DOI: http://dx.doi.org/10.26483/ijarcs.v13i2.6810

Volume 13, No. 2, March-April 2022

ISSN No. 0976-5697

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

SERVICE WORKERS (SW): FACILITATING OFFLINE ACCESSIBILITY IN WEB APPLICATIONS

S. A. Idowu
Software Engineering Department
Babcock University
Ogun State, Nigeria

A. O. Adebayo
Information Technology Department
Babcock University
Ogun State, Nigeria

C. Ajaegbu
Information Technology Department
Babcock University
Ogun State, Nigeria

O. O. Adetunji
Software Engineering Department
Babcock University
Ogun State, Nigeria

Abstract: Consistent access to web application contents is of paramount importance to its end users. Irrespective of a user's location and the internet condition, there is the desire to interact with web applications, particularly through the use of mobile devices. Attempts have been made by previous researchers to introduce the Hypertext Transfer Protocol (HTTP) caching otherwise known as the browser cache which is automatically enabled by the browser and the Application Cache (AppCache) as an attempt to foster offline accessibility. While these approaches have helped, there still exists the problem of poor memory management and content validity.

A systematic Literature Review (SLR) was done on existing techniques, after which an overview of Service Workers (SW) and the identification of various SW caching strategies were proposed. Any of the proposed SW strategies can be adopted by web application developers based on the network condition and the contents of the application as this will help in reducing the loading time of the application, promote efficient mobile memory management, and increase the number of active users.

Keywords: Service Worker, AppCache, Caching, Web Applications, Offline Accessibility

I. Introduction

With the increase in the number of mobile devices, accessibility to web application contents and resources has become a great deal. According to (Statista, 2021) the world has 4.6 billion active Internet users out of which 4.3 billion access the Internet via mobile devices. Also, (Mayuran Sivakumaran & Iacopino, 2019; Taylor & Silver, 2019) showed that there are more than 5 billion people who possess at least a mobile device of which 57% of these devices are smartphones. It has therefore become a necessity to satisfy the need of the increasing smartphone users by developing necessary applications. The swift movement of humans from one place to another either on land, sea, or air can guarantee 100% access to mobile networks which can therefore lead to a sudden disconnect from the internet and cause a web application to go into an offline state thereby disrupting a user's flow of work.

Web application developers have tried to overcome these challenges faced by users through the rendering of pre-stored contents and assets which might become obsolete over time (Akeem & Sun, 2018) to users in an offline state. while this appears as a breakthrough for some developers, such web applications battle with the problem of poor memory management as it uses up limited storage within mobile devices (Al-Hunaiyyan et al., 2017; Eka et al., 2019; Oyelere et al., 2016) thereby bringing about poor memory management. Other developers have tried to harness the benefits offered by cache memories in a web browser. While caching might seem to be a preferred solution by keeping web contents for a specific period, it does not support the background synchronization of contents with the database (Chen, 2020). This research work, therefore, aims at proposing the integration of service workers in providing accessibility of web content to its users.

II. Literature Review

Caching is a mechanism that helps software applications reduce the number of hits to the database server which in turn improves the overall system performance (Meysam, 2018). Caching strategies in software applications have been developed because of the constant increase in application users requests (Zulfa et al., 2020), these strategies can be implemented in different locations (client, proxy, or server). The Hypertext Transfer Protocol (HTTP) caching otherwise known as the browser cache which is automatically enabled by the browser stores related resources (assets) the first time a browser loads a web page and subsequently presents such assets to the users in the future as long as the life of such resources has not expired. While HTTP caching still plays an important role in web applications, it is not reliable when the network is unstable or down (Chen, 2020). To overcome the limitation of the browser cache, the Application Cache (AppCache) was introduced to provide offline browsing to users as well as increase the speed and resilience of application software (Bidelman, 2013). In the AppCache, once a file is cached the browser shows the cached version even if there has been a change in the file on the server except if a change is made to the manifest file which will cause the entire files to be downloaded again (James, 2012), also an AppCache cannot be partially updated which has led to security and terrible usability issues (Firtman, 2016), currently there are plans remove support for AppCache by Chrome and other Chromium-based browsers (Posnick, 2021).

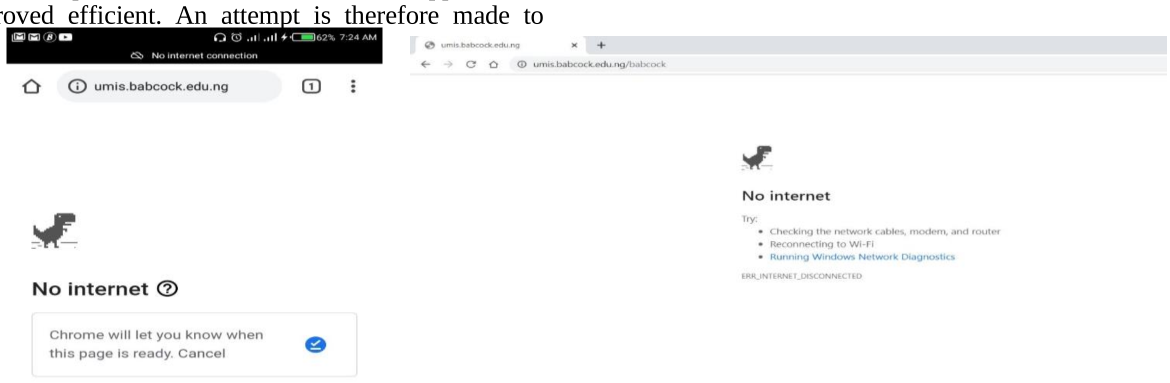
These limitations associated with the AppCache have led to the introduction of Service Worker (SW) which is a script that executes in a separate thread from the browser user interface in the background. It makes it possible for web applications to function offline (Chris, 2021; Firtman, 2016). The cache logic of the service worker does not need to be consistent with the HTTP caching expiry logic. SW which is a major component in the emerging Progressive Web Applications (PWA) mobile development approach, is a set of Application Programming Interfaces (APIs) that allows applications to work offline by intercepting network requests to deliver programmatically cached responses and preloaded assets, it also manages push notification, synchronizes background data even when the application is not in use and allows users to install the application to the home screen of their mobile devices irrespective of the form factor (Lee et al., 2018).

Based on the review, it is evident that existing mechanisms that attempted to provide offline access to web applications have not proved efficient. An attempt is therefore made to propose the use of service workers along with some caching strategies to effectively offer offline access to web application users.

III. SERVICE WORKERS

A. *Overview of Service Workers*

This is a technological component that facilitates the major functionalities of the Progressive Web Applications (PWAs). It is the key to unlocking the powers of PWAs because PWAs cannot work correctly without the implementation of service workers (Hume, 2018). It is an event-driven script that runs in the background on a separate browser thread to offer technical features such as background synchronization which defers actions until the user has stable connectivity, and push notifications that engage the users while the application is not opened (Gambhir & Raj, 2018; Parbat, 2018). As of today, when users attempt to view a website or web application with a flaky network connection, or lost network connection in the process, they are shown a response as depicted in figure 1.


No internet connection

umis.babcock.edu.ng

No internet

Chrome will let you know when this page is ready. Cancel

umis.babcock.edu.ng/babcock

No internet

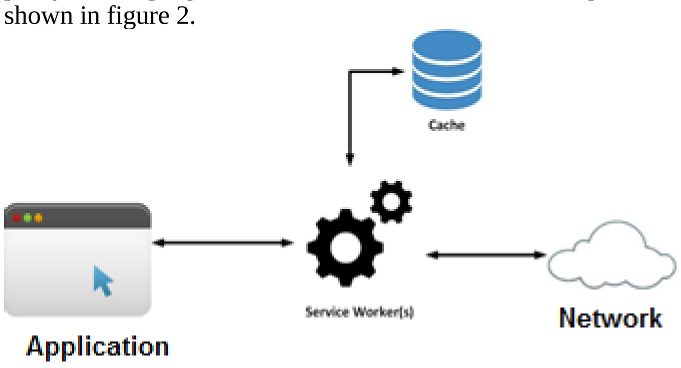
Try:

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

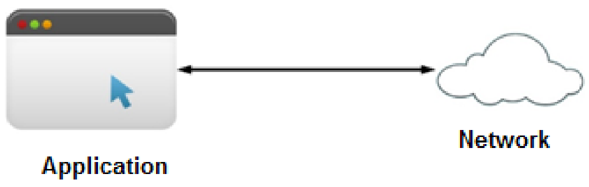
ERR_INTERNET_DISCONNECTED

Figure 1: A Mobile and Desktop View of a Web Application in an Offline State

Figure 1 implies that the current offline state of websites cannot provide any iota of useful information to its users. This is where the service worker comes in to turn the error as seen in figure 2.10 into something that can gracefully be handled. Invariably, the service worker acts as a proxy between the browser and the network that can run even when the browser is closed. According to (Mishra, 2016), SW is a programmable proxy where programmers can handle the network requests as shown in figure 2.



(a) Regular Web Applications



(b) Web Applications with Service Worker

Figure 2: Service Worker as a Network Proxy (Researcher's Image)

While figure 2(a) shows how a regular web connects to the Internet to retrieve a user's request, figure 2.11(b) shows how the SW resides between the application and the network which then decides when to retrieve contents from the cache and when to retrieve them from the network.

Every registered service worker runs on a separate thread different from the browser's main thread which runs persistently in the background even when the associated application is closed (Lee et al., 2018). (Malavolta et al., 2017) researched to confirm if there is a negative impact of the service worker on the energy efficiency of PWA. The result showed there was no significant impact on energy consumption. SW does not have direct access to the Document Object Model (DOM) because it is a JS worker, this is handled such that the SW communicates with pages on which it is applied by responding to messages sent through the

PostMessage() which in turn will modify the DOM (Gambhir & Raj, 2018). Because the SW can intercept network requests and can in one way or the other modify the content or even replace it with new ones, it is therefore imperative that SW are served over HTTPS (Parbat, 2018). HTTPS registers an SW on a browser and binds it t a website origin as defined by the HTTPS protocol, port, and domain. The essence of this is to secure the applications and prevent users from security attacks such as man-in-the-middle attack (Lee et al., 2018; Parbat, 2018).

Service workers are quite different from the standard JS files, though they are both written in JavaScript, the differences are highlighted as follows:

- SW runs its global script context.
- SW is not tied to a particular web page.
- SW cannot modify elements on the web page because it has no DOM access.
- SW can be only be served on the HTTPS protocol.

SW is designed to be completely asynchronous thereby making it impossible to access synchronous components such as the local storage.

B. *Service Worker (SW) Life Cycle and Events*

The SW is a short-life span event-driven script that is downloaded in the background whenever the user visits the pages (Gambhir & Raj, 2018; Parbat, 2018). Developers having good control over the cached resources is imperative in developing offline applications which can work optimally on a poor network. A service worker goes through different stages otherwise known as the service worker life cycle. Figure 3 shows the SW lifecycle that will take place when a user visits a webpage.

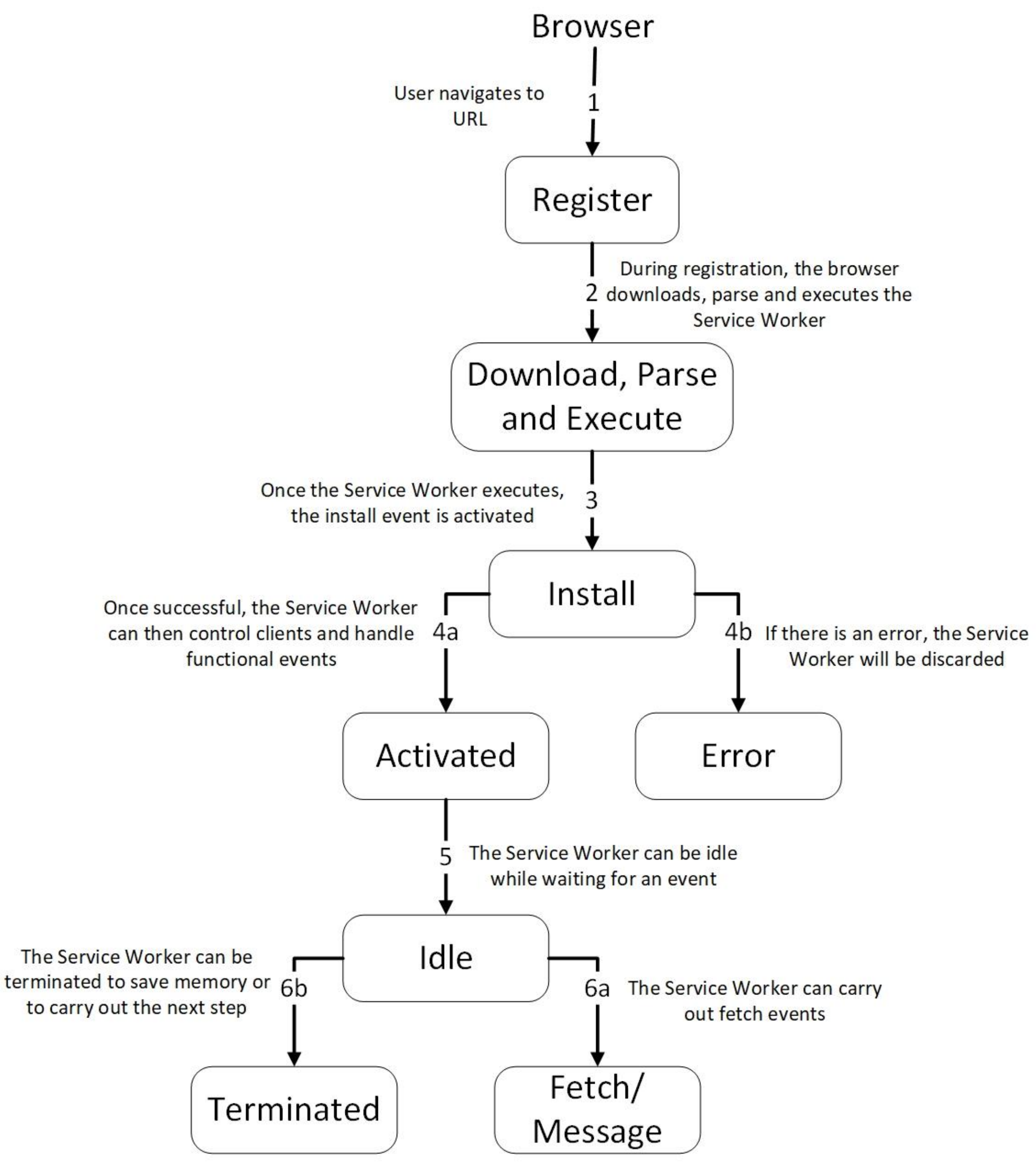

Browser
User navigates to URL
1
Register
2
During registration, the browser downloads, parse and executes the Service Worker
Download, Parse and Execute
Once the Service Worker executes, the install event is activated
3
Install
Once successful, the Service Worker can then control clients and handle functional events
4a
4b
If there is an error, the Service Worker will be discarded
Activated
Error
5
The Service Worker can be idle while waiting for an event
Idle
The Service Worker can be terminated to save memory or to carry out the next step
6b
6a
The Service Worker can carry out fetch events
Terminated
Fetch/ Message

Figure 3: Lifecycle of a Service Worker (Researcher's Image)

As depicted in figure 3 when a user accesses a webpage for the first time, the *register()* function is called during which the browser will download, parse, and execute the SW. Also, before a successful registration, there will be a check to see if the browser supports SW. Listing 1 shows sample codes used to register a service worker.

Once the SW has been successfully registered, the install event is activated to install the SW and initially cache the important assets of the websites. Listing 2 shows the install events and the caching assets. Once the installation of the SW is successful, the activate event is fired once and the SW is now in control of things within the scope in which it was defined. Listing 3 shows the activate event. Once activated, the SW begins to receive fetch and synchronization events which are triggered multiple times (based on each HTTPS request) as the user navigates from one page to another as shown in Listing 4.

1) Register Event: This event is used to register the presence of the service worker in the CB-MLMS and also to specify the location of the SW. Listing 4.2 shows the codes used in registering the service worker.

```
1.  let swURL = `path-to-serviceWorker/sw.js`;
2.    if ("serviceWorker" in navigator) {
3.      navigator.serviceWorker
4.        .register(swURL)
5.        .then((res)  =>  console.log("Service  Worker
    Registered", res))
6.        .catch((err) => console.log("Service Worker not
    registered", err));
7.    }
```

Listing 1: Sample code for the Register Event

From the above listing,

- Line 1: Specifies the path to the service worker script.
- Line 2: Checks to see if the student's browser supports SW before proceeding with the registration.
-
- Line 3-4: Registers the service worker found in the specified location.
- Line 5: Logs a statement to the console to the console if the registration is successful.
- Line 6: Logs a statement to the console to the console if the registration is not successful.

2) Register Event: After a successful registration, this install event is fired to install the specified SW, while this process is ongoing, the constant assets of the CB-MLMS are pushed into a static cache memory.

```
1.  self.addEventListener("install", (evt) => {
2.    evt.waitUntil(
3.      caches.open(staticCacheName).then((cache) => {
4.        cache.addAll(assets); //addding all assets
5.        console.log("Caching all assets");
6.      })
7.    );
8.  });
```

Listing 2: Sample Codes to Install and Cache Assets

From the above Listing,

- Line 1: This fires the install event (self – refers to the service worker).
- Line 2: While the event is being fired, this line puts the install event on hold
- Line 3-4: While the event is on hold, the static cache is opened and loaded with the static assets of the CB-MLMS.
- Line 5: Logs a statement to the console to show the caching of all assets.

3) Activate Event: A successful installation of the service worker does not connote the activeness of a service worker. The service worker only becomes activated after the activate event is fired. Upon activation, the old cache memory is expunged for the service worker.

```
1.  self.addEventListener("activate", (evt) => {
2.    evt.waitUntil(
3.      caches.keys().then((keys) => {
4.        return Promise.all(
5.          keys
6.           .filter((key) => key !== staticCacheName && key
    !== dynamicCacheName)
7.           .map((key) => caches.delete(key))
8.        );
9.      })
10.   );
11. });
```

Listing 3: Sample Codes to Activate the Service Worker

From the above Listing,

- Line 1: This fires the activate event (self – refers to the service worker).
- Line 2: While the event is being fired, this line puts the activated event on hold
- Line 3-7: This checks the current key(s) for the current cache memories (static and dynamic) respectively and deletes any cache memory that does not correspond with the current cache memory.

4) Fetch Event: After activation, the service worker begins to receive requests and synchronization events which will be triggered multiple times based on the HTTPS request as the users navigate from one page to another. While requests are made, the Dynamic cache memory is checked for a corresponding response, if the response is not found, then the request is fetched and the dynamic cache is updated.

```
1.  self.addEventListener("fetch", (evt) => {
2.    evt.respondWith(
3.          caches
4.           .match(evt.request)
5.           .then((cacheRes) => {
6.             return (
7.               cacheRes ||
8.               fetch(evt.request).then((fetchRes) => {
9.                 return
    caches.open(dynamicCacheName).then((cache) => {
10.                  cache.put(evt.request.url, fetchRes.clone());
11.                  limitCacheSize(dynamicCacheName, 40);
12.                  return fetchRes;
13.                });
14.              })
15.            );
16.          })
17.          .catch(() => {
18.            if (evt.request.url.indexOf(".js") > -1) {
19.              return caches.match("/fallback");
20.            }
21.          })
22.      );
23.  });
```

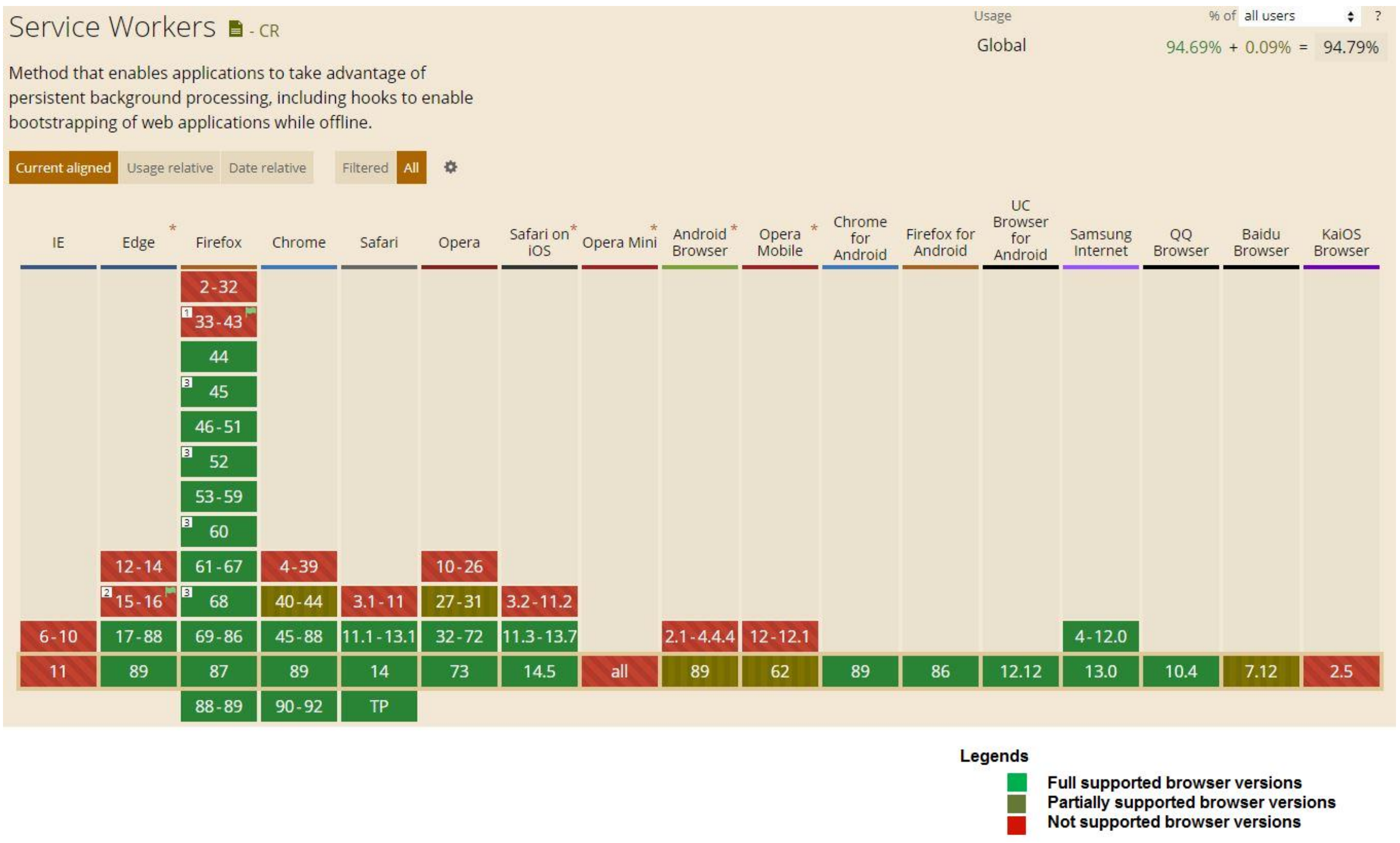
Listing 4: Sample Codes for the Service Worker Fetch Event

From the above Listing,

- Line 1: This fires the fetch event (self – refers to the service worker) whenever a request is made.
- Line 2-4: This checks the cache for a match with the user's request.
- Line 5-10: Once there is a match, the cached response is sent back to the user, if not the response to the request is pulled from the server, and a clone is made which is sent and stored in the dynamic cache.
- Line 11: Shows the number of responses that can be stored in the Dynamic cache at any given time. Once the number is exceeded, the first response in the dynamic cache is automatically deleted.
- Line 17-20: This shows a customized fallback page to the user if the user requests any file other than JavaScript file.

C. Browser Compatibility with Service Worker (SW)

SW is progressive and thus the experience varies across browsers (Parbat, 2018). For a PWA to fully implement its SW functionality, the user's browser must be fully compatible with SW. however, browsers with partial compatibility will experience some benefits of SW due to to its progressive enhancement nature. Figure 4 shows the current browser compatibility.



Service Workers - CR

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

Usage % of all users ?
Global 94.69% + 0.09% = 94.79%

IE	Edge	Firefox	Chrome	Safari	Opera	Safari on iOS	Opera Mini	Android Browser	Opera Mobile	Chrome for Android	Firefox for Android	UC Browser for Android	Samsung Internet	QQ Browser	Baidu Browser	KaiOS Browser
		2-32														
		1 33-43														
		44														
		3 45														
		46-51														
		3 52														
		53-59														
		3 60														
	12-14	61-67	4-39		10-26											
	2 15-16	3 68	40-44	3.1-11	27-31	3.2-11.2										
6-10	17-88	69-86	45-88	11.1-13.1	32-72	11.3-13.7		2.1-4.4.4	12-12.1				4-12.0			
11	89	87	89	14	73	14.5	all	89	62	89	86	12.12	13.0	10.4	7.12	2.5
		88-89	90-92	TP												

🟩 Full supported browser versions
🟫 Partially supported browser versions
🟥 Not supported browser versions

Figure 4: Current Browser Compatibility for Service Worker [19]

As shown in figure, there are eleven (11) browsers that fully support SW, two (2) browsers partially support SW, and three (3) browsers do not support SW at all. Comparing this report with that of (Parbat, 2018) where only 5 browsers fully supported SW, shows a consistent and significant increase in the number of browsers that fully supports SW, which implies that in coming years, all browsers will fully support SW thus making PWA a great choice.

IV. SERVICE WORKERS CACHING STRATEGIES

The SW can implement caching various scenarios to effectively present offline contents to users based on the network condition and the content. Below are various SW caching strategies that can be utilized by web application developers:

A. Fastest Caching Strategy

This strategy presents to the users the content that can be loaded first from the server through a network request or the cache memory. This is an efficient strategy with almost fresh content. The fastest strategy is depicted in figure 5.

B. Fastest Caching Strategy

Here, an attempt is made to provide a response based on the user's request from the cache. If the corresponding response is found in the cache (cache hit), it is sent to the user but the corresponding response is not found in the cache (cache miss), then the response will be pulled from the server through the network after which the cache will be updated. This caching strategy is efficient as it reduces the response time. However, the content in the cache might be outdated. This strategy is suitable for data that are stable over time. Figure 6 diagrammatically shows this strategy.

C. Network First Strategy

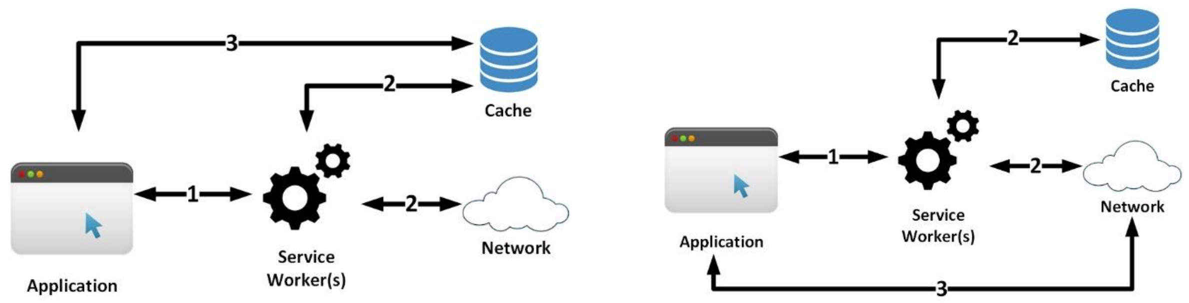
This is the direct opposite of the cache first approach. Here, an attempt is made to retrieve the user's response first from the network. If the network connection is established, then it is a success. If a network connection is unavailable, then the cache is accessed to retrieve the most recent data that correspond to the user's request. The advantage of this approach is that data presented to the users will always be updated provided there is a network connection. Network connection time out is however a challenge. This approach is the best fit for data that are constantly updated. Figure 7 shows this strategy diagrammatically.

D. Cache Only Strategy

This strategy loads users' responses from the cache only. There is no option for a network connection. This approach provides a fast response to the user but can only be utilized for data that are permanently cached and never needs to be updated. The cache-only strategy is depicted in figure 8.

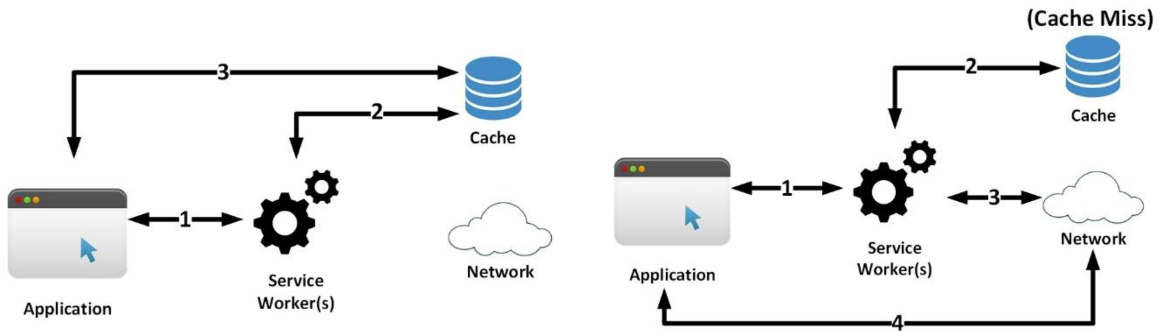
E. Network Only Strategy

This approach ensures that responses are only gotten through the network. Cache memory is not accessed even if they are available. Figure 9 shows the representation of this strategy.



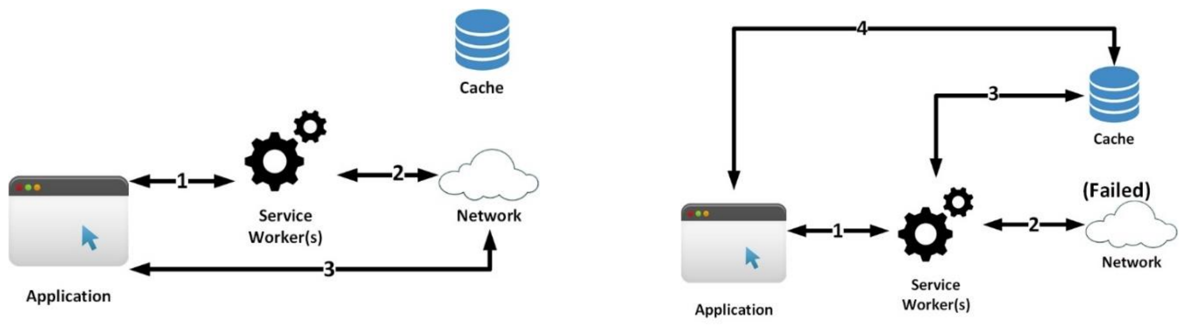
(a) Web Resources Already Cached (b) Web Resources Not Cached

Figure 5: Fastest Caching Strategy



(a) Web Resources Already Cached (b) Web Resources Not Cached

Figure 6: Cache First Caching Strategy



(a) Presence of Network Connection (b) Absence of Network Connection
Figure 7: Cache First Caching Strategy

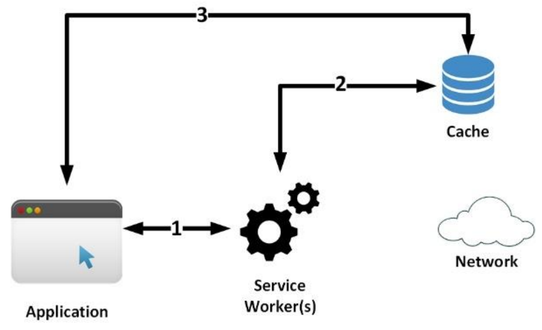


Figure 8: Cache Only Caching Strategy

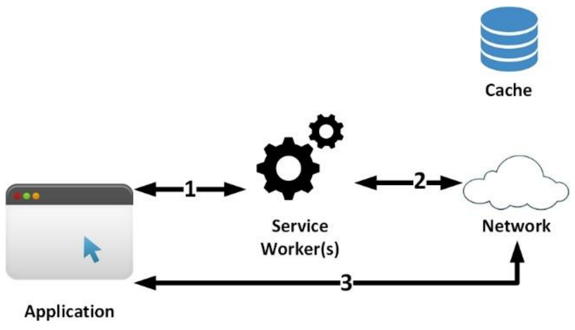


Figure 9: Network Only Caching Strategy

V. Conclusion

Various approaches can be adopted in the development of web applications by incorporating various caching mechanisms such as the Hypertext Transfer Protocol cache otherwise known as the browser cache, AppCache, and so on. So far, the research has been able to propose five (5) service worker caching strategies that can be adopted by web application developers to support offline accessibility, maintain content validity by supporting background content synchronization either in the presence or absence of internet connections. The integration of service workers into web applications can go a long way in improving the loading time as well as increase the number of daily active users of a web application.

VI. References

[1] O. Akeem and Y. Sun, “Mobile E-learning Support System for Secondary Schools in Nigeria,” in In 2018 1st IEEE International Conference on Knowledge Innovation and Invention (ICKII), 2018, pp. 262–265, doi: 10.1109/ICKII.2018.8569091.

[2] R. R. Eka, T. V. Utami, and K. D. Listriana, “Mobile Based Learning Development for Improving Quality of Nursing Education in Indonesia,” in In 2019 IEEE Conference on Sustainable Utilization and Development in Engineering and Technologies (CSUDET), 2019, pp. 39–44, doi: 10.1109/CSUDET47057.2019.9214755.

[3] A. Al-Hunaiyyan, S. Al-Sharhan, and R. Alhajri, “Prospects and Challenges of Mobile Learning Implementation: Kuwait HE Case Study,” 2017. Accessed: Mar. 14, 2021. [Online]. Available: https://www.researchgate.net/publication/346473760.

[4] S. S. Oyelere, J. Suhonen, and E. Sutinen, “M-learning: A New Paradigm of Learning ICT in Nigeria,” Int. J. Interact. Mob. Technol., vol. 10, no. 1, pp. 35–44, 2016, doi: 10.3991/ijim.v10i1.4872.

[5] J. Chen, “Service Worker Caching and HTTP Caching,” Web Dev, 2020. https://web.dev/service-worker-caching-and-http-caching/ (accessed Jul. 25, 2021).

[6] S. Meysam, “Caching in Web Applications,” Codementor, 2018. https://www.codementor.io/@meysamsamanpour/caching-in-web-applications-fz1gzizpa (accessed Jul. 25, 2021).

[7] M. I. Zulfa, R. Hartanto, and A. E. Permanasari, “Caching Strategy for Web Application – A Systematic Literature Review,” International Journal of Web Information Systems, vol. 16, no. 5. Emerald Group Holdings Ltd., pp. 545–569, Nov. 09, 2020, doi: 10.1108/IJWIS-06-2020-0032.

[8] E. Bidelman, “A Beginner’s Guide to Using the Application Cache,” HTML5Rocks, 2013. https://www.html5rocks.com/en/tutorials/appcache/beginner/ (accessed Jul. 25, 2021).

[9] B. James, “Problems with Application Cache,” Disqus, 2012. https://blog.jamesdbloom.com/ProblemsWithApplicationCache.html (accessed Jul. 25, 2021).

[10] M. Firtman, “Service Workers Replacing AppCache: A Sledgehammer to Crack a Nut,” Medium, 2016. https://medium.com/@firt/service-workers-replacing-appcache-a-sledgehammer-to-crack-a-nut-5db6f473cc9b (accessed Jul. 25, 2021).

[11] J. Posnick, “Preparing for AppCache Removal,” WebDev, 2021. https://web.dev/appcache-removal/ (accessed Jul. 25, 2021).

[12] L. Chris, “What is a Service Worker? Transform Your Web Site to an Instant Loading Powerhouse!,” Love2Dev, 2021. https://love2dev.com/blog/what-is-a-service-worker/ (accessed Jul. 25, 2021).

[13] J. Lee, H. Kim, J. Park, I. Shin, and S. Son, “Pride and Prejudice in Progressive Web Apps: Abusing Native App-Like Features in Web Applications,” in Proceedings of the ACM Conference on Computer and Communications Security, 2018, pp. 1731–1746, doi: 10.1145/3243734.3243867.

[14] A. D. Hume, Progressive Web Apps, 1st ed. New York: Manning Publications Co., 2018.

[15] T. Parbat, "Evaluation and Implementation of Progressive Web Application," Helsinki Metropolia University of Applied Sciences, 2018.

[16] A. Gambhir and G. Raj, "Analysis of Cache in Service Worker and Performance Scoring of Progressive Web Application," in 2018 International Conference on Advances in Computing and Communication Engineering, ICACCE 2018, 2018, pp. 294–299, doi: 10.1109/ICACCE.2018.8441715.

[17] R. S. Mishra, "Progressive Web App: Review," Int. Res. J. Eng. Technol., vol. 3, no. 6, pp. 3028–3032, 2016.

[18] I. Malavolta, G. Procaccianti, P. Noorland, and P. Vukmirovic, "Assessing the Impact of Service Workers on the Energy Efficiency of Progressive Web Apps," in In 2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft), 2017, pp. 35–45, doi: 10.1109/MOBILESoft.2017.7.

[19] caniuse.com, "Can I Use Service Workers," Fyrd, 2021. .