The Implementation of PWA (Progressive Web App) Technology in Enhancing Website Performance & Mobile Accessibility

Ahyar Muawwal[1*]
[1]Program Studi Sistem Informasi, STMIK Kharisma Makassar
[1]Jln. Baji Ateka No 20, Makassar, Indonesia
Email* : [1]ahyar@kharisma.ac.id

ARTICLE INFORMATION

Received on 15 May 2024
Revised on 04 June 2024
Accepted on 08 July 2024

Keywords :
Desktop
Mobile
Native applications
PWA
Website performance

ABSTRACT

The implementation of PWA as a necessary feature aims to provide added value and enhance website performance. This is intended to address several common issues in websites, such as limitations in displaying pages offline and the cost of developing native applications across various operating system platforms, both for desktop and mobile devices. Data collection methods involve literature studies and direct measurements using various tools. Testing conducted includes installation testing, evaluation of PWA criteria, performance, size of transferred resources, and offline mode. Components used in PWA include the web app manifest, service worker, and cache storage. PWA implementation involves creating a web app manifest, service worker registration, service worker configuration, adding script tags, creating specific routes within the website using Express.js, and PWA testing. Test results indicate that the website can be installed and used effectively on various types of devices, both mobile and desktop, and can be accessed offline or with unstable connections.

1. Introduction

The speed of a website's loading pages reflects its performance, which influences user satisfaction. The benchmarks for this include speed, website size, and ease of use (Muriyatmoko et al., 2022). In modern website development, slow performance and limited accessibility on mobile devices are significant challenges that need to be addressed. Slow performance can reduce user satisfaction and lead to a decrease in visitors, while limited accessibility on mobile devices can hinder a smooth user experience. Moreover, an increasing number of people are using smartphones and tablets to access the internet, not just desktop computers and notebooks. Therefore, websites need to be optimized for all these devices to provide the best user experience. Responsive web design offers flexibility for a website to adapt to each of these devices (Noorkaran Bhanarkar et al., 2023).

In addressing these challenges, Progressive Web App (PWA) technology has emerged as a promising solution. PWA technology allows developers to enhance website performance and increase its accessibility on mobile devices (Bhilare, 2019). With its ability to work offline and provide features such as push notifications, PWA offers a user experience similar to native apps without requiring installation from an app store. Additionally, PWAs also provide advantages in effective and fast development methods, accelerating the testing process as well (Mhatre et al., 2023).

However, despite its significant potential, there are still challenges in effectively implementing PWA in website development. Therefore, this research aims to investigate and analyze the implementation of PWA technology in improving website performance and mobile accessibility. Understanding the potential and challenges associated with PWA implementation is hoped to provide valuable insights for web developers to optimize user experience and enhance website competitiveness in the evolving digital era.

DOI: doi.org/10.17933/bpostel.v22i1.395

2. Literature review

2.1. Native Application

Websites typically adopt a rapid-release approach with the Minimum Viable Product (MVP) concept, where MVP focuses on essential features sufficient to meet users' fundamental needs. This allows developers to launch their products quickly without chasing perfection (Zott et al., 2024). In MVP-based product development, there are terms "must-have" and "should-have" to prioritize the features and functions of the product. Therefore, the researcher implemented PWA on the website to continue adding "should-have" features to the website, as well as to provide added value and enhance the quality of the website.

Native applications are applications developed specifically for use on one operating system platform, such as Android or iOS. Therefore, they can only run on one operating system and cannot be used across multiple platforms (Wahyurianto et al., 2018). Therefore, the researcher aims to implement PWA on the existing website to address several issues, such as the cost of development and maintenance associated with creating native applications for various operating system platforms, enhancing website performance, particularly in terms of page loading speed as testing results using Google Lighthouse indicate the necessity of implementing cache policy, and overcoming limitations of the website to display specific static pages when the user is offline. The objective of this study is to implement Progressive Web Apps (PWA) technology on the website using Express.js so that it can be used in the form of a native application on smartphones via the smartphone home screen without the need for installation through the Play Store or app store, and can be used when the user is offline.

2.2. Progressive Website Application (PWA)

PWA is a web-based development technology that enables a website to have experiences similar to using a native mobile application, thus providing users with an impressive experience (Nurwanto, 2019). PWA technology was developed by Google and other browser and mobile developers to simplify the creation of multi-platform applications (leha soleha et al., 2019). By utilizing service workers, web app manifest, and cache API, PWA offers several features such as offline mode, installation on the home screen (Add To Home Screen), push notifications, background sync, and splash screens (Aminudin et al., 2019).

Research related to the implementation of Progressive Web app technology on a website has been conducted by several researchers before. In the study (Aripin et al., 2021) conducted by Aripin, Somantri titled "Implementation of Progressive Web Apps (PWA) on Student E-Portfolio Repository." The study (Riady et al., 2019) conducted by Riady, Palit, and Andjarwirawan titled "Progressive Web App-Based E-Learning Application on Indonesian Apologetics.". Meanwhile, research conducted by Bahari, Sumaryana (Bahari et al., 2019) titled "Implementation of Progressive Web Apps in Job Vacancy Applications for Lecturers at Perjuangan University". And also, in the research (Haryanto et al., 2021) conducted by Haryanto, Elsi titled "Performance Analysis of Progressive Web Apps in the Shopee Application." Additionally, the study (Aslan et al., 2022) conducted by Aslan, Bahtiar, Sudianto titled "Development of the Hamzanwadi University Faculty of Engineering Website Based on Progressive WEB APP (PWA)." Similarly, the research conducted by Joarno, Fajar, Yunus (Phie Joarno et al., 2022) titled "Implementation of Progressive Web Apps on the Gethelp Website Using Next.js." In this study, a different framework, Express.js, was used compared to previous research. Additionally, installable testing was conducted after PWA implementation, fulfilling PWA criteria, website performance, size of transferred resources, and offline mode were evaluated both before and after PWA implementation.

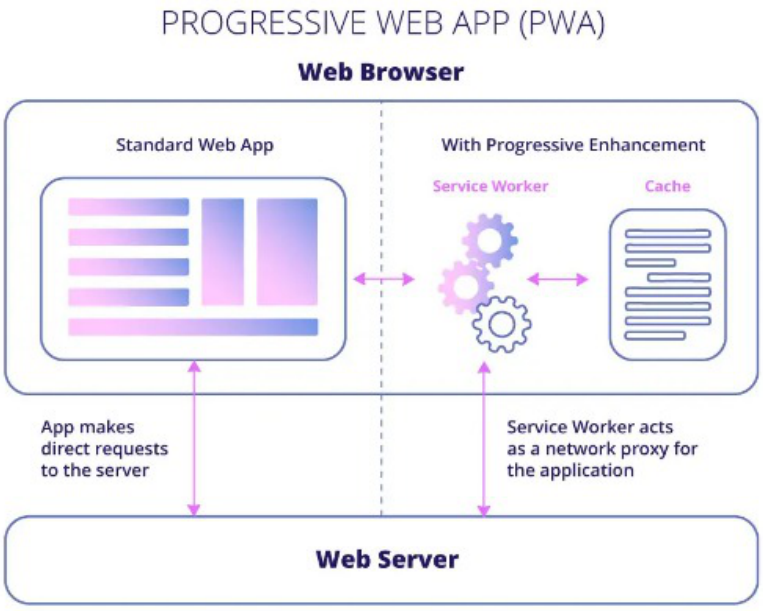

Figure 1. Architecture of PWA Implementation (source : https://mobidev.biz)

2.3. Web App Framework (Express.js)

Express.js is a web app framework built on top of Node.js (JavaScript Runtime Environment), which falls into the category of the most popular frameworks in the Node.js world because it is known for being minimalist and flexible when it comes to creating a web application (Mardan, 2014). The considerations for implementing PWA on a website using Express.js include ease of managing static website templates as it integrates with various template engines, server-side rendering to speed up website page display, ensuring SEO friendliness and good website performance, ease of setup and deployment of projects, featuring API development and support for third-party middleware, and being unopinionated, which means it is flexible in terms of customizing the application folder structure and middleware functions.

3. Method

3.1. Types of Data

The type of data used in this study is quantitative data, obtained from experimental results in the form of direct measurements of the website using several tools such as Browser Edge and Chrome, PWABuilder, Google Lighthouse, GTMetrix, Chrome DevTools. This data includes installable testing after PWA implementation, fulfillment of PWA criteria, website performance, size of transferred resources, and offline mode both before and after PWA implementation. In addition to this primary data, secondary data sources were obtained from literature studies or library research consisting of relevant journal articles related to the research on PWA implementation architecture, components, and main features of PWA such as offline mode, add-to-home screen, cache storage usage, service worker life cycle, and others.

3.2. Research Stages

During the system testing phase, various tests were conducted, such as installable testing using various types of mobile devices and browsers, fulfillment testing of PWA criteria using Lighthouse, which consisted of two parameters (installable, PWA optimized), and manual testing. Meanwhile, PWABuilder

consists of three parameters (Service worker, Manifest, Security), with several criteria tested for each parameter. Performance testing using Lighthouse focused on six testing variables, while GTMetrix focused on 12 testing variables. As for testing the size of transferred resources and website offline mode, browser DevTools were used. The test results were then processed into tables for comparison and analysis before and after PWA implementation, enabling conclusions to be drawn regarding the research conducted. The research stages are presented in Figure 1.

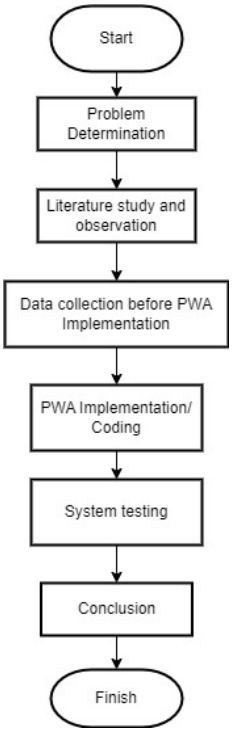

Figure 2. Research Stage

4. Result and Discussion

4.1. PWA Implementation / Coding

There are several stages in implementing PWA on the website using Express.js, namely:

4.1.1. Addition of Web App Manifest

The Web App Manifest is a JSON (JavaScript Object Notation) file that provides detailed information to the browser about the web application and its behavior when installed. In this stage, the author creates a Web App Manifest named manifest.json to store all the detailed information about the website, such as the application name, icons, description, theme color, screen orientation, and others, enabling the website to have add-to-home-screen and splash screen features when the web application is opened. The manifest.json file can be viewed in Figure 3.

After creating the manifest.json file, the next step is to add a link tag to the website's header. This is done to load the manifest.json file and inform the browser that the website adopts PWA technology and is installable, as shown in Figure 4.

Figure 3. File manifest.json

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

Figure 4. Calling manifest.json in the Website Header

Furthermore, it is necessary to create a specific route to handle requests for /manifest.json, because the manifest.json file is placed in the root directory of the website application folder and not in the static (public) folder as determined using Express.js's built-in middleware. Therefore, the route to handle requests for /manifest.json can be seen in Figure 5.

```
app.get('/manifest.json', (req, res) => {
    res.sendFile(path.join(__dirname, 'manifest.json'));
});
```

Figure 5. Request Route manifest.json

4.1.2. Addition of Service Worker

Web App Manifest is a JSON (JavaScript Object Notation) file that provides information about the Service Worker to the browser. It is a JavaScript script that runs in the background of the user's browser and serves as the gateway for PWA features such as push notifications, background sync, resource caching, offline mode, and more. In this stage, the author creates a file named swregist.js to configure the service worker registration. Next, the author adds a script tag so that the swregist.js file can be loaded, and the service worker can be registered on the website using the file located at the route /sw.js, as shown in Figure 6 and Figure 7.

```
if ('serviceWorker' in navigator) {
    window.addEventListener('load', function () {
        navigator.serviceWorker.register('/sw.js').then(function (registration) {
            console.log('ServiceWorker registration successful with scope: ',
            registration.scope);
        }, function (err) {
            console.log('ServiceWorker registration failed: ', err);
        });
    });
}
```

Figure 6. Service Worker Registration Configuration

Next, the process can be carried out by calling the script on the website to input it into the service worker so that the service worker can better handle the existing data.

```
<script src="/js/swregist.js"></script>
```

Figure 7. Calling Script swregist for Service Worker Registration

Next, create a new file named sw.js in the root directory containing all the service worker configurations for the website. After configuring the sw.js file, the author creates a specific route to handle requests for /sw.js because the sw.js file is placed in the root directory and not in the static (public) folder, as determined by Express.js.

```
var CACHE_NAME = 'frizfoo-cache-v1';
var urlsToCache = [
    '/',
    '/offline',
    '/css/style.css',
    '/js/navbarscript.js',
    '/js/scrolltotop.js',
    '/js/subscribeform.js',
    '/img/logo.webp',
    '/img/offline.gif',
];

self.addEventListener('install', function (event) {
    event.waitUntil(
        caches.open(CACHE_NAME)
            .then(function (cache) {
                console.log('Opened cache');
                return cache.addAll(urlsToCache);
            })
    );
});

self.addEventListener('fetch', function (event) {
    event.respondWith(
        caches.match(event.request)
            .then(function (response) {
                if (response) {
                    return response;
                }

                return fetch(event.request).then(
                    function (response) {
                        if (!response || response.status !== 200 || response.type !== 'basic') {
                            return response;
                        }
                        var responseToCache = response.clone();
                        caches.open(CACHE_NAME)
                            .then(function (cache) {
                                cache.put(event.request, responseToCache);
                            });
                        return response;
                    }
                )
            }).catch(function () {
                return caches.match('/offline');
            })
    );
});

self.addEventListener('activate', function (event) {
    var cacheAllowlist = CACHE_NAME;
```

Figure 8. Service Worker Configuration File

The details of the service worker configuration file and the route to handle requests for /sw.js can be seen in Figure 7 and Figure 8.

4.2. System Testing

There are several types of system testing conducted to ensure the success of PWA implementation on the website, namely:

4.2.1. Installable Testing

Installable testing is conducted on both mobile and desktop devices to ensure that the website can be installed on various user devices. Desktop installable testing is performed using Google Chrome and Microsoft Edge browsers. In contrast, mobile installable testing is conducted on five different devices, including iPhone XS, Xiaomi Redmi 10, Samsung Galaxy A6, Samsung Galaxy A30, and Vivo V20. Mobile installable and desktop installable testing are conducted on devices with different platforms to observe the installation capability of the website that has adopted PWA technology. It was found that the website has been successfully installed on various mobile and desktop devices, indicating the success of the website in mobile installable and desktop installable testing, as shown in Figure 9.

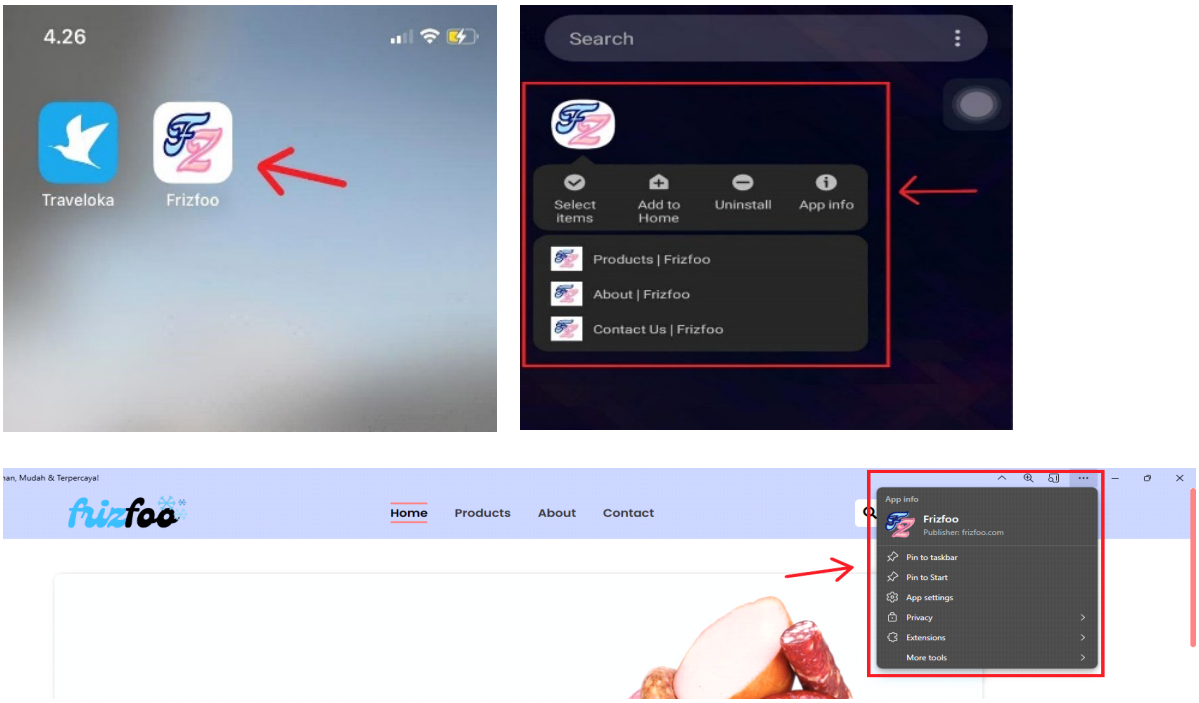

Figure 9. Testing the Installation of the website on iPhone, Android, and Desktop

4.2.2. Performance Testing

This performance testing is conducted to assess the speed of the website, implementation on search engines using SEO, and accessibility using several PWA testing tools, Lighthouse and GT Metrix, where the results obtained are as follows:

a. Lighthouse

The results of website performance testing, both before and after PWA implementation using Lighthouse, can be seen in the image below:

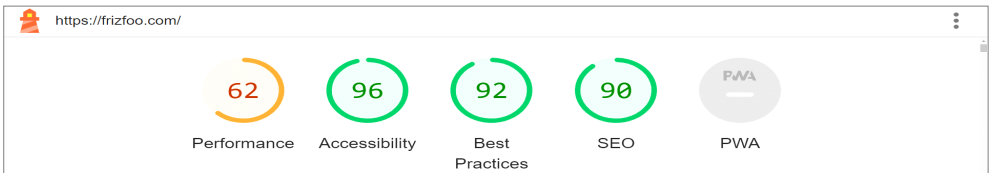

Figure 10. Before PWA Implementation

It can be seen that in the testing before implementation, the performance score was only 62, indicating the need for improvement in performance and the implementation of PWA.

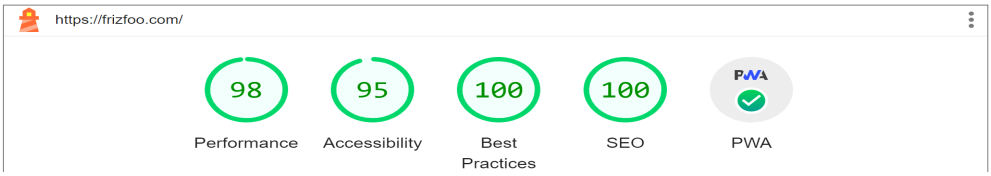

Figure 11. After PWA Implementation

The image above shows that after implementation, the application experienced a 36% improvement in performance and also scored 100% in SEO and best practices. The details of the website performance testing aspects using Lighthouse are shown in Table 3.

Table 1. Results of Website Performance Testing using Lighthouse

No.	Variable	Before PWA Implementation	After PWA Implementation
1.	Performance	62%	98%
2.	First Contentful Paint	1.5s	0.9s
3.	Total Blocking Time	220ms	0ms
4.	Speed Index	2.3s	1.0s
5.	Largest Contentful Paint	2.8s	0.9s
6.	Cumulative Layout Shift	0.153	0.01

Based on table 1, the results of website performance testing using Lighthouse after implementing PWA and several optimizations show a significant improvement. The performance score increased from 62% to 89%, and various testing variables, as seen in Table 3, experienced significant improvements. These results indicate that implementing PWA technology on the website can enhance website performance.

b. GTMetrix

In addition to using Lighthouse, this research also utilized GTMetrix to observe the website performance testing results, both before and after implementing PWA, as another measurement tool. This approach ensures that the testing results are more varied and not solely dependent on one tool.

Figure 12. Testing with GTMetrix Tools Before PWA Implementation

Before implementing PWA, the results show a score of 63% and a loading time of 0.31 seconds, with a Grade D indicating that the website needs improvement in terms of loading time and performance. Let's see the results after implementing PWA, as shown in Figure 13.

Figure 13. Testing with GTMetrix Tools After PWA Implementation

In general, there is an improvement in performance, with the website now only requiring 0.08 seconds to load. For more details, here are the specifics of the website performance testing aspects using GTMetrix, as shown in Table 2:

Table 2. Results of Website Performance Testing Using GTMetrix

No.	Variable	Before PWA Implementation	After PWA Implementation
1.	Performance	63% (Grade D)	99% (Grade A)
2.	First Contentful Paint	1.5s	174ms
3.	Time to Interactive	3.3s	488ms
4.	Speed Index	2.3s	432ms
5.	Total Blocking Time	77ms	63ms
6.	Largest Contentful Paint	2.7s	572ms
7.	Cumulative Layout Shift	0.31	0.08
8.	Redirect Duration	0ms	0ms
9.	Time to First Byte (TTFB)	667ms	40ms
10.	Fully Loaded Time	7.5s	1.6s
11.	Total Page Size	2.34 MB	2.30 MB
12.	Total Page Requests	52	53

Based on table 2, the results of website performance testing using GTMetrix after implementing PWA and several optimizations show a significant improvement. The performance score increased from 63% to 99%, and there was a significant improvement in the speed of loading the website pages. Before implementing PWA, the website required 7.5 seconds to load a full page (Fully Loaded Time), but after implementing PWA, it only took 1.6 seconds.

c. Testing Size Transferred Resources

This testing involves directly observing the browser used to access the website, then assessing the size of the website data transferred and the duration from when the website is opened until loading is complete. The results of Size Transferred Resources testing on the website using browser DevTools in a cache-disabled state are as follows:

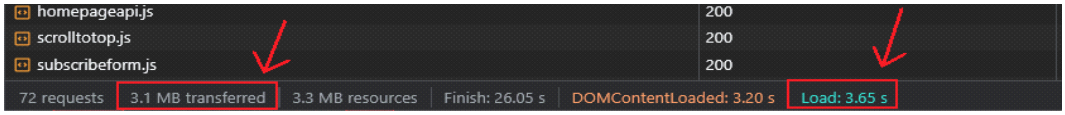

Figure 14. Size Transferred Resources Data Before PWA Implementation

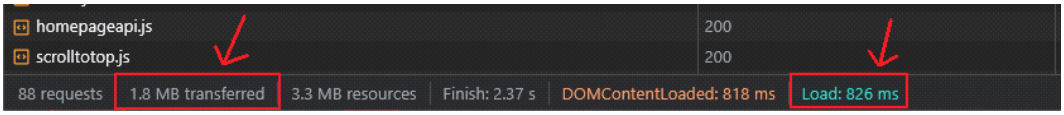

Figure 15. Size Transferred Resources Data After PWA Implementation

The measurement results of size transferred resources on the website after implementing PWA show a value of 1.8 MB, compared to the previous size of 3.3 MB. Thus, users need to download approximately 1.8 MB each time they load that webpage. Additionally, the website loading time has also improved, decreasing from 3.65 seconds to 826 milliseconds after implementing PWA. Therefore, PWA implementation can reduce the size of transferred resources and accelerate website loading time.

d. Offline Content Testing

This test directly observes the browser behavior when subjected to offline conditions or when the network connection is disconnected. The Offline Mode Testing is conducted to ensure that the

website has successfully adopted one of the PWA features, which is Offline Mode. The test is performed using the network section of the browser DevTools by switching the throttling option to offline or when the device is disconnected from the internet.

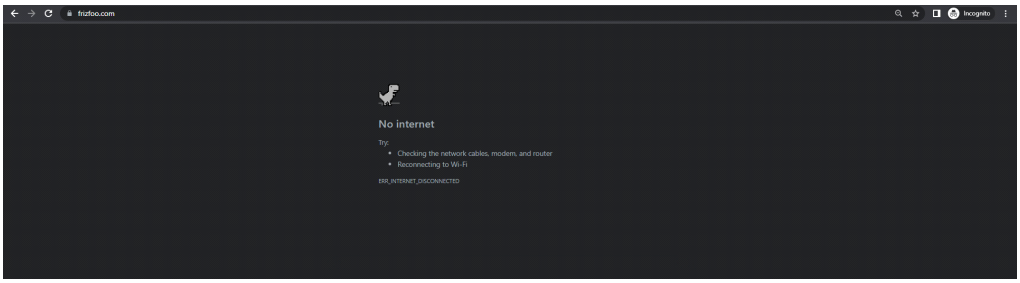

Figure 16. Offline Mode Testing Before PWA Implementation

As seen in the figure 16, during website testing, when the network is disconnected, it is evident that before implementing PWA, the website did not display any content at all. This becomes a drawback when users intend to access initial information from the application. To observe the difference when PWA is implemented on the website, the results can be seen below.

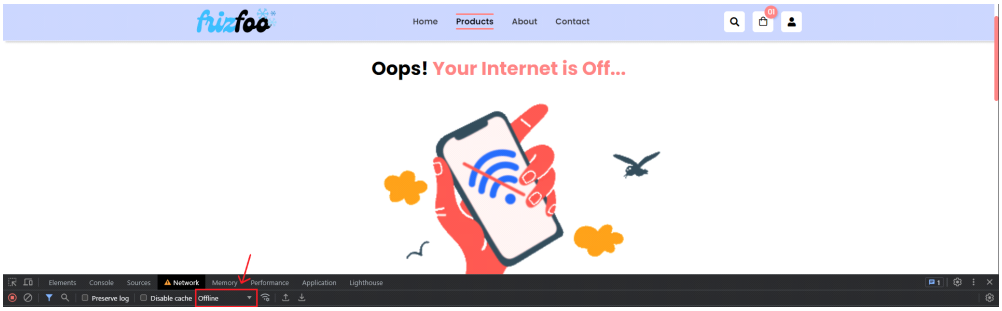

Figure 17. Offline Mode Testing Before PWA Implementation

The results of the offline mode testing on the website were successful because the website can be accessed when there is no internet connection available or when the connection is unstable. Additionally, it can display specific static pages indicating that the user is offline, thus indicating the success of the website in the offline mode testing.

5. Conlusion

Based on the research conducted by the author, several conclusions can be drawn as follows:

1. In implementing PWA on the website using Express.js, a Web App Manifest is required to provide detailed information to the browser about the web application, and a Service Worker connected to the main website page is needed to be registered as a background process in the user's browser. Additionally, if the files required for PWA implementation are not located in the static (public) folder, a specific route needs to be created in Express.js to handle those requests. This is necessary because Express.js uses built-in middleware to determine which static folder can be accessed to use its files.
2. By implementing PWA on the website, the website can be installed and accessed like a native mobile application via the smartphone's home screen or a desktop application on the user's device. This allows users to access the website without needing to use a browser or install it through the Play Store or App Store.

3. PWA technology has been successfully implemented on the website and meets all the criteria needed for a website to adopt PWA technology. After testing the PWA criteria after implementing PWA with Lighthouse, I obtained a score of 10/10, where all the PWA criteria recommended by Lighthouse were met. Meanwhile, testing the PWA criteria using PWABuilder obtained a total score of 25, with 25 testing criteria being met. These results even exceed some of the criteria recommended by PWABuilder, such as the Service Worker (1 Highly Recommended criterion), Manifest (4 Required criteria), and Security (3 Required criteria) that must be met for a website to be considered adopting PWA technology.
4. There has been an improvement in website performance after implementing PWA. Performance testing using Lighthouse yielded results of 62% before PWA implementation, which increased to 98% after PWA implementation. Meanwhile, performance testing using GTMetrix yielded results of 63% (Grade C) before PWA implementation, which increased to 99% (Grade A) after PWA implementation. Various optimizations were carried out by the author to improve website performance, including converting images to the .webp format, storing necessary static files in cache storage, updating the CDN (Content Delivery Network) Font Awesome, removing unused CDN script tags, adding the async attribute to script tags, minifying some CSS and JavaScript files, and so on.
5. There has been a decrease in the size of transferred resources and faster website loading time after implementing PWA. Testing size transferred resources using browser DevTools showed a value of 3.1 MB before PWA implementation, which then decreased to 1.8 MB after PWA implementation. Additionally, the website loading time has become faster, decreasing from 3.65 seconds before PWA implementation to 826 ms after PWA implementation, allowing the website to load faster and be used by users more quickly
6. A website that has adopted PWA technology can utilize the offline mode feature, allowing the website to be accessed in offline conditions or unstable internet connection situations. Thus, if users access the website offline, the website has the ability to display specific static pages indicating that the user is offline.

6. Acknowledgements

All praise be to Allah SWT for His abundant blessings and grace, enabling the author to complete this article. The author would like to express gratitude to all friends, lecturers, and academic staff of STMIK Kharisma Makassar, especially the Information Systems study program at STMIK Kharisma Makassar.

References

Aminudin, A., Basren, B., & Nuryasin, I. (2019). Perancangan Sistem Repositori Tugas Akhir Menggunakan Progressive Web App (PWA). *Techno.Com*, *18*(2), 154–165. doi: 10.33633/tc.v18i2.2309

Aripin, S., & Somantri, S. (2021). Implementasi Progressive Web Apps (PWA) pada Repository E-Portofolio Mahasiswa. *Jurnal Eksplora Informatika*, *10*(2), 148–158. doi: 10.30864/eksplora.v10i2.486

Aslan, I., Bahtiar, H., & Sudianto, A. (2022). Pengembangan Website Fakultas Teknik Universitas Hamzanwadi Berbasis Progressive WEB APP (PWA). *Infotek : Jurnal Informatika Dan Teknologi*, *5*(1), 99–107. doi: 10.29408/jit.v5i1.4448

Bahari, C. C. B., & Sumaryana, Y. (2019). Penerapan Progressive Web Apps Pada Aplikasi Lowongan Pekerjaan Dosen Universitas Perjuangan. *Informatics and Digital Expert (INDEX)*, *1*(1). doi: 10.36423/ide.v1i1.285

Bhilare, A. (2019). Progressive Web App (PWA) for Organization System. *International Journal for Research in Applied Science and Engineering Technology*, *7*(5), 610–613. doi: 10.22214/ijraset.2019.5104

Direktorat Spektrum Frekuensi Radio dan Orbit Satelit. (2005). *Perencanaan Frekuensi TV Siaran UHF di Indonesia*. Jakarta.

Haryanto, D., & Saputra Elsi, Z. R. (2021). Analisis Performance Progressive Web Apps Pada Aplikasi Shopee. *Jurnal Ilmiah Informatika Global*, *12*(2). doi: 10.36982/jiig.v12i2.1944

ITU-D ICT Statistics. (2014). *Mobile-cellular telephone subscriptions*. Www.Itu.Int.

leha soleha, Budiman, E., & Wati, M. (2019). *Pengembangan Progressive Web Application Portal Program Studi Teknik Informatika Berbasis Restful API*. Retrieved from https://api.semanticscholar.org/CorpusID:213685401

Mardan, A. (2014). Starting with Express.js. In Pro Express.js (pp. 3–14). Berkeley, CA: Apress. doi: 10.1007/978-1-4842-0037-7_1

Mhatre, A., & Mali, S. (2023). Progressive Web Applications, a New Way for Faster Testing of Mobile Application Products. *2023 3rd Asian Conference on Innovation in Technology (ASIANCON)*, 1–6. doi: 10.1109/ASIANCON58793.2023.10269806

Muriyatmoko, D., & Aziz Musthafa. (2022). Website Performance Testing Using Speed Testing Model: A Case of Reputable Indonesian Journals. *Teknik: Jurnal Ilmu Teknik Dan Informatika*, *2*(1), 40–45. doi: 10.51903/teknik.v2i1.120

Noorkaran Bhanarkar, Aditi Paul, & Dr. Ashima Mehta. (2023). Responsive Web Design and Its Impact on User Experience. *International Journal of Advanced Research in Science, Communication and Technology*, 50–55. doi: 10.48175/IJARSCT-9259

Nurwanto, N. (2019). Penerapan Progressive Web Application (PWA) pada E-Commerce. *Techno.Com*, *18*(3), 227–235. doi: 10.33633/tc.v18i3.2400

Phie Joarno, R. J., Mohammad Fajar, & Arfan Yunus. (2022). Implementasi Progressive Web Apps Pada Website GetHelp Menggunakan Next.js. *KHARISMA Tech*, *17*(2), 1–15. doi: 10.55645/kharismatech.v17i2.219

Riady, J., Palit, H. N., Andjarwirawan, J., & Petra. (2019). Aplikasi E-Learning Berbasis Progressive Web App Pada Apologetika Indonesia. *Jurnal Infra Petra*, 1–5.

Wahyurianto, F., Arwani, I., & Soebroto, A. A. (2018). Pembangunan Aplikasi Informasi Kesehatan Masyarakat Kota Malang Berbasis Mobile Native Android. *Jurnal Pengembangan Teknologi Informasi Dan Ilmu Komputer*, *3*(1 SE-), 416–425. Retrieved from https://j-ptiik.ub.ac.id/index.php/j-ptiik/article/view/4126

Zott, C., & Amit, R. (2024). Business Models and Lean Startup. *Journal of Management*, *29*(3), 508–509. doi: 10.1177/01492063241228245