PERFORMANCE ENHANCEMENT OF WEBPAGE USING PROGRESSIVE WEB APP FEATURES

Dr. V. Karpagam,
Professor/IT Department
Sri Ramakrishna Engineering College
Coimbatore, India

Padmavathe. R,
IV Information Technology
Sri Ramakrishna Engineering College
Coimbatore, India

Lakshana. R,
IV Information Technology
Sri Ramakrishna Engineering College
Coimbatore, India

Priyadharshini.S
IV Information Technology
Sri Ramakrishna Engineering College
Coimbatore, India

Manuscript History
Number: IJIRAE/RS/Vol.04/Issue03/MRAE10102
Received: 07, March 2017
Final Correction: 20, March 2017
Final Accepted: 28, March 2017
Published: March 2017

Abstract— The progressive web application combines the best of web and mobile apps. It is a website built using web technologies that acts like an app. Recent advancements in the browser, availability of service workers, Cache and Push APIs have enabled web developers to allow users to install web apps to their home screen, receive push notifications and even work offline. To use a traditional app, the user must install it beforehand which includes multiple clicks making the app unappealing to the user. This problem is solved by using PWA enabled webpage. The user is given the advantage of accessing the webpage app-like by creating a desktop icon which eliminates the need for multiple clicks. The primary characteristic of this progressive web app is that it must work on all devices and must enhance on devices and browsers that allow it. They take advantage of the much larger web ecosystem, plugins and community and the relative ease of deploying and maintaining a website when compared to a native application in the respective app stores.

Keywords— Progressive, Offline, Service Worker, AppShell, App Manifest, Cache recycle

1. INTRODUCTION

Progressive Web Apps (PWA) are an idea first espoused by Google engineer Alex Russell in June 2015. Progressive Web App uses modern web capabilities to deliver an app-like user experience. PWA describes a collection of technologies, design concepts, and Web APIs that work in tandem to provide an app-like experience on the mobile web. Native mobile app store apps do things like send push notifications, work offline, look and feel like an app (as Apple and Google have imagined them), load on the home screen. Mobile Web Apps accessed in a mobile browser, by comparison, historically haven't done those things. Progressive Web Apps fix that with new Web APIs, new design concepts, and new buzzwords. Progressive Web Apps bring features we expect from native apps to the mobile browser experience in a way that uses standards-based technologies and run in a secure container accessible to anyone on the web. Progressive Web Apps are linkable with an URL, fully responsive and secure. Progressive Web Apps start out as tabs in Chrome and become progressively more "app" like; the more people use them, to the point where they can be pinned on the home screen of a phone or in the app drawer and have access to app-like properties such as notifications and offline use.

II. FEATURES AND CORE TENETS OF PROGRESSIVE WEB APPS

A. FEATURES OF PWA

Progressive Web Apps are user experiences that have the reach of the web. They have the following features:

1. RELIABLE - Load instantly and never show the downasaur, even in uncertain network conditions. When launched from the user's home screen, service workers enable a Progressive Web App to load instantly, regardless of the network state. A service worker is like a client-side proxy, written in JavaScript and puts you in control of the cache and how to respond to resource requests. By pre-caching key resource, you can eliminate the dependence on the network, ensuring an instant and reliable experience for your users.
2. FAST - Responds quickly to user interactions with silky smooth animations and no janky scrolling. 53% of users will abandon a site if it takes longer than 3 seconds to load. And once loaded, they expect them to be fast, no janky scrolling or slow to respond interfaces. Improving performance is a process that starts with minimizing, or at least, optimizing the data that users download. Understanding how a browser renders those resources is a prerequisite for improving code efficiency.
3. ENGAGING - Feel like a natural app on the device, with an immersive user experience. Progressive Web Apps are installable and live on the user's home screen, without the need for an app store. They offer an immersive full screen experience with help from a web app manifest file and can even re-engage users with web push notifications. The Web App Manifest allows you to control how your app appears and how it's launched. You can specify icons for the home screen and splash screen which is shown while the app is loading. Which page is loaded when the app is launched, screen orientation, even whether to show the browser chrome or not.
4. INSTALLABLE - Allow users to keep apps they find most useful on their home screen.
5. FRESH - Always up-to-date thanks to the service worker update process.
6. APP-LIKE - Feel like an app to the user with app-style interactions and navigation.

B. CORE TENETS OF PWA

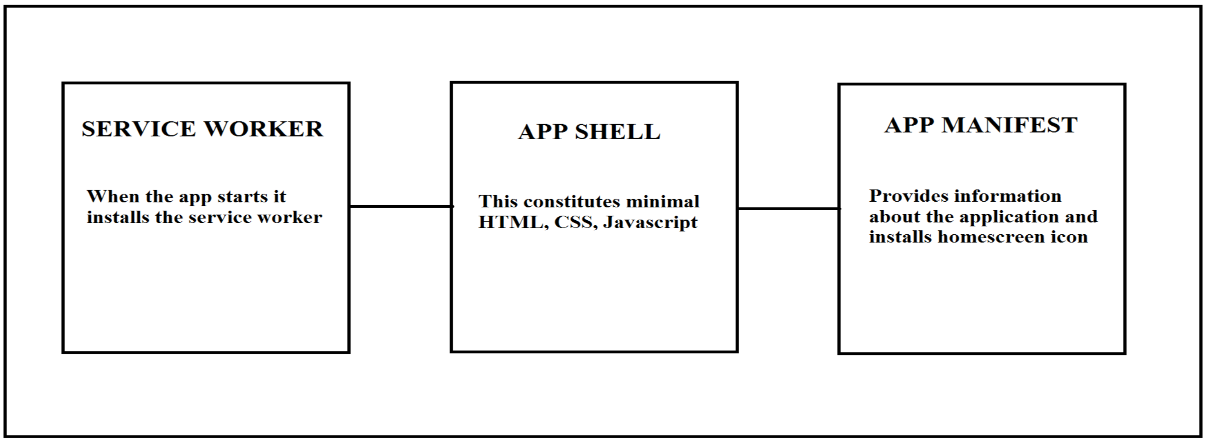
SERVICE WORKER
When the app starts it installs the service worker
APP SHELL
This constitutes minimal HTML, CSS, Javascript
APP MANIFEST
Provides information about the application and installs homescreen icon

Fig 1: Core tenets of PWA

1. SERVICE WORKERS

Service Workers are an incredibly powerful tool behind Progressive Web App. They provide offline functionality, push notifications, background content updating, content caching, and a whole lot more.

Service workers perform the following functionalities

- *Caches the App Shell - html/css/js in the phone.*
- *Update the Content in background.*
- *Gets the push notification id from the user to send the notification.*
- *Invalidates the cache when needed. Service Worker - Libraries*
- *sw-precache*
- *sw-toolbox*
- *sw-offline-google-analytics*

2. APP SHELL

In Application Shell Architecture, the shell is served up by the Service Worker and then the content is delivered. These are often cached by the Service Worker from its source through API requests. Sites that people visit more often will be able to hold the last content the person visited while waiting for the network to dynamically load the latest refresh. With the App Shell model, the focus is on keeping the shell of app UI and the content inside of it separate, and they are cached separately. Ideally, App Shell is cached such that it loads as quickly as possible when a user visits and returns later. Having the shell and the content load separately theoretically improves the user's perception of the performance and usability of the app.

3. INSTALLABILITY AND APP MANIFEST

Web pages were not installed like an app to the home screen. Sure, a user could "pin" a mobile website to their home screen on iOS and Android. The web App Manifest gives the developer the ability to control how the app appears to the user in areas where they would expect to see apps (for example, a mobile device's home screen), direct what the user can launch, and define its appearance at launch.

Web App Manifests provide the ability to save a site bookmark to a device's home screen. When a site is launched this way:

- *It has a unique icon and name so that users can distinguish it from other sites.*
- *Used to create an icon in the home screen of the phone.*
- *Contains the background colour for the app.*
- *Contains the start url for the app.*

III. EXISTING SYSTEM

When the user goes offline in traditional WebPages he/she will be notified as server not found error. If the browser supports caching of the webpage data, then the cached data will be displayed. When a webpage is bookmarked, then it can be viewed only when the user is online. Traditional WebPages could not behave app like. Traditional web pages couldn't be high performant. The developer must take care of handling offline scenarios as the networks were flaky. When there are multiple platforms, the maintenance became tough.

IV PROPOSED SYSTEM

Progressive Web App uses modern web capabilities to deliver app-like user experience. PWA starts out as tabs in Chrome and become progressively more "app" like. PWA allows users to install web apps to their home screen, receive push notification and even work offline. The core tenets of PWA are Service Worker, App Shell and App Manifest. When the application starts, the service worker will be enabled. The main JavaScript and css files will be cached inside the app shell. The PWA enabled WebPages to provide an advantage of viewing the cached copy of webpage even without internet. The user can also subscribe for receiving notifications from the webpage. When a page is bookmarked, an active internet connection is needed to view the data on webpage. In PWA, a desktop icon can be created by using App Manifest for a webpage which allows us to view the cached copy of the website in offline mode. It caches large or frequently used resources, like images, at runtime at the first instance. The cache will be recycled, so that the user will be able to view the recently used page of website. Thus, it loads fresh images, API responses, or other dynamic content from the network while online, but a cached copy while offline.

V. MODULE DESCRIPTION

- *Offline*
- *Desktop icon*
- *Push notification*
- *Cache recycle time*

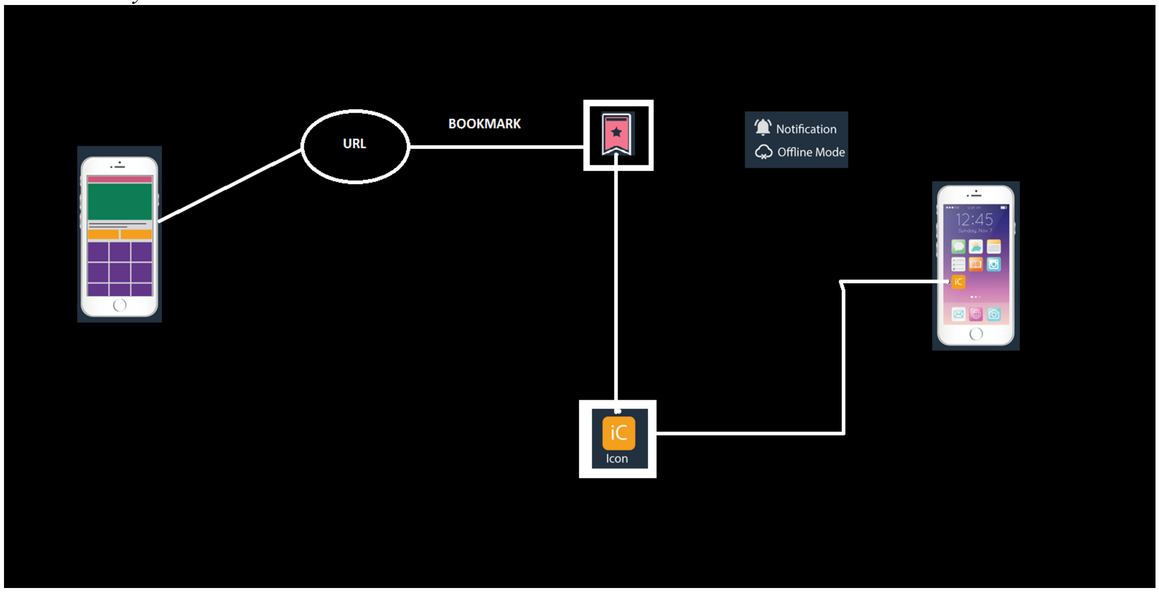

Fig 2: Overview Diagram

A. OFFLINE

Internet connections can be flaky or non-existent on the go, which is why offline support and reliable performance are common features in progressive web apps.

General recommendation for storing data offline

- *For URL, addressable resources use the Cache API (part of service workers).*
- *For all other data, use Indexed DB (with a promises wrapper).*

The App Shell is the minimal HTML, CSS, and JavaScript that is required to power the user interface of a progressive web app and is one of the components that ensures reliably good performance. Its first load should be extremely quick and immediately cached. "Cached" means that the shell files are loaded once over the network and then saved to the local device. Every subsequent time that the user opens the app, the shell files are loaded from the local device's cache, which results in blazing-fast start up times. App Shell architecture separates the core application infrastructure and UI from the data. All the UI and infrastructure is cached locally using a service worker so that on subsequent loads, the Progressive Web App only needs to retrieve the necessary data, instead of having to load everything. The App Shell consists of the core components necessary to get your app off the ground, but likely does not contain the data.

The key components will consist of

- *Header with a title, and add/refresh buttons*
- *Container for forecast cards*
- *A forecast card template*
- *A loading indicator*

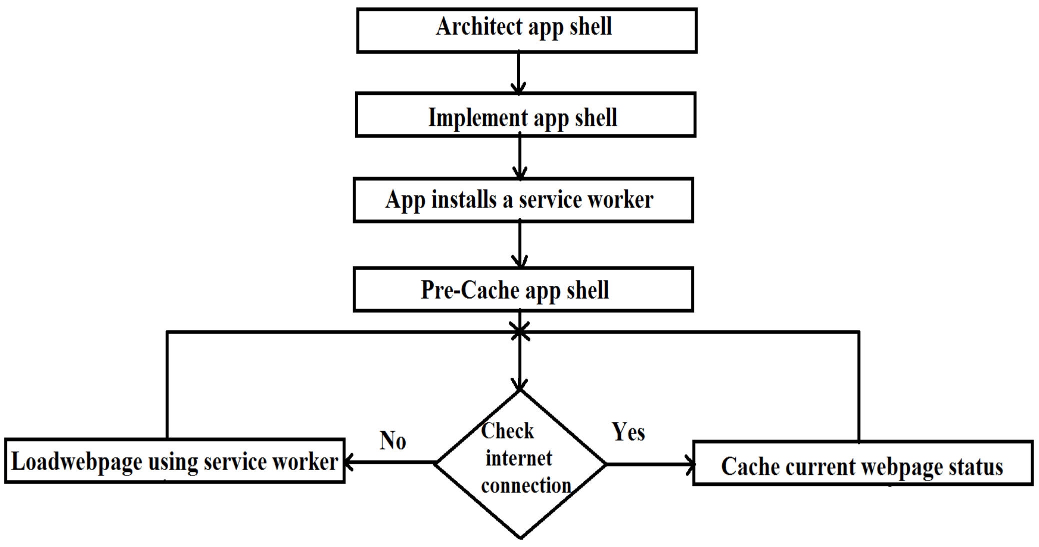
Architect app shell
Implement app shell
App installs a service worker
Pre-Cache app shell
Loadwebpage using service worker
No
Check internet connection
Yes
Cache current webpage status

Fig 3: Flow chart for Offline module

To make the app work offline, register the Service Worker, a script that allows background functionality without the need for an open web page or user interaction.

This takes two simple steps:

1. *Tell the browser to register the JavaScript file as the Service Worker.*
2. *Create a JavaScript file containing the Service Worker.*

When the Service Worker is registered, an install event is triggered the first time the user visits the page. In this event handler, all the assets that are needed for the application are cached. When the Service Worker is fired, it should open the caches object and populate it with the assets necessary to load the App Shell.

B. DESKTOP ICON

Feel like an app to the user with app-style interactions and navigation. Allow users to keep the apps which they find most useful on their home screen.

A Web App Manifest is a simple JSON file that gives you the ability to:

- *Control how the app appears to users in areas they would expect to see apps*
- *Choose a launch URL.*
- *Decide whether a launched app shows the browser chrome.*

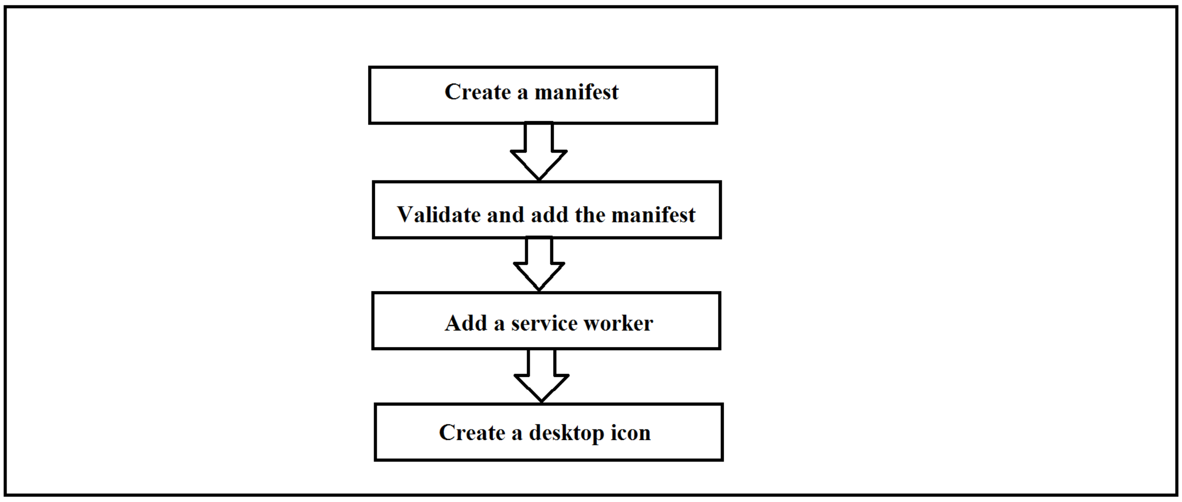

Fig 4: Flow chart for desktop icon module

Web App Manifests are part of a collection of web technologies called Progressive Web Apps, which are web applications that can be installed to the home screen of a device without needing the user to go through an app store, along with other superpowers such as being available offline and presenting users with push notifications when application content changes.

C. PUSH NOTIFICATION

Push API gives web applications the ability to receive push notification messages pushed to them from a server. This works hand in hand with the service worker. The process of a typical push notification flow in a web application is as follows: The web app brings forward a popup asking the user to subscribe to notifications. The user subscribes to receive push notifications. The service worker's push manager is responsible for handling the user's subscription. The user's subscription ID is used whenever messages are posted from the server. Every user can have a customized experience based on their subscription ID.

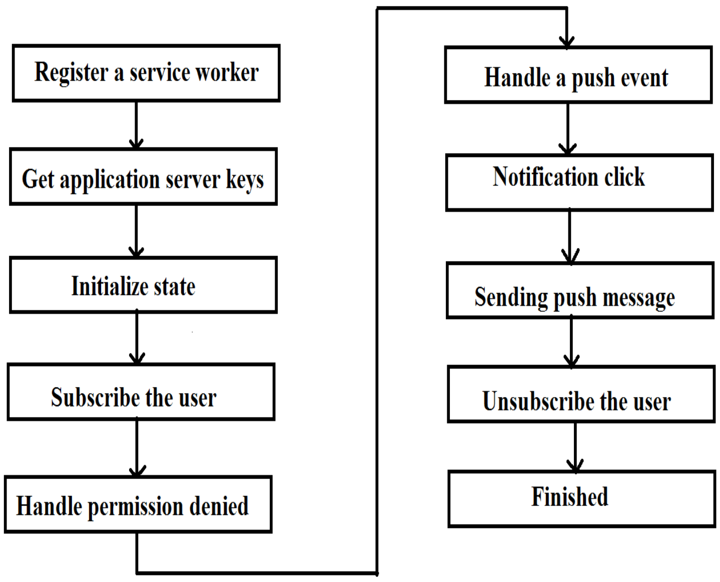

Fig 5: Flow chart for push notification module

First, the user request file needs to be registered as the Service Worker. This code checks if Service Workers and push messaging is supported by the current browser and if it is, it registers the Service Worker file. It has two functions in the main Service Worker file, one is used to initialize the UI, which will check if the user is currently subscribed, and the other one which will enable our button and change the text if the user is subscribed or not. When the user clicks the push button, the button is disabled just to make sure the user can't click it a second time while we're subscribing to push as it can take some time. Then we call subscribe user method when we know the user isn't currently subscribed.

When the subscribe, method is called, it follows two steps,

1.Checks whether the user has granted permission to display notifications.

2.The browser has sent a network request to a push service to get the details to generate a Push Subscription.

If the permission is denied, then the user can't be subscribed and there is nothing more that can be done, so disabling the button for good is the best approach. When we trigger a push message, the browser receives the push message, figures out what service worker the push is for before waking up that service worker and dispatching a push event.

D. CACHE RECYCLE TIME

Service Worker gives an install event which starts a new service worker. The event is made non-dependent. The new service worker gets activated. If it is not possible to save the whole page offline, then the user selects the contents to be viewed online. Update the cache when the user is online. Push message allows the Service Worker to be initiated in response to a message from the OS's messaging service. This happens even when the user doesn't have a tab open to your site, only the Service Worker is activated. You request permission to do this from a page & the user will be prompted. Background synchronization allows you to request background data synchronisation as a one-off, or on an interval.

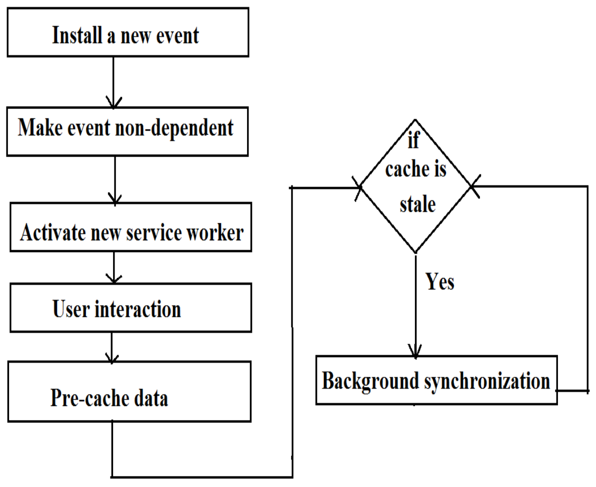

Fig 6: Cache recycle time diagram

VI. PERFORMANCE ANALYSIS

Google Lighthouse can be run as a Chrome Extension, from the command line, or used programmatically as a Node module. Lighthouse focuses on performance metrics and the quality of PWAs. An URL is given to the Lighthouse that needs to be audited, it runs a barrage of tests against the page, and then it generates a report on how well the page did. The failing tests can be used as indicators on what can be done to improve your app.

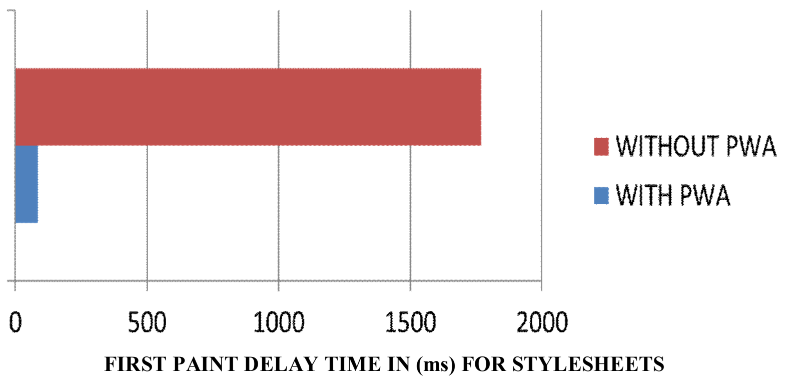

Fig 7: First paint delay time

First page is the first point where the browser has all the information it needs to render the page. This refers to the first time the browser will paint an image of the rendered page on the screen.

Style sheets are a type of template file consisting of font and layout settings to give a standardized look to documents. The time taken for the first paint from the style sheet of PWA enabled webpage is smaller (85 ms) when compared to webpage without PWA (1769 ms).

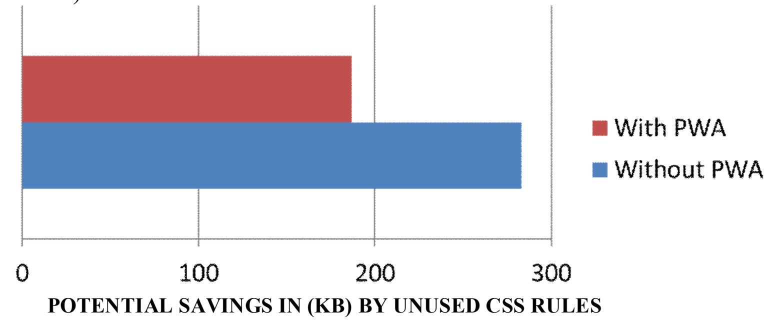

Fig 8: Potential Savings by Unused CSS rules

Before a browser can begin to render a web page, it must download and parse any style sheets that are required to lay out the page. Often, many web sites reuse the same external CSS file for all their pages, even if many of the rules defined in it don't apply to the current page. The best way to minimize the latency caused by style sheet loading and rendering time is to cut down on the CSS footprint. The potential savings of PWA enabled webpage is smaller (187 KB) when compared to webpage without PWA (283 KB).

VII. CONCLUSION

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.

REFERENCE

[1]. http://insanelab.com/blog/web-development/progressive-web-apps-pwa-vs-native/
[2]. http://www.mobileqazone.com/profiles/blogs/advantages-and-disadvantages-of-web-application-over-native
[3]. https://www.sitepoint.com/retrofit-your-website-as-a-progressive-web-app/
[4]. https://arc.applause.com/2015/11/30/application-shell-architecture/
[5]. https://medium.com/@slightlylate/progressive-apps-escaping-tabs-without-losing-our-soul-3b93a8561955#.idmgsmkxr
[6]. https://developers.google.com/web/updates/2015/11/app-shell
[7]. http://myventurepad.com/android-instant-apps-vs-progressive-web-apps-opportunity-challenges-future-ahead/
[8]. https://www.udacity.com/course/intro-to-progressive-web-apps--ud811
[9]. https://developers.google.com/web/progressive-web-apps/
[10]. https://codelabs.developers.google.com/codelabs/add-to-home-screen/#0