Implementation of Progressive Web Application (PWA) on Mobile Devices: A Case Study of the LAYAR Application

Wikussangker Wirabumi[1], Rina Yulius[2], Muchamad Fajri Amirul Nasrullah[3]
Politeknik Negeri Batam, Jl. Ahmad Yani, Batam 29461, Indonesia[1], [2], [3]
wikuswb0212@gmail.com[1], rinayulius@polibatam.ac.id[2], fajri@polibatam.ac.id [3]

***Abstract*— This study aims to evaluate the impact of implementing Progressive Web Application (PWA) technology on the LAYAR platform, a web-based administrative system for RT/RW communities in Indonesia. The research addresses key limitations such as load time, installability, and offline access on mobile devices, which are the primary access points for users. The implementation involved PWA components including service workers, a web app manifest, and caching strategies. A quantitative experimental approach was employed using a one-group pretest-posttest design, with system performance measured through Google Lighthouse and Chrome DevTools under various network conditions. The results show an increase in Lighthouse performance score from 87% to 93%, a reduction in Cumulative Layout Shift from 0.15 to 0.01, and successful offline access to static content. Under fast 4G conditions, load time decreased by nearly 50%, while service worker caching contributed to performance consistency across different networks. These findings demonstrate that PWA integration can significantly improve performance, offline functionality, and installability—offering practical advantages for public service applications in mobile-first environments. Future research is recommended to involve real users in usability testing and investigate the adoption of PWA in other sectors such as healthcare and education.**

Keywords*— *Progressive Web Application, LAYAR Aplication, Google Lighthouse, Chrome DevTools

***Abstrak*—Penelitian ini bertujuan untuk mengevaluasi dampak penerapan teknologi Progressive Web Application (PWA) pada platform LAYAR, sebuah sistem administrasi berbasis web untuk komunitas RT/RW di Indonesia. Penelitian ini menangani keterbatasan seperti waktu muat (load time), kemampuan instalasi, dan akses offline pada perangkat mobile yang merupakan titik akses utama bagi pengguna. Implementasi dilakukan dengan menambahkan komponen PWA seperti service worker, web app manifest, dan strategi caching. Metode yang digunakan adalah pendekatan eksperimen dengan data kuantitatif menggunakan desain pretest-posttest satu kelompok, di mana performa sistem diukur menggunakan Google Lighthouse dan Chrome DevTools pada berbagai kondisi jaringan. Hasil pengujian menunjukkan peningkatan skor performa Lighthouse dari 87% menjadi 93%, penurunan nilai Cumulative Layout Shift dari 0.15 menjadi 0.01, serta keberhasilan akses konten statis secara offline. Pada kondisi jaringan 4G cepat, waktu muat aplikasi menurun hampir 50%, sementara mekanisme caching melalui service worker berkontribusi terhadap konsistensi performa di berbagai jaringan. Temuan ini menunjukkan bahwa integrasi PWA dapat secara signifikan meningkatkan performa, fungsionalitas offline, dan kemampuan instalasi—memberikan manfaat praktis untuk aplikasi layanan publik dalam lingkungan yang mengutamakan perangkat mobile. Penelitian selanjutnya disarankan untuk melibatkan pengguna secara langsung dalam pengujian kegunaan serta mengeksplorasi adopsi PWA di sektor lain seperti kesehatan dan pendidikan.**

Kata Kunci*— *Progressive Web Application, Aplikasi LAYAR, Google Lighthouse, Chrome DevTools

I. INTRODUCTION

The rapid advancement of information technology has led to the emergence of various applications designed to simplify daily activities. In this digital era, data access speed has become a crucial factor for enhancing operational efficiency across many sectors [1]. As mobile devices become the primary means of internet access for most users, the demand for lightweight, fast, and accessible applications continues to grow. In response to these evolving needs, Progressive Web Application (PWA) technology has emerged as a promising solution.

PWA is a modern approach that combines the strengths of web and native applications. It enables web apps to deliver near-native performance while running in the browser, offering features like offline access, fast load times, and the ability to be launched from a device's home screen without needing installation via app stores [2][3]. According to Google, PWAs can improve conversion rates by up to 50%, increase loading speeds by 2–3 times, and reduce data consumption by up to 90% compared to native apps [4]. These benefits are especially critical given that 53% of users abandon websites that take more than three seconds to load [5], and 92.1% of global internet users now access the web primarily via mobile devices [6].

In Indonesia, this mobile-first trend is even more pronounced. Data from GS StatCounter shows that 63.09% of internet traffic in Indonesia comes from mobile devices, with desktop usage trailing behind at 36.39% [7]. Furthermore, the number of mobile connections in Indonesia exceeds the total population, reaching 353.8 million [8]. These statistics highlight the strategic importance of optimizing web applications for mobile platforms, particularly when designing digital services aimed at local communities.

LAYAR, a web-based administrative platform for

p-ISSN 2301-7988, e-ISSN 2581-0588
DOI : 10.32736/sisfokom.xx.xx, Copyright ©2020
Submitted : , Revised : , Accepted : , Published :

neighborhood (RT/RW) services to support the digitization and simplification of administration at the neighborhood community level. This application, designed by the author and his team in the previous semester, is a relevant case study in this context. Preliminary assessments using Google Lighthouse revealed performance issues, particularly on mobile devices, where the app recorded a performance score of 87%. Further analysis pointed to areas needing improvement, such as a Speed Index of 4.2 seconds, a Largest Contentful Paint of 2.6 seconds, and a Cumulative Layout Shift (CLS) score of 0.15, indicating suboptimal visual stability.

Baseline performance testing using Chrome DevTools under different network conditions provided further insights. On 3G, the app loaded in 2.15 seconds with 26 requests and transferred 4.5 kB of data. On slow 4G, load time was 639 ms, while fast 4G yielded an interactive time of 495 ms. However, testing under offline conditions showed the application failed to function without internet, preventing users from accessing essential forms like Pelaporan (reporting templates) and Surat Pengajuan (application letters). This exposes a limitation in user accessibility and convenience particularly for residents who depend on these forms for local administrative services.

These findings present a clear opportunity to enhance the application by implementing PWA features, especially on mobile. Offline functionality would allow residents to access key documents without visiting the RT office to get the form, improving service delivery and reducing dependency on constant connectivity.

This study aims to evaluate the implementation of PWA technology on the mobile version of the LAYAR application, focusing on key enhancements such as offline access, installability, and performance optimization. The development integrates Service Workers, a Web App Manifest, and caching strategies.

By focusing exclusively on mobile where user traffic and usage needs are highest in Indonesia this research contributes to the broader understanding of PWA's potential in public service digital platforms and offers insights for practical improvements in mobile-first web application development.

II. PREVIOUS STUDIES

Muawwal [9] conducted a study that aimed to improve website performance by addressing offline access limitations and the high cost of native app development. The results indicated that PWA-based websites can be installed and function well across various devices while remaining accessible under unstable network conditions.

Similarly, Kurniawan [10] evaluated PWA features and performance on mobile platforms, including metrics such as speed, accessibility, SEO, and page size. His findings suggest that PWAs effectively bridge the gap between web and native applications by enhancing load times, optimizing data efficiency, and integrating native-like features without compromising overall performance. Unlike Kurniawan's study, the present research focuses on offline access and performance optimization on low-end devices through the LAYAR application.

In addition, Adelin et al. [11] found that PWA adoption led to 39% performance improvement and reduced page load times from 12 seconds to just 2 seconds. Although a minor drop in SEO scores was observed, overall the best practices improved, reflecting substantial site performance enhancements. In contrast, the LAYAR study additionally considers data efficiency and offline accessibility alongside performance metrics.

Furthermore, Haryanto et al. [12] examined PWA performance using indicators such as page speed, file size, and the number of requests. Their results indicated that smaller file sizes and fewer requests improved page speed, achieving a good YSlow score. However, full-page load times remained relatively slow, although this did not significantly impact overall application performance. While this research emphasized the technical aspects of PWA in an e-commerce context, the current study highlights user experience within a public administration setting through the LAYAR platform.

III. METHOD

This study employed quantitative data collection methods through experimental testing and direct measurement of website performance metrics. The primary data was gathered using web development tools including Chrome browser, Google Lighthouse, and Chrome DevTools to evaluate multiple aspects of the web application both before and after Progressive Web App (PWA) implementation. The measured parameters encompassed installability functionality, PWA compliance standards, website performance metrics, resource transfer sizes, and offline functionality capabilities.

Supplementary data was acquired through comprehensive literature review, examining scholarly articles and journals focused on PWA architectural frameworks, core components, and essential functionalities including offline accessibility, home screen installation features, caching mechanisms, service worker management, and related PWA technologies. This approach provided both empirical performance data and theoretical foundation necessary for analyzing the effectiveness of PWA implementation on the target web application.[9]

A. Research Stages

The system evaluation phase utilized two primary testing tools to assess PWA implementation effectiveness. Google Lighthouse served as the primary assessment tool for measuring four critical performance indicators: performance scores, accessibility compliance, best practice adherence, and search engine optimization (SEO) ratings. These metrics provided comprehensive evaluation of overall application quality and user experience standards.

Chrome DevTools functioned as the secondary evaluation instrument, specifically employed for network performance analysis and offline functionality testing. This tool enabled simulation of various network conditions to measure page load times across different connection speeds, providing insights into application responsiveness under diverse network scenarios. Additionally, Chrome DevTools facilitated offline mode testing to verify PWA functionality

p-ISSN 2301-7988, e-ISSN 2581-0588
DOI : 10.32736/sisfokom.xx.xx, Copyright ©2020
Submitted : , Revised : , Accepted : , Published :

when internet connectivity is unavailable.

The dual-tool approach ensured comprehensive assessment of both quantitative performance metrics through Lighthouse scoring and practical functionality verification through DevTools network simulation and offline testing. Test results from both tools were systematically documented and organized into comparative tables to analyze performance differences between pre- and post-PWA implementation phases. This methodology provided thorough evaluation of PWA implementation impact on both measurable performance standards and real-world user experience scenarios, enabling comprehensive analysis of the application's enhanced capabilities.

B. *Testing Environment*

The testing was conducted in a controlled environment with the these specifications:

- Hosting: Hostinger
- Framework: Laravel v8.75
- Database: MySQL v8.0
- Browser: Google Chrome

C. *Testing Tools*

1. *Google Lighthouse*

Google Lighthouse is an open-source automated tool designed to enhance the quality of web pages. It can be executed on any webpage, whether public or requiring authentication. The tool provides audits across several key categories, including performance, accessibility, Progressive Web App (PWA) compliance, SEO, and more. Lighthouse can be run via Chrome DevTools, the command line, or as a Node module.

In this study, Google Lighthouse was selected due to its ability to deliver comprehensive evaluations through quantitative scoring (ranging from 0 to 100) alongside intuitive visual reports that facilitate in-depth analysis. These scores allow for an objective comparison of application performance before and after PWA implementation. As an official tool developed by Google, Lighthouse adheres to standardized and widely recognized audit criteria, making it a trusted reference among developers of progressive web applications [16].

2. *Chrome DevTools*

Chrome DevTools (CDT) is a set of development tools integrated directly into the Chrome browser, widely utilized by web developers during the web application development process [17]. It serves as a comprehensive platform for diagnostics and optimization, enabling detailed analysis of web application performance and user experience.

The rationale for selecting Chrome DevTools in this study lies in its ability to provide real-time insights into how the application interacts with the network, cache, and individual page elements—critical aspects in evaluating Progressive Web Application (PWA) performance. Features such as Network and Performance allow researchers to simulate various network conditions and verify whether the service worker and caching mechanisms operate correctly, especially during offline scenarios. With its capabilities to monitor page load time, data transfer size, and the number of requests, Chrome DevTools is an essential instrument for demonstrating the effectiveness of PWA implementation in the LAYAR application.

The combined use of Google Lighthouse and Chrome DevTools in this study ensures a more holistic and complementary evaluation of application performance. While Google Lighthouse provides standardized and benchmarked scores across key web quality indicators, Chrome DevTools offers granular, real-time diagnostics that reveal how the application behaves under various network and system conditions. This dual-tool approach not only strengthens the reliability of performance assessments but also aligns with best practices in PWA evaluation, particularly for identifying optimization opportunities in resource loading, caching behavior, and offline functionality. As noted by Faizin et al [18]. automated performance tools are highly effective for web-based applications, offering prescriptive insights that support developers in iteratively improving speed, accessibility, and usability.

D. *Testing Devices*

The testing was conducted on four mobile devices with different specifications to ensure consistency of results in the installability assessment:

TABLE I. TESTING DEVICES

Device	Operating System	OS Version	RAM	Processor
Samsung A55	Android	13	8GB	Exynos 1380
Infinite Note 40	Android	11	4GB	MediaTek Helio G85
iPhone 12 Pro	iOS	16.5	6GB	A14 Bionic
Oppo A54	Android	10	4GB	MediaTek Helio P35

IV. RESULTS AND DISCUSSION

A. *PWA Implementation*

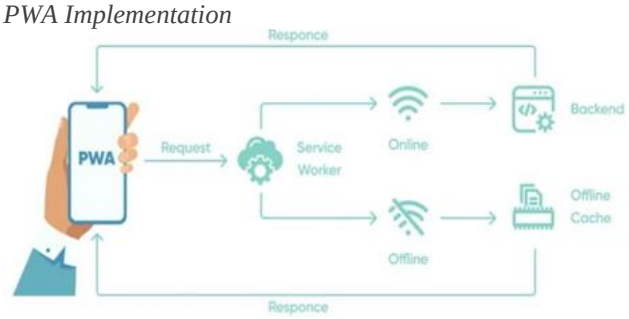

Fig 2. Workflow of Progressive Web Application (PWA) [19]

Figure 2 illustrates the workflow of Progressive Web Application (PWA) technology, demonstrating how the

p-ISSN 2301-7988, e-ISSN 2581-0588
DOI : 10.32736/sisfokom.xx.xx, Copyright ©2020
Submitted : , Revised : , Accepted : , Published :

application maintains optimal functionality under both online and offline conditions through the use of a Service Worker. A PWA is a modern web application technology designed to behave similarly to a native application, while still operating within a web browser environment.

A core component of PWA is the Service Worker, a background script that intercepts and manages network requests initiated by the user. When a user accesses the PWA, requests are first routed to the Service Worker. In an online scenario, the Service Worker forwards the request to the backend or server, retrieves the required data, and returns the response to the application. In contrast, when the device is offline, the Service Worker responds by serving data previously stored in the local cache (offline cache), ensuring the application remains functional without an internet connection. The general stages of PWA implementation are as follows:

1. *Web App Manifest Implementation*

   The manifest is a simple JSON document that defines how a web application appears and behaves when accessed by the user. Typically named manifest.json, this file is interpreted by the browser when the user opens the web application. Once detected, the browser loads the necessary resources, renders the content, and executes related processes [20].

   Figure 3 displays the contents of a typical manifest.json file. This file includes essential metadata such as the application's name (name), short name (short_name), description, and the starting URL (start_url). The display property is set to "standalone", allowing the web app to launch in a way that closely mimics a native mobile application. Additionally, it specifies the theme color, background color, and provides a set of application icons in various resolutions to support different devices. The use of a manifest file enhances the user experience by enabling a more consistent and app-like appearance across platforms, allowing users to install the web app on their home screens just like native applications.

```
{
  "name": "Aplikasi Layanan Administrasi RT/RW",
  "short_name": "LAYAR",
  "description": "Aplikasi administrasi untuk layanan RT/RW",
  "start_url": "/",
  "display": "standalone",
  "background_color": "#ffffff",
  "theme_color": "#000000",
  "orientation": "any",
  "lang": "id-ID",
  "scope": "/",
  "categories": ["utilities", "productivity"],
  "icons": [
    {
      "src": "/images/layar192.png",
      "sizes": "192x192",
      "type": "image/png",
      "purpose": "any"
    },
    {
      "src": "/images/layar512.png",
      "sizes": "512x512",
      "type": "image/png",
      "purpose": "any"
    }
  ]
}
```

Fig 3. Web App Manifest

```
<link rel="manifest" href="{{ asset('manifest.json') }}">
```

Fig 4. Calling Manifest.json

Figure 4 presents the HTML code <link rel="manifest" href="{{ asset('manifest.json') }}">, which is used to link the manifest.json file to the web page. This line of code plays a crucial role in Progressive Web Application (PWA) development, as it enables the browser to retrieve and apply the manifest configuration.

The manifest file defines how the web application should appear and behave when accessed from a user's device—particularly when the application is added to the home screen. By linking the manifest, developers ensure that metadata such as the app name, icons, theme colors, and launch behavior are recognized by the browser, enabling a more seamless and native-like user experience.

2. *Service Worker Implementation*

   A Service Worker is a type of JavaScript file that operates independently from the main scripts of a web page. Essentially, it functions as a proxy server between the web application, the browser, and—when available—the network. Service Workers enable the application to intercept network requests, manage responses based on network availability, and update server assets as needed [21].

   Figure 5 displays a JavaScript implementation of a Service Worker designed to enhance the performance and user experience of a web application through caching and notifications. The code defines a cache named layar-cache-v1 and an array of assets (PRECACHE_ASSETS) that are pre-cached during the install event to enable offline access. The activate event is used to remove outdated caches, ensuring the application uses the most current resources.

   During the fetch event, the Service Worker intercepts all GET requests. If a matching response is found in the cache, it is returned directly. Otherwise, the request is forwarded to the network, and if the response is valid, it is added to the cache for future use. This strategy ensures both improved loading speed and offline functionality, particularly beneficial in unstable network environments.

p-ISSN 2301-7988, e-ISSN 2581-0588
DOI : 10.32736/sisfokom.xx.xx, Copyright ©2020
Submitted : , Revised : , Accepted : , Published :

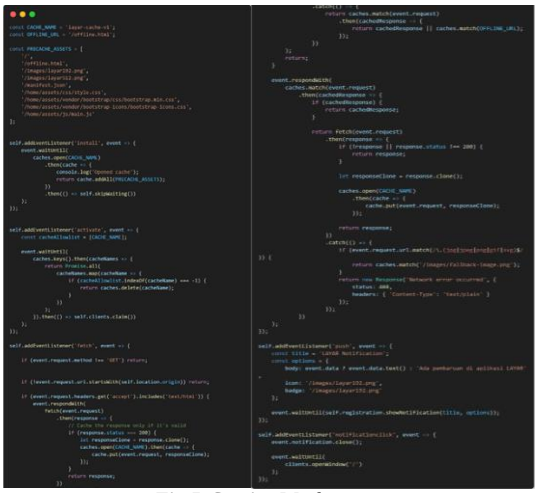

Fig 5. Service Worker

B. *Performance Testing*

This performance testing was conducted to evaluate website speed, search engine optimization (SEO) implementation, and accessibility using several PWA testing tools, specifically Google Lighthouse and Chrome DevTools. The testing was carried out on a laptop that met the necessary hardware and software requirements, with a network speed of 14.77 Mbps at the time of testing. The performance evaluation utilized two testing tools, as described below.

1. *Google Lighthouse Testing*

Based on the tests conducted, a comparison of the website's performance before and after the implementation of Progressive Web Application (PWA) technology using Google Lighthouse indicates a significant improvement. Google Lighthouse is an auditing tool used to assess the performance of websites, particularly in relation to PWA standards. It evaluates five key parameters defined by Google Developers: Performance, Accessibility, Best Practices, SEO, and PWA implementation. Each parameter is scored on a scale from 0 to 100.

Lighthouse uses a color-coded rating system to represent the quality of performance: scores between 0–49 are categorized as poor (red), scores from 50–89 indicate average performance (orange), and scores ranging from 90–100 are considered good or excellent (green). The step-by-step process of conducting a Lighthouse audit is illustrated in Figure 14. This scoring system provides a clear, standardized framework for evaluating and comparing web application performance improvements.

TABLE II. EVALUATION SCORES BY CATEGORY

Score	Status	Score Color
0-49	Slow	Red
50-89	Average	Orange
90-100	Fast	Green

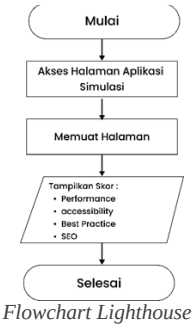

Fig 6. Flowchart Lighthouse Testing

Based on the testing conducted, the comparison of the website's performance evaluation results before and after the implementation of PWA technology using Google Lighthouse can be observed through the visualizations presented below.

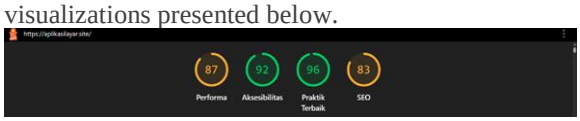

Fig 7. Prior to PWA Implementation

Before implementation, the performance score was 87, indicating a need for further optimization and the integration of PWA features.

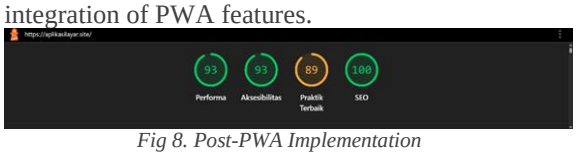

Fig 8. Post-PWA Implementation

As shown in Figure 6, after the implementation, the application demonstrated a 7% improvement in performance and achieved a perfect score of 100% in SEO. The accessibility score showed a modest improvement from 92 to 93, while the best practice score experienced a decline from 96 to 89 following PWA implementation. A detailed breakdown of the website's performance metrics assessed using Google Lighthouse is presented in Table 3.

TABLE III. WEBSITE PERFORMANCE VIA LIGHTHOUSE

Variabel	Before PWA	After PWA
Performa	87%	93%
First Contentful Paint	1,8 s	1,8 s
Total Blocking Time	50 ms	80 ms
Speed Index	4,2 s	3,9 s
Largest Contentful Paint	2,6 s	2,7 s
Cumulative Layout Shift	0,15	0,01

Based on the results of the Google Lighthouse audit, the implementation of PWA demonstrated an improvement in technical performance, with the overall score increasing from 87% to 93%—a 6-point gain. Notable enhancements were observed in the Speed Index, which decreased from 4.2 seconds to 3.9 seconds, and in the Cumulative Layout Shift (CLS), which significantly dropped from 0.15 to 0.01. Additionally, the website achieved a perfect SEO score of 100%. Although there were minor increases in Total Blocking Time (from 50 ms to 80 ms) and

p-ISSN 2301-7988, e-ISSN 2581-0588
DOI : 10.32736/sisfokom.xx.xx, Copyright ©2020
Submitted : , Revised : , Accepted : , Published :

Largest Contentful Paint (from 2.6 seconds to 2.7 seconds), the overall implementation of PWA successfully optimized the technical performance of the website, resulting in an approximate 7% improvement in performance.

2. Chrome DevTools Testing

Application performance testing was also conducted using Chrome DevTools, specifically through the Network and Performance features. This testing aimed to evaluate how the application responds under various network conditions, focusing on load time, request size, and network efficiency before and after the implementation of PWA. The tests were carried out under three different network scenarios: 3G, fast 4G, and slow 4G. The testing scenarios are illustrated in Figure 6.

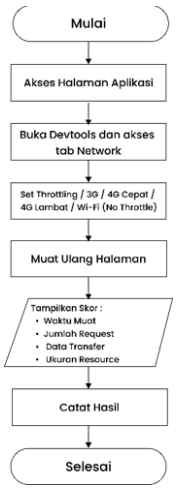

Fig 9. Flowchart Chrome DevTools

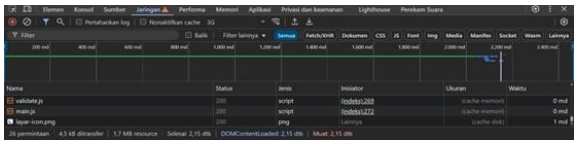

Fig 10. DevTools on 3G Network Before PWA Implementation

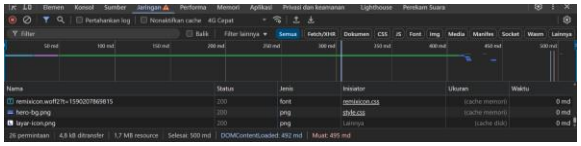

Fig 11. DevTools on 3G Network After PWA Implementation

The test results on a 3G network show that before the implementation of PWA, the application had a load time of 2.15 seconds, with 26 requests and 4.5 kB of data transferred. After implementing PWA, the load time minor increased to 2.19 seconds, while the number of requests rose to 31 and data transfer increased modestly to 7.6 kB. Although there was a slight delay in load time, the increase in requests indicates that the service worker successfully managed a larger set of cached assets. The total resource size remained consistent at approximately 1.8 MB, reflecting efficient resource handling despite the added caching processes.

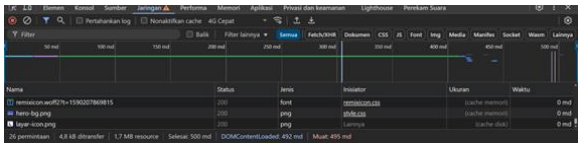

Fig 12. DevTools on Fast 4G Network Before PWA Implementation

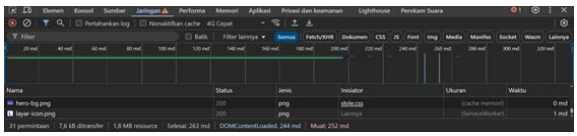

Fig 13. DevTools on Fast 4G Network After PWA Implementation

Under fast 4G network conditions, the implementation of PWA had a significant positive impact. The application's load time was reduced substantially, decreasing from 495 milliseconds to 252 milliseconds—representing nearly a 50% improvement in performance. The number of requests increased from 26 to 31, and data transfer rose from 4.8 kB to 7.6 kB, indicating that the application accessed more assets while maintaining efficiency through effective cache utilization. The overall resource size remained stable at around 1.8 MB, further demonstrating consistent and optimized asset management.

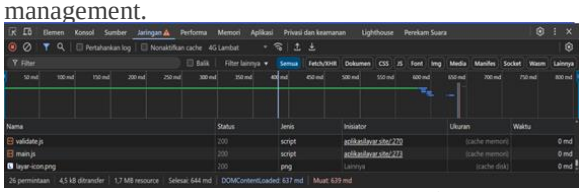

Fig 14. DevTools on Slow 4G Network Before PWA Implementation

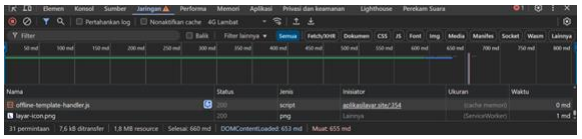

Fig 15. Performance Testing on Slow 4G Network After PWA Implementation

Meanwhile, under slow 4G network conditions, the load time minor increased from 639 milliseconds to 655 milliseconds after the activation of the PWA. However, similar to the other test scenarios, both the number of requests and the amount of data transferred showed a consistent increase, indicating that the service worker was actively managing cache and handling resource requests. The slight increase in load time remains within a reasonable range, considering the inherent variability of slower network speeds and the caching process. The following are the test results obtained using Chrome DevTools:

p-ISSN 2301-7988, e-ISSN 2581-0588
DOI : 10.32736/sisfokom.xx.xx, Copyright ©2020
Submitted : , Revised : , Accepted : , Published :

TABLE IV. PERFORMANCE RESULTS USING CHROME DEVTOOLS

Jaringan	PWA Status	Load Time	Total Request	Data Transfer	Ukuran Data
3G	Before PWA	2.15 s	26	4.5 kB	1.7 MB
3G	After PWA	2.19 s	31	7.6 kB	1.8 MB
Fast 4G	Before PWA	495 ms	26	4.8 kB	1.7 MB
Fast 4G	After PWA	252 ms	31	7.6 kB	1.8 MB
Slow 4G	Before PWA	639 ms	26	4.5 kB	1.7 MB
Slow 4G	After PWA	655 ms	31	7.6 kB	1.8 MB

As shown in Table 4, under fast 4G network conditions, the implementation of PWA resulted in a significant improvement, with load time reduced by nearly 50%. The number of requests increased after PWA implementation, indicating that more assets were successfully managed and delivered from the cache by the Service Worker. Despite the increase in requests, data transfer remained low (≤ 7.6 kB), reflecting efficient bandwidth usage due to caching. The overall resource size showed no significant change (approximately 1.7–1.8 MB), but a substantial portion was served from the cache, contributing to improved performance.

C. *Offline Testing*

This test was conducted to directly observe the browser's response when operating without an internet connection. The primary objective was to verify the successful implementation of the Offline Mode feature, which is a core characteristic of Progressive Web Applications (PWA). The testing process was carried out using the Network tab in Chrome DevTools by setting the throttling option to "offline" or by physically disconnecting the device from the internet.

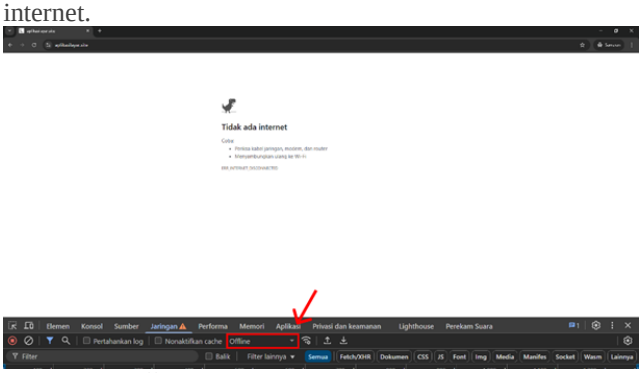

Figure 16. Offline Mode Evaluation Prior to PWA Implementation

As illustrated in Figure 16, the test results indicate that prior to the implementation of PWA, the website failed to display any content when the network connection was lost. This condition highlights a significant limitation, as users were unable to access even basic information from the application. To observe the improvements made following the implementation of PWA, a comparative analysis of the test results is presented in the following section.

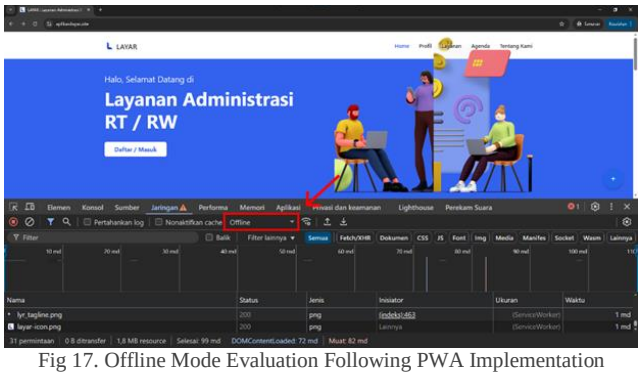

Fig 17. Offline Mode Evaluation Following PWA Implementation

The offline mode testing yielded positive results, demonstrating that the website remained accessible even in the absence of an internet connection or under unstable network conditions. The site was able to display a static page informing users of their offline status, indicating the successful implementation of this feature. The presence of offline functionality offers tangible benefits, particularly in supporting community administrative processes. Residents can now directly access and download necessary forms through the application without requiring an internet connection, eliminating the need to wait for or contact the neighborhood head (RT) to obtain the required documents.

D. *Installable Testing*

Installability testing was conducted on both desktop and mobile platforms to verify that the website could be successfully installed across a variety of user devices. On the desktop side, the testing utilized Google Chrome and Microsoft Edge browsers. For mobile testing, five different devices were used: Samsung A55 (Android), Infinix Note 40 (Android), iPhone 12 Pro (iOS), and Oppo A54 (Android). The results demonstrated that the website was successfully installed on all tested devices, indicating the successful implementation of cross-platform installability features, as illustrated in Figure 18.

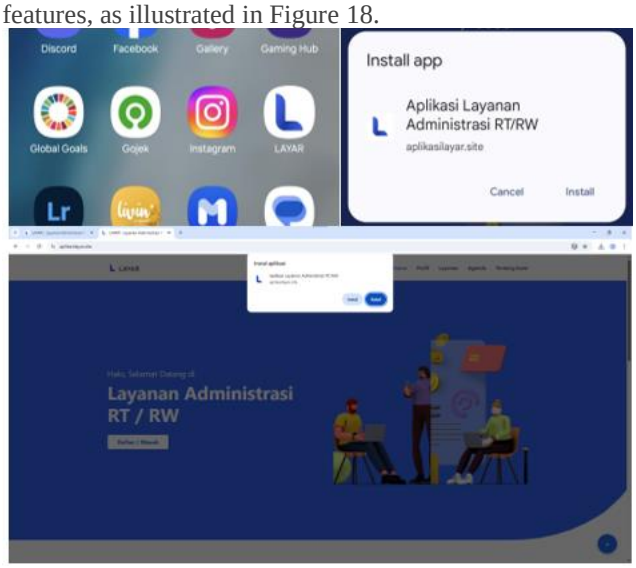

Fig 18. Installability Testing

p-ISSN 2301-7988, e-ISSN 2581-0588
DOI : 10.32736/sisfokom.xx.xx, Copyright ©2020
Submitted : , Revised : , Accepted : , Published :

V. CONCLUSION AND SUGGESTIONS

The implementation of Progressive Web Application (PWA) technology in the LAYAR application has demonstrated improvements in both performance and functionality. However, these improvements reflect technical performance as measured by automated tools such as Google Lighthouse and Chrome DevTools. They do not yet capture user experience factors such as usability, accessibility for elderly residents, or perceived usefulness of features in a real-world RT/RW context. Evaluations using Google Lighthouse revealed an increase in performance score from 87% to 93%, particularly in terms of visual stability (Cumulative Layout Shift) and initial load speed (Speed Index). Furthermore, testing via Chrome DevTools showed a load time reduction of up to 50% under fast 4G network conditions, indicating the effectiveness of caching mechanisms managed by the service worker.

Core PWA components—namely service workers, cache storage, and the web app manifest—have been successfully deployed. The application operates in offline mode and can be installed across multiple device types, including both Android and iOS platforms. This capability has the potential to enhance service delivery in areas with limited internet connection. While there was a slight increase in Total Blocking Time, the change remains within an acceptable range and does not substantially affect user experience.

Despite these achievements, this study has limitations particularly the absence of user experience testing involving actual end-users. This aspect is critical for capturing qualitative feedback on usability, satisfaction, and overall interaction with the application. Future research is strongly encouraged to incorporate comprehensive user testing to complement technical evaluations and provide a more holistic understanding of the application's impact. Building on the success of this implementation, several considerations are proposed for future development:

1. Continuous performance monitoring and optimization are essential, particularly regarding JavaScript blocking. Approaches such as lazy loading and minimizing non-essential scripts are recommended to further enhance responsiveness.
2. The adoption of PWA technology should be expanded to other public service applications such as those in healthcare, education, and government sectors to improve accessibility and operational efficiency in areas with unreliable internet infrastructure.
3. Further studies should include user testing involving real users, as well as long-term evaluations to assess the sustained effectiveness of PWA compared to alternative approaches like native or hybrid applications.
4. It is also important to ensure alignment with current best practice standards. This study observed a decrease in the Google Lighthouse Best Practices score, which dropped from 96 to 89 following the implementation of PWA. Although the overall system performance remained high, this decline indicates a need for further optimization to stay in line with updated web standards and to prevent any potential issues related to quality or security.
5. The development of additional features, such as push notifications, is recommended to enhance user engagement and facilitate the delivery of important RT/RW (neighborhood administrative) information directly through the application.

REFERENCES

[1] Oktasari, A. J., Kurniadi, D.: Perancangan Sistem Informasi Manajemen Kegiatan Mahasiswa Berbasis Web. Voteteknika (Vocational Teknik Elektronika Dan Informatika) 7(4), 149–157 (2020)

[2] Coding Studio, https://codingstudio.id/blog/progressive-web-app-adalah/, diakses 11/09/2024

[3] MyEduSolve Indonesia, https://myedusolve.com/id/blog/progressive-web-app-pwa-definisi-keunggulan-dan-cara-membuatnya, diakses 11/09/2024.

[4] Google Developers. The Business Impact of Progressive Web Apps (2023). https://developers.google.com/web/progressive-web-apps

[5] DataReportal. Digital 2023: Global Overview Report. (2023). https://datareportal.com/reports/digital-2023-global-overview-report

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron Microsoft Azure. User Experience and Page Load Speed. (2023). https://azure.microsoft.com

[7] GS StatCounter, "Platform Market Share Indonesia," https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/indonesia

[8] DataReportal, "Digital 2023: Indonesia," https://datareportal.com/reports/digital-2023-indonesia

[9] Muawwal, A.: The Implementation of PWA (Progressive Web App) Technology in Enhancing Website Performance & Mobile Accessibility. Buletin Pos dan Telekomunikasi, 22(1), 25-36. (2024).

[10] Kurniawan, A. A.: Analisis Performa Progressive Web Application (PWA) Pada Perangkat Mobile. Jurnal Ilmiah Informatika Komputer 25(1), 18-31 (2020)

[11] Adelin, N. M.: Implementasi progressive web apps (pwa) untuk meningkatkan kinerja dan performa situs maritimepreneur. Jurnal Informatika Dan Teknik Elektro Terapan 12(1) (2024)

[12] Haryanto, D., Elsi, Z. R. S.: Analisis Performance Progressive Web Apps Pada Aplikasi Shopee. Jurnal Ilmiah Informatika Global 12(2) (2021)

[13] Cook, T. D., & Campbell, D. T.: Quasi-experimentation: Design and analysis issues for field settings. Houghton Mifflin. (1979).

[14] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A.: Experimentation in Software Engineering. Springer. (2012). https://doi.org/10.1007/978-3-642-29044-2

[15] Chrome for Developers.: Pengantar Lighthouse. (2016, September 27). https://developer.chrome.com/docs/lighthouse/overview?hl=id

[16] Chrome for developers: Chrome Devtools (2016, March 28) https://developer.chrome.com/docs/devtools/overview?hl=en

[17] Faizin, M. A., Nevin, M., & Yuhana, U. L.: Indonesia egovernment website performance and accessibility evaluation using automated tool lighthouse. In 2024 2nd International Conference on Software Engineering and Information Technology (ICoSEIT) (pp. 210-215). IEEE. (2024, February)

[18] Ramadurai Venkataraajalu, Sivaramarajalu. (2024). The Role of Progressive Web Apps and WebGL in Modern Front-End Engineering. IJARCCE. 10. 10.17148/IJARCCE.2021.10830.

[19] C. Love, Progressive Web Application Development by Example: Develop fast, reliable, and engaging user experiences for the web (2018)

[20] D. Hume, Progressive Web Apps. Simon and Schuster (2017)

p-ISSN 2301-7988, e-ISSN 2581-0588
DOI : 10.32736/sisfokom.xx.xx, Copyright ©2020
Submitted : , Revised : , Accepted : , Published :

p-ISSN 2301-7988, e-ISSN 2581-0588
DOI : 10.32736/sisfokom.xx.xx, Copyright ©2020
Submitted : , Revised : , Accepted : , Published :