IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol.-7, Issue-9, Sept 2019 E-ISSN: 2347-2693

Progressive Web Applications: Architectural Structure and Service Worker Asset Caching

Arush Agarwal[1*], Akhil Dixit[2]

[1]Electronics and Communication, Netaji Subhas University of Technology, Dwarka, Delhi
[2]Electronics and Communication, Delhi Technological University, Bawana, Delhi

**Corresponding Author: Agarwal.arush98@gmail.com*

DOI: https://doi.org/10.26438/ijcse/v7i9.127139 | Available online at: www.ijcseonline.org

Accepted: 09/Sept/2019, Published: 30/Sept/2019

***Abstract*—**Progressive Web Application has emerged to be the top contender to alternative native application development. Despite the fact that the development practices for traditional applications have matured and become systemic by providing templates, cross platform development still remains a prominent topic. Developing an application with different architectural structure is redundant and comparing these on different platforms becomes demanding. PWA provides a solution to these problems by having to write a single codebase and providing similar features to all platforms with browser support. In this research paper we analyse the foundations of PWA, its features and core functionalities. A detailed investigation of the architectural structure of a PWA application and its benefit over its counterparts has been viewed. A case study analysing web applications on different platform is carried out.

Keywords—Progressive Web Applications, Web View, Native Applications, Performance Testing, Service Worker

I. INTRODUCTION

Traditionally the mobile application development is performed by writing non re-usable native codes for different platforms. This often leads to creating different teams for the production of same application and due to the hardware differences between operating systems, the features also vary vastly. The web applications were truly a welcoming step towards a unified application development that would run and perform on the web browser of the device. Due to this a single code in html and JavaScript allowed to be fed through all the major electronics devices such as smartphones, personal computers and smart watches. But the major lag behind the unpopularity of these unifying web designs was that they could not give the user the satisfaction and amount of experience that traditional native applications provided. These native operating systems such as Android, IOS, and Blackberry market had applications that were developed specifically to their standards and hardware requirements. Due to this the working of the software was optimum with the user possessing maximum functionalities of the device. As it is with the advent of new technology, Progressive Web App emerged as an optimal solution to provide the user with a native application like experience, and offer the advantage of web applications[1][2][3]. PWA acts as a unifying technology that seeks to interconnect the traditional and modern features of development. Technologies such as "React Native" also try to overcome the bridge between cross platform development but still a truly unifying mode for the interconnection of application with the device is possible only by using Progressive Web Apps. This research paper provides a background to the architectural structure by discussing the implementation by different organizations. Moreover a comparative case study and future development is presented in this paper.

II. BACKGROUND

Progressive Web Apps is a new software and architectural development methodology which seeks to unify cross platform application development. PWA acts as a hybrid of the traditional web pages and mobile applications. It provides features of both the application styles by providing fast reload, smooth animations, junk free scrolling and seamless navigation even on low power network. It provides the user with an app like experience by providing home icon, and functionalities like push notifications which pop up on the screen. PWA are quite secured and can be used in corporates to reduce the cost of development of application for different platforms. Progressive web Applications are responsive in nature and can fit modern devices screen size, with the advent of service worker class these applications provide offline support and connectivity independency by caching the pages. Throughout the app development, HTTPS protocol is used which establishes a secure connection. A web app can be called a progressive web app if it possesses the **manifest file and service worker file**. Manifest

© 2019, IJCSE All Rights Reserved

file provides the app with an app icon for an app like experience. Service worker is responsible for providing the provision of an application like features such as push notifications services and offline functionality by using back-ground synchronization. Thus, when offline, service worker synchronizes the background data and serve it to the user until a network is connected[2][4].

The most important advantages that PWA have are, fast loading and a very low application size.

The analysis of Flipkart application showed: Flipkart android app size:11 Mb ; flipkart lite the PWA version: 250 KB. These applications due to their smaller sizes can even work in 2G or slow connectivity with relative ease.

III. ARCHITECTURE

Progressive Web App uses modern web technologies to deliver fast, engaging, and reliable mobile web experiences that are optimum for end user. PWAs evolve from pages in browser tabs to immersive apps by utilizing HTML and JavaScript and enhancing it to provide a first class native like experience for the user. Overall PWA represents a traditional webpage with added functionalities which are provided by the service worker class. The architecture behind PWA consists **of application shell which then utilises service worker for progressive capabilities.**

A. Application Shell

Certain local resources that the web app needs to create for an overall frame of the user interface, which it further populates using JavaScript are loaded onto the shell for extended offline capability. The shell of the functionality is loaded and displayed to the user and then the rest of the interface is dynamically loaded[5], [6]. The purpose of this shell is to instantly load the contents of the application. One main advantage of using this architecture is that app shell is not loaded every time the user visits. Some initial HTML is always in the hands of user when the device is not connected to the internet, thus offering the user with instant as well reliable performance. The app is filled with dynamic content afterwards whenever the user requests it. It is the App shell that provides the progressive web app the features of the native apps.

An app shell typically includes HTML, JavaScript, CSS, and also might contain static resources that provide the structure for the application. However, it does not include the actual content specific information for the page. In other words, the app shell contains the parts of the page that change infrequently and can be cached so that they can be loaded instantly from the cache on repeated visits.[7] Generally, this includes the pieces of UI common across a few different pages of site headers, toolbars and footers. Due to this we can compose everything other than the primary content of the page. Some static web apps, where the page content does not change at all, consists entirely of an offline app shell[4],[8].

B. Service Worker

The application shell contains the static resources but the availability of these resources are made available by the service worker. The caching and the storage functionalities allow to precache certain resources for faster loading, this is essentially the creation of application shell[14].The service worker provides application the ability to intercept and handle network requests, including managing multiple caches, minimizing data traffic, and saving offline user-generated data until online again[1] [11].

The service worker performs its functions without the need for having an open web page or user interaction. This enables new services such as Push Messaging or capturing user actions while offline and delivering them while online. Push Notifications provides the user with time to time updates- increasing user engagement with the app[12],[13].

So a service worker runs the application shell by:

The basic user interface elements developed using HTML, CSS and JS are displayed when the user first logs on. The service worker responsible for controlling the future navigation of the app is registered by the page on the initial web visits. A new service worker instance is created and an install event is triggered. The service worker responds to this install event. Cache is then added with the app shell once the installation of service worker is done. Future navigation on the website can thus be controlled afterwards by the service worker which is installed.

© 2019, IJCSE All Rights Reserved

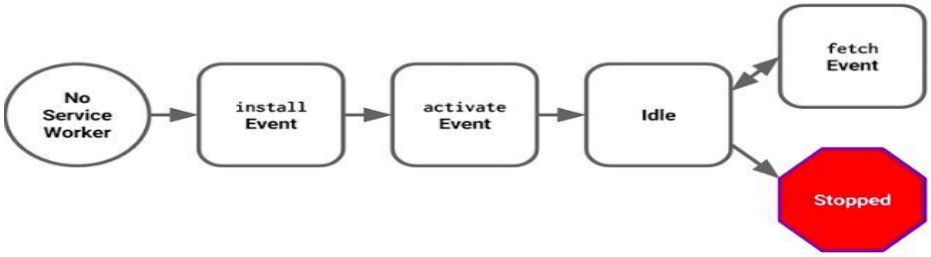


Fig. 1. Service Worker Life Cycle

2) The application screen and view is populated once the app shell content is uploaded and the app requests the same content. The complete rendered page consists of the app shell as well as new dynamic content. There is a **fetch event** inside the service worker. Every time a request is made this fetch event is triggered. Every time there is no event being fired the service worker goes into idle state. The service worker will remain in this idle state unless an event is fired. If remains idle for a long period of time the service worker gets automatically stopped and can be recovered on the occurrence of the fetch event.

3) It intercepts HTTPS requests so that application can decide what gets loaded from the cache, local data storage or the network. As the service worker runs independently from the rest of the application it provides link onto the underlying operating system. It responds to events from the OS, including push messages.

IV. CROSS PLATFORM APPLICATION DEVELOPMENT

As PWA consists of traditional HTML, CSS and JS, with the added characteristic of service worker. The typical web page can be displayed over any browser and hence website load up on smartphones, tablet, consoles, computers and watches. Hybrid applications such as react native does allow building upon android and ios devices but these can then cannot be used in websites or over other devices[15]. The emergence of PWA and its framework provides a true experience over all platforms. The cross platform development can be divided upon two categories:

1. Generative Approach: The main aim is to provide a native app for each supported platform. This native application has been generated from a single code base can be further differentiated. Two approaches are present- Model-driven approaches rely on a platform-independent model from which apps are generated. Typically, they employ a usually textual and, or possible alternatively, graphical and Domain Specific Language (DSL) to describe an app, Transpilers take (usually native) code written for one platform and transform it to native code for another platform.

2. Runtime environments: This approach does not generate new application for each platform. Hence instead of code running directly onto the OS or the platform, the run time behaves as a bridge between the application and OS. This environment in our case of PWA are web browsers or web view. In runtime environments the strategy for the rendering of the screen in the different modes is provided to the client side devices. web apps that work offline. Also, by pre-caching your site's critical static assets (or app shell, if your site is a single page application), you can give your users the experience of an almost instant page load on subsequent visits to your site.

V. ASSET CACHING WITH SERVICE WORKER

Service worker is a script that browser runs in the back-ground and has whatsoever no relation with web pages or the DOM, and provide out of the box features. Proxying network requests, push notifications and background sync are some of the features provided by service worker. Service workers ensure that the user has a good offline experience.

To allow authors to manage their content caches for offline use, the Service Worker API provide the Cache API that open and manipulate Cache objects. The Cache API was created to allow Service Workers to have a deeper and fine-grained control over the caching of resources. Through the Cache API, a Service Worker can decide to cache resources for offline consumption and retrieve them later. With resources, we mean complete responses from URL, like HTML pages, CSS, JavaScript files, images, JSON, and any other resource that can be consumed by your web pages or web apps.
In this section, we outline a few common patterns for caching resources: on service worker install, on user inter-action, and on network response.

© 2019, IJCSE All Rights Reserved

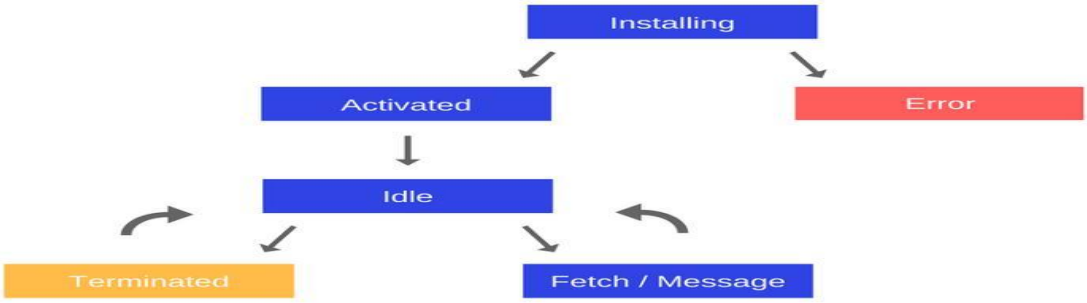


Fig. 2. Service Worker Life Cycle

A. **On install - caching the application shell**
The service worker's "install" event is a strategic time to cache static assets. This is often referred to as " precaching", since you are caching content ahead of the service worker being used. This is distinct from runtime caching, which saves files to the cache when they are returned from a network re-quest, after the service worker has been installed and activated.

Caching the application shell on install ensures that the service worker has access to all shell assets (if any assets fail to cache, the service worker installation is aborted).
The main reasons for doing this is that it gives developers control over the cache, meaning they can determine when and how long a file is cached as well as serve it to the browser without going to the network, meaning it can be used to create

B. **On Activate- remove outdated resources**
It's important to remove outdated resources to conserve space on users' devices. Old caches are deleted in the active event to ensure that caches are deleted before the new service worker has taken over the page (in case the new service worker activation fails, we don't want to remove the existing service workers caches). To remove outdated caches, an array of caches is created that are currently in use and delete all other caches.

C. **On Fetch-**
Whenever a fetch request is made, a fetch event is fired. In this event, caching strategies are implemented. The service worker working as a middle man, all the requests go through the service workers. From here on we decide whether we want that request to go to the network or to the cache. One of the examples of the strategy could be where the request is intercepted by the service worker and the request is made to the cache if the cache doesn't return a valid response then the request is fired to the network.

This approach works best for resources that frequently update, such as a user's inbox or article contents, since it is hard to make all files cached static in the service workers. Also, there could be the possibility that the user might not be visiting each and every page of your app then polluting their cache by installing so many files and slowing down the phone is of no use. Instead, we could keep caching the pages as per the user visit the pages. And this is known as the Dynamic cache or cache on demand.
It is important as :

1. Clean Code – The code should not end up looking like the complete application with several pages is being cached.

2. **Clean Cache** – The cache will only have those pages which the user visits. Hence, it does not pollute the users cache and only caches the user current application demand.

VI. CACHING STRATEGIES

To serve content from the cache and make the app avail-able offline, it needs to intercept network requests and respond with the files stored in the cache. All the caching strategies are to be implemented in the fetch event. Some of the caching strategies are-

© 2019, IJCSE All Rights Reserved

A. **Cache only**

The cache only strategy is useful for static sites which update infrequently, in this technique, we cache all the static resources needed and then all requests are served from the cache. It is cached in the install event, so the application uses the data saved in it. The response will look like a connection error if a match isn't found in the cache. This is suited for serving the resources for the App Shell and other static content that is only updated when the application version is updated.

When to use- This is used when the app needs access to only static-assets.

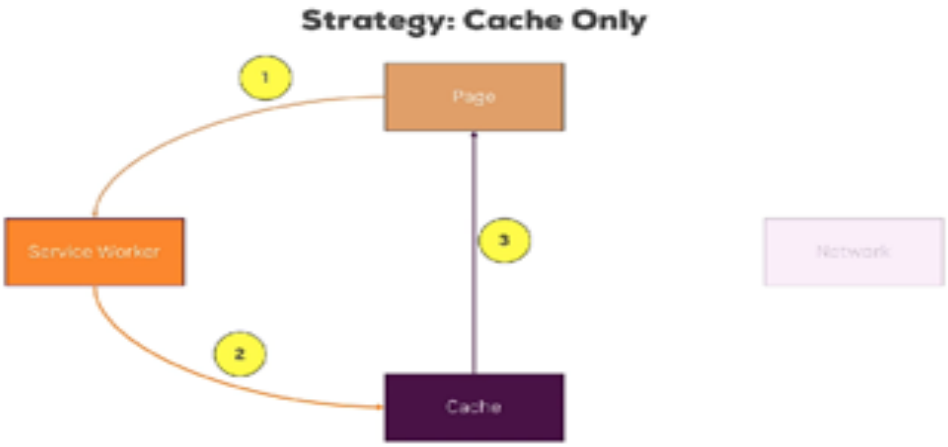


Fig. 3. Cache only

B. **Network Only**

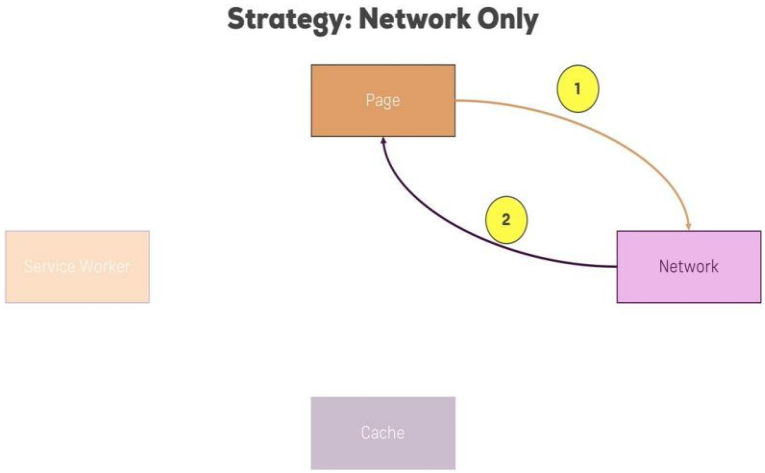


Fig. 4. Network Only

The client makes the request, the service worker intercepts it and makes the request to the network. This is the correct approach for things that can't be performed offline, such as analytics pings and non-GET requests.

When to use- When assets or data have no offline equivalent or saved information.

C. **Cache with Network fallback**

User makes the request, where the request is intercepted by the service worker and the request is made to the cache; if the cache doesn't return a valid response then the request is fired to the network. If application is on an offline-first, this is how it handles the majority of requests. Other patterns will be exceptions based on the incoming request. This gives PWA the "Cache only" behaviour for data in the cache and the "Network only" behaviour for anything not cached (which includes all non-GET requests, as they cannot be cached).

© 2019, IJCSE All Rights Reserved

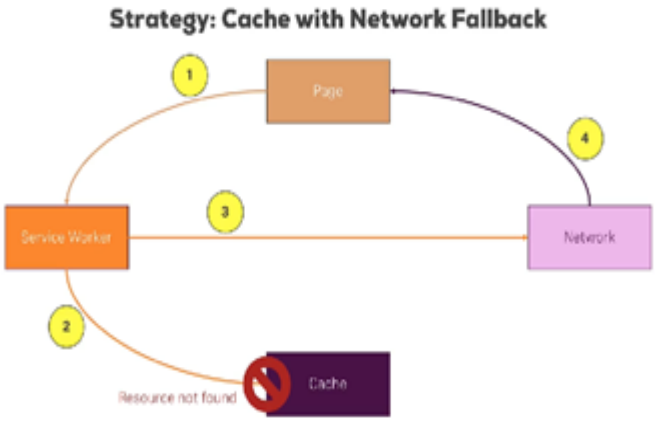


Fig. 5. Cache with Network fallback

When to use- If it is an offline first application

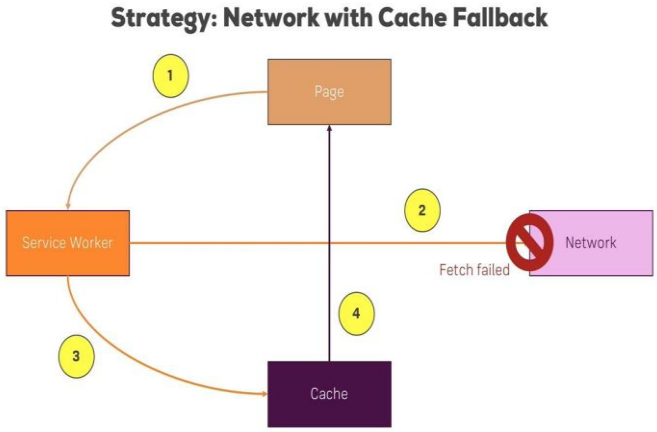


Fig. 6. Network with cache fallback

D. **Network with cache fallback**

First, the service worker makes a request to the network, if the request is successful then it proceeds else it falls back to the cache. If the resources are updated frequently, and they are not of the same version of the site (for example, articles, avatars, social media timelines, game leader boards), then this can be a good approach. Handling network requests this way means the online user get the most up-to-date content, and offline users get an older cached version.

However, this method has flaws. If the user has an intermittent or slow connection they'll have to wait for the network to fail before they get content from the cache. This can take an extremely long time and hence proves to be extremely frustrating. Here we first send the request to the network using fetch(), and only if it fails do we look for a response in the cache.

When to use- When you are building something that changes very often like a post page or a game leader board. When the priority is the latest data, this strategy is used.

E. **Cache, then Network**

This is also a good approach for resources that update frequently. This approach will get content on the screen as fast as possible, but still displays up-to-date content once it arrives. This requires the page to make two requests: one to the cache, and one to the network. The idea is to show the cached data first, then update the page when/if the network data arrives.

© 2019, IJCSE All Rights Reserved

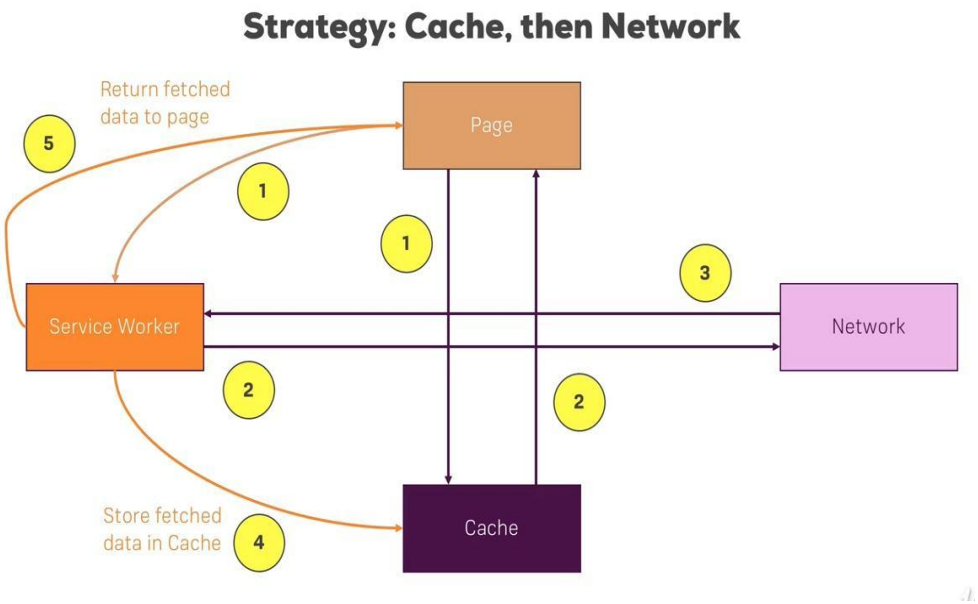


Fig. 7. Cache, then network

We are sending a request to the network and the cache. The cache will most likely respond first and, if the network data has not already been received, we update the page with the data in the response. When the network responds we update the page again with the latest information.

When to use- When it is needed to develop something that does not change very often and the old data is not very different from newer one. For example, if you load a page of blog posts from the cache and then add new posts to the top of the page as they are fetched from the network. The application can switch to this strategy if the priority is -speed and then the fetching the latest data.

F. **Generic Fallback**

When both the requests fail, one to the cache and other to the network, it displays a generic fallback so that the user won't experience a blank screen or some unknown errors. This technique is ideal for secondary imagery such as avatars, failed POST requests, "Unavailable while offline" page.

VII. **CASE STUDY**

Progressiveness of a Web App was analysed by performing a case study on web apps on market leaders of their field like Instagram, Swiggy, Tinder, Twitter, and Pinterest. We have analysed their performance on various fronts like installation size, performance analysis, caching techniques used and offline compatibilities. We then cumulated the results which can help developers to decide how to develop their PWAs.

A. **Installation Size**

Installation size is one of the biggest factors which led to the boom of PWAs. Where their native counterparts occupy space in several MBs, these PWAs have sizes in KBs solving the problem of insufficient storage space in mobiles.

Below are the comparison among sizes of native apps, mobile PWAs and the newly came in market, desktop PWAs showing the mammoth difference between their sizes. Data is taken for all the 5 apps taken into study.

From the results, we can infer that PWAs (mobile or desktop) are a great option if storage space is a limitation barrier in your system, and you don't want to compromise on performance.

© 2019, IJCSE All Rights Reserved

TABLE I

INSTALLATION SIZE

	Native App	Mobile PWA	Desktop PWA
Swiggy	42.68MB	209KB	Doesn't Exist
Pinterest	63.48MB	205KB	Doesn't Exist
Instagram	69.15MB		520KB
Twitter	59.19MB	209KB	Doesn't Exist
Tinder	94.93	274KB	575KB

B. **Performance**

Performance analysis was done using testing tools like Lighthouse, WebPageTest, and PageSpeed Insights to check the performance of app on various factors like Load Time, First Byte, Rendering Time Starting, Speed Index, First Contentful Paint, and First Meaningful Paint Tinder

An initial test was performed on Lighthouse tool which is provided by Google to test the progressiveness of a web application. The tool on performing the tests give a score based on the following metrics which are Performance, Accessibility, Best Practice, and Progressive Web App score, and SEO. On performing Lighthouse test on the Tinder Application following score was obtained for the app.

Fig. 8. Lighthouse performance score for application

The application obtained a performance score of 44 which depicts that the application was loaded slower as compared to the basic standards. The performance score was given to the app on the basis of the various metrics:

1. **First Contentful Paint**: First Contentful Paint (FCP) measures the time from navigation to the time when the browser renders the first bit of content from the DOM. This is an important milestone for users because it provides feedback that the page is actually loading.

2. **Speed Index**: Speed Index is a page load performance metric that shows you how quickly the contents of a page are visibly populated. The lower the score, the better.

3. **Time to Interactive**: The Time to Interactive (TTI) metric measures how long it takes a page to become interactive. "Interactive" is defined as the point where the page has dis-played useful content, which is measured with First Contentful Paint and event handlers are registered for most visible page elements.

4. **First Meaningful Paint**: First Meaningful Paint measures when the primary content of a page is visible. The lower your First Meaningful Paint score, the faster that the page appears to display its primary content.

5. **First CPU Idle**: First CPU Idle marks the first time at which the page's main thread is quiet enough to handle input.

6. **Max Potential First Input Delay**: First Input Delay (FID) measures the time from when a user first interacts with your site to the time when the browser is actually able to respond to that interaction.

© 2019, IJCSE All Rights Reserved

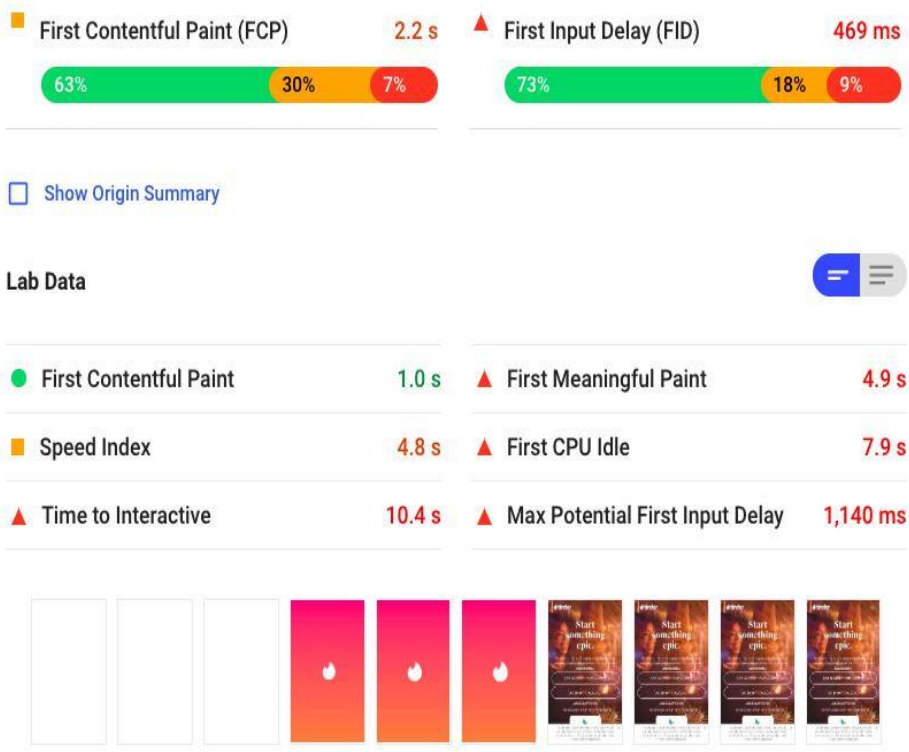


Fig. 9. Performance metrics of PWA

We have also analysed the same PWA using another testing tool **WebPageTest to confirm our results** that we got from LightHouse. This was done to ensure that our observations were not from only one source and were crosschecked from another testing tool. The outputs that we got from Web-Page Test were in alignment with our previous observations, thus, strengthening our results.

Serve offline page. This cache is persistent and independent from browser cache or network status. Caching resources makes the app content load faster on most network conditions.

It chooses to use SW-PreCache, a node module for generating the service worker and to precache static data that is not going to change. This is being used along with the sw-toolbox library for runtime caching. It is integrated with the build process, so a new service worker is generated after each build since Webpack generates dynamic chunk names, build process writes new chunk names directly to the SW-Precache loads template.

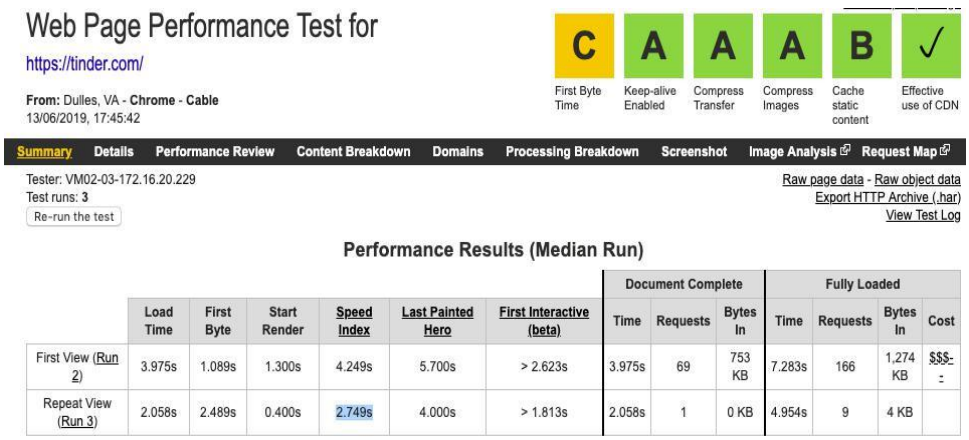
Network First cache strategy for API calls and CacheFirst, then Network strategy for the static assets are being used to cache assets.

2) Pinterest: Pinterest use the Workbox libraries for gen-eating and managing their Service Workers. They define a precache for the initial bundles loaded by the application shell (web pack's runtime, vendor and entry Chunks) too. As Pinterest is a site with a global presence, supporting multiple languages, they also generate a per-locale Service Worker configuration, so they can precache locale bundles. Pinterest also use web pack's named chunks to precache top-level async route bundles.

They decided to cache it in the application shell, which required some management of when to invalidate the app shell (logout, user information updates from settings etc.). Each request response has an 'appVersion' — if the app version changes, they unregister the Service Worker, register the new one then on the next route change they do a full page reload.

Pinterest cache any JavaScript or CSS bundles using a cache-first, then network strategy and also cache their user-interface (the application shell).

© 2019, IJCSE All Rights Reserved



Web Page Performance Test for
https://tinder.com/

From: Dulles, VA - Chrome - Cable
13/06/2019, 17:45:42

C	A	A	A	B	✓
First Byte Time	Keep-alive Enabled	Compress Transfer	Compress Images	Cache static content	Effective use of CDN

Summary | Details | Performance Review | Content Breakdown | Domains | Processing Breakdown | Screenshot | Image Analysis | Request Map

Tester: VM02-03-172.16.20.229
Test runs: 3
Re-run the test

Raw page data - Raw object data
Export HTTP Archive (.har)
View Test Log

Performance Results (Median Run)

	Load Time	First Byte	Start Render	Speed Index	Last Painted Hero	First Interactive (beta)	Document Complete Time	Document Complete Requests	Document Complete Bytes In	Fully Loaded Time	Fully Loaded Requests	Fully Loaded Bytes In	Cost
First View (Run 2)	3.975s	1.089s	1.300s	4.249s	5.700s	> 2.623s	3.975s	69	753 KB	7.283s	166	1,274 KB	$$$- -
Repeat View (Run 3)	2.058s	2.489s	0.400s	2.749s	4.000s	> 1.813s	2.058s	1	0 KB	4.954s	9	4 KB	

Fig. 10. Performance Analysis in WebPageTest

We can see the outputs that we got from WebPageTest were in alignment with our previous observations, thus, strengthening our results. Although some names were different like first byte and first contentful paint means same and also, first CPU idle and fully loaded document time, but all in all, they mean the same thing and have approximately equal values.

Similar performance analysis was performed for each of the remaining subject PWAs, results of which are tabulated in Table II.

C. **Caching Mechanics and Offline Compatibilities**

In this section, we have performed various tests to check how these PWAs have implemented the service worker caching using caching API. Precaching static data, response to offline (whether they respond with a 200 response or not), caching strategy implemented to fetch data, etc. were some of the factors on which analysis were done.

1) Swiggy: Swiggy PWA serves a response from cache even when the user is offline. Although, Service Workers caches JavaScript bundles, custom fonts, images, and also

2)

```
self.addEventListener('fetch', event => {
  event.respondWith(
    // match the request in cache
    caches.match(event.request).then( response => {
    // if request is not valid then fire a fetch request
      return response || fetch(event.request);
    })
  );
});
```

Fig. 11. Code Snippet for Cache falling back to network

3) Instagram: Instagram has used a totally different approach for caching assets in service worker. It has neither pre-cached any static files nor cached any dynamic files, although it has used disk and memory caching for images, JavaScript and CSS bundles. Due to this, it does not serve a response when the app is in offline stage. It has however, cached login credentials in cache storage. This strategy works for Instagram, in the sense that, it has to always show new data to the user.
The strategy used by Instagram is Network Only since no Service Worker Cache exists here.

© 2019, IJCSE All Rights Reserved

TABLE II

PERFORMANCE ANALYSIS

	First Contentful Paint	First Meaningful Paint	Speed Index	First CPU Idle	Time to Interactive	First Input Delay
Swiggy	2.6s	2.6s	3.3s	5.0s	5.6s	.09s
Pinterest	3.4s	4.2s	6.1s	6.6s	7.6s	.120s
Instagram	3.1	5.3	6.5	6.5	8.3	.180s
Twitter	4.8s	5.8s	9.7s	8.7s	9.4s	.240s
Tinder	2.2s	4.9s	4.8s	7.9s	10.4s	1.14s

4) Twitter: Twitter adopted an incremental approach to leveraging service workers for offline and network resilience, starting with a custom offline page presented whenever a net-work connection wasn't available. Lastly, they added support for offline caching of their application shell. Repeat visits are nearly instant thanks to service worker caching of current views, feed updates, notifications, messages and settings as they transitioned to offline caching of static resources like CSS, images and JavaScript bundles.

Using web pack, the app's scripts are broken up into granular pieces and loaded on demand. This means when available, a Service Worker will precache additional resources and allow instant future navigations to other screens.

Twitter PWA uses Cache first, for its application shell data and for dynamic data, it used Network Only strategy much like Instagram.

5) Tinder: Tinder PWA pre caches the application shell, so as to serve a response whenever a user goes offline. This cache is persistent and is not dependent on the type of browser.

Tinder uses the Workbox Webpack plugin for caching both their Application Shell and their core static assets like their main, vendor, manifest bundles and CSS. This enables network resilience for repeat visits and ensures that the application starts-up more quickly when a user returns for subsequent visits.

Tinder caches any of its static data using Cache First, with Network Fallback strategy and for dynamic data, it uses Network Only strategy. In the user engagement as well as they have provided the user with less loading time by adopting the progressive web architectural pattern. The industry is investing resources into progressive web apps (PWA) but the lack of academic involve-mint denotes a significant knowledge gap but at the same time provides research potential.

However, the massive interest by practitioners mandates further research. We have therefore sketched future developments and suggested a research agenda. We suggested a balanced approach of experimental and qualitative work. While we hope to encourage other researchers as well, our interest in the topic undoubtedly has been sparked.

The list below provides suggestions for further research of the same subject:

1. Continue the development of technical implementations for comparison and evaluation purposes against established development approaches.

2. Search for better strategies for caching and provide better responses when user is offline.

3. Study how initial bundles of JS, CSS and HTML could be disintegrated for faster loading.

4. Deepen the analysis on how performance can be improved further.

© 2019, IJCSE All Rights Reserved

TABLE III

COMPARISON OF CACHING STRATEGIES AND OFFLINE COMPATIBILITY

	Caching Strategy	Offline Compatibility
Swiggy	Network First for Dynamic data, CacheFirst, then Network for static data	Yes (appshell)
Instagram	Network Only	No
Pinterest	CacheFirst, then Network for all types of data.	Yes (appshell)
Tinder	CacheFirst, then Network for static data, Network Only for dynamic data.	Yes (appshell)
Twitter	CacheFirst, then Network for static data, Network Only for dynamic data.	Yes

VIII. CONCLUSION AND FUTURE SCOPE

Progressive Web Applications holds a strong future in the app development industry. Apps which the user does not use too often occupy a large part of memory. It is convenient to develop a progressive web application as there would be no need to install and save to memory. Therefore, PWA's will provide cost savings by reducing the amount of memory required. Several big companies have started to see increase

REFERENCES

[1] I. Malavolta, G. Procaccianti, P. Noorland, and P.Vukmirovi, "Assessing the Impact of Service Workers on the Energy Efficiency of Progressive Web Apps," pp. 52–62, 2017.
[2] I. Malavolta, "Beyond Native Apps: Web Technologies to the Rescue! (Keynote)." ACM Mobile!'16, Amsterdam, Netherlands, pp. 5–6, 2016.
[3] T. Leadership and W. Paper, "Native, web or hybrid mobile-app development."
[4] S. K. Gudla, J. K. Sahoo, A. Singh, J. Bose, and N. Ahamed, "A Systematic Framework to Optimize Launch Times of Web Apps," in International World Wide Web Conference Committee (IW3C2), published under Creative Commons CC BY 4.0 License. , 2017, pp. 785–786.
[5] I. Malavolta, G. Procaccianti, P. Noorland, and P. Vukmirovi, "Assessing the Impact of Service Workers on the Energy Efficiency of Progressive Web Apps," 2017.
[6] A. I. Wasserman, "Software Engineering Issues for Mobile Application Development," in FoSER 2010, 2010, pp. 397–400.
[7] H. Takabi, J. B. D. Joshi, and G.-J. Ahn, "Security and Privacy Challenges in Cloud Computing Environments," IEEE Secure. Priv. Mag., vol. 8, no. 6, pp. 24–31, 2010.
[8] N. Koch, P. Fraternali, and M. (Eds.), "Lecture Notes in Computer Science: Web Engineering," in 4th International Conference, ICWE 2004 Munich, Germany, July 26-30, 2004 Proceedings, 2004.
[9] G. S. T. Koziokas, Panagiotis T., Nikolaos D. Tselikas, "Usability Testing of Mobile Applications: Web vs. . . Hybrid Apps," in PCI 2017, September 28–30, 2017, pp. 9–10.
[10] H. Muccini, D. Informatica, A. Di Francesco, D. In-formatica, P. Esposito, and D. Informatica, "Software Testing of Mobile Applications: Challenges and Future Research Directions," pp. 29–35, 2012.
[11] I. Malavolta, S. Ruberto, T. Soru, and V. Terragni, "End Users ' Perception of Hybrid Mobile Apps in the Google Play Store," in Mobile Services (MS), 2015 IEEE International Conference on , 2015.
[12] Ian Warren ; Andrew Meads ; Satish Srirama ; Thi-ranjith Weerasinghe ; Carlos Paniagua, "Push Notification Mechanisms for Pervasive Smartphone Applications," IEEE Pervasive Compute. , vol. 13, no. 2, pp. 61–71, 2014.
[13] Satish Narayana Srirama, "Mobile web and cloud services enabling Internet of Things," CSI Trans. ICT, vol. 5, no. 1, pp. 109–117, 2017.

© 2019, IJCSE All Rights Reserved

[14] K. Behl, G. Raj, "Architectural Pattern of Progressive Web and Background Synchronization", in the International Conference on Advances in Computing and Communication Engineering (ICACCE-2018) Paris, France, pp. 366-371, 2018.

[15] T. A. Majchrzak, A.B. Hansen, T.M. Grønli, "ProgressiveWeb Apps: the Definite Approach to Cross-Platform Development?" In the Proceedings of the 51st Hawaii International Conference on System Sciences, pp. 5735-5744, 2018.

Authors Profile:

Mr. Arush Agarwal is currently pursuing his Bachelor of Engineering(undergraduation) from Netaji Subhas University Of Technology, Delhi. Currently in his final year he has published 4 research papers and has done extensive research work in his field. He currently majors in Electronics and Communication with.

Mr. Akhil Dixit is currently pursuing his Bachelor Of Technology in Electronics and Communication from Delhi Technological University, Bawana. Currently in his final year he has published 4 research papers in the domain of Machine Learning and Softwares. Having advent interest in computer vision and imageprocessing, he is currently working on the research work on the same.

© 2019, IJCSE All Rights Reserved