used for the Share moment app to ensure that the existing cache version is visible to users until the new data is fetched from the network. (Archibald, 2019b).

Program 33. An example of code in fetch event listener that caches file dynamically

```
event.respondWith(
//Finding match in cache file
caches.match(event.request).then(function (response) {
if (response) {return response;}
//If the file is not found request is fetched from network and
even new cache file is opened

else {return fetch(event.request).then(function (res) {
return caches.open(CACHE_DYNAMIC_NAME).then(function (cache) {
cache.put(event.request.url, res.clone());
return res; })
}).catch(function (err) {//Code to display static cache});
}
}));
```

To sum up, a PWA might use a combination of all these caching strategies, depending upon the nature of the file and the targeted users. Apart from these strategies, a separate fallback page with pre-defined content can also be an alternative solution, and the following paragraph illustrates this idea in detail.

7.4 Providing generic fallback page

PWAs runs from the browser and pages displaying no internet connection can hinder the user experience. A browser can statically or dynamically cache the assets to view it later in no internet connection. However, if the page has not been visited at least once, before losing the network connection, then a fallback page can solve this issue. The content of a generic fallback page is pre-defined, so instead of an error message, the browser shows a statically pre-cached page. A basic fallback page avoids the sudden display of error page. (Archibald, 2019b).

In the Share moments app, the fallback page is the ‘help’ page, and it has the minimum app shell and link to redirect back to the home page.

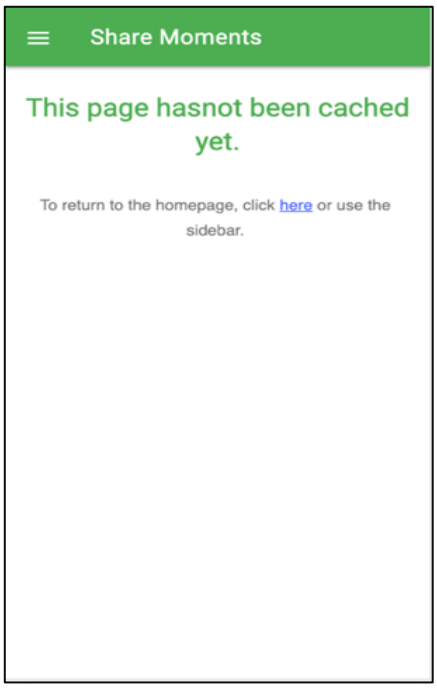

Figure 23. An example of a generic fallback page.

For the function in program 34 as follows, a file named offline.html is the fallback page, which resides in the root directory of the app. This page has the minimum app shell, taken from the main index page, a pre-defined message of error and link to the homepage. Every time an unvisited page is requested in offline condition, open () method

and catch() method of the cache API listens to that event and responds with the fallback page. This page is displayed again when the internet connection is re-established.
Hence, all the features mentioned earlier adds offline-capability to the user and progressively enhance the app.

Program 34. An example of code in fetch event listener responding to network error

```
.catch(function (err) {
//Code to be executed in case of error received from network
return caches.open(CACHE_STATIC_NAME)
.then(function (cache) {
//This checks header that accepts text/html and is a flexible
way of loading fallback page or any fallback file.
if(event.request.headers.get('accept').includes('text/html'))
{
return cache.match('/offline.html');
}});
});
```

8 CONCLUSION

This thesis deals with the technologies and implementation of a PWA, based on the resources mentioned in the introduction and reference section. It aims to be an understandable guide for the basic concept of PWA. One of the significant limitations for this topic was to find its official documentation or different documented books. Since PWA is a new concept, there is much ongoing discussion and interpretation as blogs and separate websites, but the book materials are comparatively fewer. Hence, referring to the documentation of the Google Developer website and blog of Alex Russell, a software engineer from Google, was the primary source of information. Alex Russell is one of two persons to draft the concept of PWA.

Rapid enhancement of web and browsers provides a promising future for more capabilities of PWAs. PWAs are still in their infancy stage, but their rapid progress is unignorable. It is undoubtedly a multi-dimensional concept to use one website as both mobile and desktop app. On the one hand, the device integration capabilities of web browsers include cameras, location sensors, and payment methods, and on the other hand, language required for PWAs is mainly JavaScript, which is commonly used programming languages in both client and server side web development. Hence, the argument favoring PWAs is relevant.

Similarly, from the study of PWAs technologies, it has been learned that making a basic functional PWA requires basic familiarity of HTML, CSS, JavaScript and its function in a webpage. If any static or moderately dynamic websites such as blogs are running over HTTPS, they can be turned into PWAs, by adding manifest.json, service worker and cache APIs. A web developer or person familiar with HTML, CSS, and JavaScript can understand the required syntax and concept faster, and learning PWA can be faster than learning a new completely new syntax for such as Java and Swift. In such a case, website owners might not need to hire mobile or desktop app developer separately. For WordPress websites, few plugins such as WordPress Mobile Pack, Wp-AppKit, Progressive Wordpress, PWA, and SuperPWA, are also available to turn websites into PWAs.(WP Missing, 2018).

In conclusion, PWAs might not replace native apps completely within few years; however, this technology has leveled up the standard of a modern website and provided more scope to web developers using HTML, CSS, and JavaScript. If the browser supports NFC (Near Field Communication), contacts, SMS (Short Message Service), augmented and virtual reality, PWAs can be used for contactless payment or sharing of files and creating immersive games using virtual reality. Therefore, PWAs might replace native apps and be the basic necessity of any website in the future.

REFERENCES

American Dialect Society. (2011a). *"App" 2010 Word of the Year, as voted by American Dialect Society*. Retrieved from https://www.americandialect.org/app-voted-2010-word-ofthe-year-by-the-american-dialect-society-updated.[Feb 27, 2019]

American Dialect Society. (2011b). *"App" 2010 Word of the Year, as voted by American Dialect Society*. Retrieved from http://www.americandialect.org/American-Dialect-Society-2010-Word-of-the-Year-PRESS-RELEASE.pdf.[Feb 27, 2019]

Archibald, Jake. (2019a). *JavaScript Promises an Introduction.* Retrieved from https://developers.google.com/web/fundamentals/primers/promises. [Mar 15, 2019]

Archibald, Jake. (2019b). *The Offline Cookbook.* Retrieved from https://developers.google.com/web/fundamentals/instant-and-offline/offline-cookbook/#cache-falling-back-to-network.[Mar 12, 2019]

Dashtaki, Mohammad Jamal. (2017). *Difference between a progressive web app and a hybrid mobile app.* Retrieved from https://stackoverflow.com/questions/40820592/difference-between-a-progressive-web-app-%20%20and-a-hybrid-mobile-app.[Feb 27, 2019]

Google Developers. (2019a). *Progressive Web App Checklist.* Retrieved from https://developers.google.com/web/progressive-web-apps/checklist [Feb 27, 2019]

Google Developers. (2019b). *Working with IndexedDB*. Retrieved from https://developers.google.com/web/ilt/pwa/working-with-indexeddb [Feb 27, 2019]

Infogineering. (2019). *The Web and the Internet are not the same thing*s. Retrieved from http://www.infogineering.net/web-internet.htm [Feb 27, 2019]

Lindesay, Forbes. (2019). *Promises*. Retrieved from https://www.promisejs.org/ [Mar 17, 2019]

MDN web docs. (2019a). *Introduction to web APIs*. Retrieved From https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Introduction.[Mar 17, 2019]

MDN web docs. (2019b). *Notifications API.* Retrieved from https://developer.mozilla.org/en-US/docs/Web/API/Notifications_API.[Mar 17, 2019]

Osmani, A. (2019). *The App Shell Model.* Retrieved from https://developers.google.com/web/fundamentals/architecture/app-shell.[Mar 17, 2019]

Russell, A. (2015). *Progressive Web Apps: Escaping Tabs Without Losing Our Soul.* Retrieved from https://infrequently.org/2015/06/progressive-apps-escaping-tabs-withoutlosing-our-soul/.[Feb 27, 2019]

Schwarzmüller, Maximilian. (2019). *Progressive Web Apps (PWA) - The Complete Guide.* Retrieved from https://www.udemy.com/progressive-web-app-pwa-the-complete-guide/.[Mar 12, 2019]

Statista. (2018). *Mobile OS market share in 2018.* Retrieved from https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/.[Mar 14, 2019]

Statista. (2019a). *Global market share held by smartphone operating systems from 2009 to 2017*. Retrieved from https://www.statista.com/statistics/263453/global-market-share-held-by-smartphone-operating-systems/.[Mar 14, 2019]

Statista. (2019b). *Smartphones industry: Statistics & Facts*. Retrieved from https://www.statista.com/topics/840/smartphones/.[Mar 17, 2019]

Vaadin. (2019a). *Caching strategies*. Retrieved from https://vaadin.com/pwa/learn/caching-strategies.[Mar 19, 2019]

Vaadin. (2019b). *Progressive Web Applications.* Retrieved from https://vaadin.com/pwa.[Mar 19, 2019]

Vaadin. (2019c). *Web App Manifest.* Retrieved from https://vaadin.com/pwa/learn/web-appmanifest.[Mar 19, 2019]

Vaadin. (2019d). *What is Service Worker?* Retrieved from https://vaadin.com/pwa/learn/serviceworker.[Mar 19, 2019]

World Wide Web Consortium(W3C). (2004a). *Architecture of the World Wide Web, Volume One.* Retrieved from https://www.w3.org/TR/2004/REC-webarch-20041215/.[Mar 19, 2019]

World Wide Web Consortium(W3C). (2004b). *Help and FAQ - W3C - World Wide Web Consortium.* Retrieved from https://www.w3.org/Help/.[Mar 19, 2019]

World Wide Web Consortium(W3C). (2004c). *What is the difference between the Web and the Internet?* Retrieved from https://www.w3.org/Help/#webinternet.[Mar 18, 2019]

WP Missing. (2018). *5 Best Progressive Web App Plugins for WordPress.* Retrieved from https://wpmissing.com/plugins/best-progressive-web-app-plugins/.[Mar 18, 2019]