International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 08 Issue: 06 | June - 2024 SJIF Rating: 8.448 ISSN: 2582-3930

Progressive Web Apps in Cross-Platform Development: A Comparative Analysis and Evaluation

Manu S Rao[1], Rekha B S[2]

[1] Department of ISE, RV College of Engineering® Bengaluru, India

[2] Department of ISE, RV College of Engineering® Bengaluru, India

---

Abstract - The evolution of app development methodologies has been significant, yet the discussion around cross-platform development remains pertinent. Apps must seamlessly function on both Android and iOS devices, accommodating diverse hardware configurations and platform versions. The proliferation of device categories beyond smartphones and tablets adds further complexity to achieving multi-platform compatibility. Despite the presence of multi-platform frameworks supported by both practical application and research, the challenge of efficiently developing apps for various targets persists. Progressive Web Apps (PWAs) potentially offer a solution to this ongoing quest for unifying technology. This paper aims to evaluate the role of PWAs in multi-platform development, exploring their foundational aspects and assessing the current landscape of possibilities. The evaluation includes a comparative analysis conducted in two phases. In the first phase, the different application development paradigms are analyzed with their advantages and disadvantages. In the second phase, the PWAs are compared to native and cross-hybrid apps based on launch time, installation size, and Largest Contentful Paint (LCP), with PWAs outperforming in all three metrics. By examining these aspects, the paper seeks to provide a complete understanding of PWAs' potential in cross-platform development.

Key Words*: *Progressive Web Apps, Cross-Platform Development, App Development, Web Application, Multi-Platform Compatibility, Frameworks, Comparative Analysis, Performance Evaluation, Technology Unification, Mobile Devices, Android, iOS

1.INTRODUCTION

Since the debut of the inaugural iPhone [1] over a decade ago, the landscape of mobile development methodologies has evolved significantly. During this period, the fundamentals of mobile app development have undergone simultaneous simplification and complication. On one hand, the proliferation of platforms with significant market presence has diminished, accompanied by the emergence of robust cross-platform development frameworks [2] and advancements in various other aspects. Conversely, challenges such as device fragmentation persist, necessitating support for emerging device categories like wearables amidst the relentless pace of technological innovation.

The emergence of multi-platform hybrid app development methods has significantly facilitated the creation of apps for multiple operating systems and platforms, offering benefits such as reduced learning curves, cost-effectiveness, and expedited time-to-market. However, choosing between native, multi-platform, and web apps can still be complex, especially for graphically-intensive games that often require SDK [3].

Progressive online Apps (PWAs) are a modern way to application development that claim to combine the internet technology's user-friendliness with the benefits of native apps. This process can be carried out with app sizes significantly reduced without sacrificing functionality. Despite these benefits, very little research has been done on PWAs; most evaluations have been done on the Android platform. This research aims to bridge the gap between academic inquiry and industry implementation by investigating whether PWAs can evolve into a multiplatform solution that addresses the problems with existing cross-platform development paradigms or serve as the unifying technology for cross-platform app development.

2. BACKGROUND

In the following section, we outline the background of our study. Initially, we outline native application development, cross platform development and PWAs as fundamental approaches and examine their developmental principles, unique features and their benefits and drawbacks.

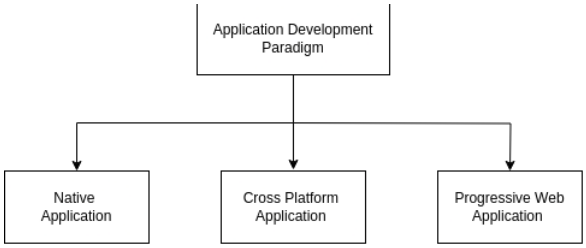

Fig -1: Classification of Application Development

2.1 Native Application Development

Developing native applications for the Android and iOS platforms requires using the official programming languages and development environments from Google and Apple, respectively. The Integrated Development Environment (IDE) for Android is called Android Studio [4], and the main IDE for iOS is called Xcode [5]. Designed specifically for each platform, these environments provide an extensive toolkit for developing, writing, testing, and debugging applications.

The two main programming languages used in Android development are Java [6] and Kotlin [7]. Although Java has long been the preferred option, JetBrains' [8] Kotlin has gained popularity due to its contemporary features and compatibility with Java. Conversely, since its introduction by Apple in 2014, Swift [9] has been the language of choice for iOS developers. It provides developers with a simplified and expressive language for creating iOS apps, and it is made to be current, safe, and efficient.

One of native app development's main benefits is that it can make use of platform-specific features and APIs to create responsive, high-performance apps that interface with the underlying operating system with ease. Furthermore, because native apps follow the platform's UI conventions and design rules, they usually provide a superior user experience.

However, there are certain drawbacks to local development as well. One significant disadvantage is that developing applications for iOS and Android requires different codebases, which can add to the expense and duration of development. Furthermore, it can be difficult to update and maintain two different codebases, particularly for smaller development teams. In addition, apps must adhere to platform-specific rules and review procedures in order to be submitted to the appropriate app stores (the Apple App Store [10] for iOS and the Google Play Store [11] for Android), which can be a time-consuming and restricted process. Native app development is still a well-liked option for creating excellent, platform-specific mobile apps that offer the best user experience and performance in spite of these difficulties.

2.2 Cross Platform Development

Cross-platform development addresses the challenges of developing native apps for many platforms, such as Android and iOS. The use of frameworks and technologies that allow developers to create code once and deliver it across various platforms lowers the need for several codebases and streamlines the development process. There are various cross-platform frameworks available, each with unique capabilities, benefits, and downsides.

React Native [12], developed by Facebook, is one of the most used cross-platform frameworks. It allows developers to create mobile applications with JavaScript and React, a popular toolkit for developing user interfaces. With React Native, a single codebase can be used for both Android and iOS, greatly increasing code reusability.

The fundamental advantage of cross-platform development is the ability to write code once and deploy it across numerous platforms, saving time and money. These frameworks often include tools and libraries that make development easier. For example, hot reloading enables developers to examine the effects of code changes without having to rebuild the entire app.

Despite these advantages, cross-platform development has several downsides. Some frameworks may not support all platform-specific features and APIs, limiting the app's functionality. Furthermore, performance concerns can develop, particularly in sophisticated systems that require great performance and responsiveness.

2.3 Progressive Web Application

Progressive Web Apps (PWAs) are a cutting-edge approach of creating websites that combine web technology capabilities with native mobile app experiences. PWAs provide customers with a more app-like experience than traditional web apps by integrating features such as push notifications, offline capabilities, and home screen installation. PWAs are built using common web technologies such as HTML, CSS, and JavaScript. Their primary characteristics include responsiveness, dependability, and speed across several devices and network configurations.

PWAs can operate offline or with very low connectivity due to the use of service workers [15], which are background-running scripts that have the ability to intercept network requests. Because of this, PWAs can cache resources and content, allowing users to access them even when they are not online. Because PWAs may be loaded on a user's device and utilized from the home screen in a manner similar to native programs, they also provide a seamless user experience.

PWAs are superior to native apps and conventional web apps in a number of ways. PWAs provide a uniform user experience across many devices and platforms, and they are quick and responsive for users, even on sluggish or unstable networks. Because PWAs don't need separate code bases or app store submissions, they are also less expensive to develop and distribute than native apps.

PWAs do, however, have certain restrictions. They might not be able to access all of the features and functionalities of native apps, like app integration or device hardware access. Furthermore, PWAs might not work with all browsers or devices, which would restrict their accessibility to particular user demographics.

3. RELATED WORK

Shubham et al., [16] presents a Data Retrieval System for small-sized and medium-sized businesses, offering a solution for tracking sales, purchases, profits, losses, and expenses. Utilizing Progressive Web Application (PWA) technology, the system functions without requiring any downloads, hence saving storage space. The app is developed using React for the user interface and a MongoDB server, operating as a SaaS model. The paper details the development, implementation, and benefits of PWAs for business data collection.

Zulkifli et al. [17] investigate the design and performance of Progressive Web Apps (PWAs) for e-commerce that employ the Angular framework and Service Worker technologies. The study underlines the growing relevance of e-commerce for business success and looks into how PWAs might improve e-commerce experiences. The researchers ran performance studies to compare systems with and without Service Worker, concentrating on parameters like response time, throughput, and latency. The findings show that PWAs are quick, dependable, and ideal for e-commerce apps.

Alonge et al. [18] present an architecture for Progressive Web Apps (PWAs) that can run in places with little or no connectivity. The authors want to address the issue of data transfer from client to server while a web application is offline, which is a restriction of present PWA capabilities. The proposed approach entails sending critical data via an SMS platform when offline, with a product created utilizing a service worker to identify offline applications and transfer data via SMS. The results show that the suggested architecture can be utilized to send data in offline environments, with suggestions for future research to improve the solution. The study also covers the significance of PWAs in today's environment, their benefits, and the principles that they must follow.

Oforji et al., [19] addresses the creation of a cross-platform mobile app for the healthcare industry utilizing Progressive Web Apps (PWA), Deep Learning, and Natural Language Processing (NLP). The app intends to process patient medical records and incorporate an intelligent system for diagnosis and therapy prescriptions. PWAs are 42 times smaller than android apps and launch faster. The article also investigates the use of Hybrid programs, which are cost-effective cross-platform programs created using web technologies. However, when compared to native apps, they may run worse and have less functionality.

Fernando et al., [20] delves into the study of caching mechanisms in Progressive Web Applications (PWAs), contrasting them with traditional web applications. The paper develops two applications, one traditional and one PWA, testing both with the Google Lighthouse tool on desktop and mobile devices. The results support PWAs' superior performance in certain contexts. The paper's primary focus is to highlight the diverse caching methods in PWAs and traditional web applications, emphasizing the performance benefits of PWAs.

Abasiama et al., [21] aims to develop an electronic model, a digital wallet, to replace traditional physical wallets with an online wallet system for small-scale organizations.The project uses structured System Analysis and Design Methodology to ensure a systematic approach from inception to completion.The goal is to provide a safe, flexible, and scalable electronic payment solution for small-scale organizations, minimizing the common problems associated with physical cash transactions, such as theft and loss, while improving the ease of electronic transactions.

Pratik et al., [22] investigates the fundamentals of Progressive Web Applications (PWAs) in cross-platform development, providing a thorough assessment of their existing capabilities in comparison to traditional cross-platform app development methodologies. The paper evaluates the PWAs to alternative cross-platform development methodologies from both technical and overall viewpoints and finishes with a comprehensive review of the results.

Jasmine et al., [23] studies how Progressive Web Apps (PWAs), a solution provided by Google, effectively address the inherent limitations of both native mobile apps and web browsers.The paper delves into how PWAs combine the best features of native apps and web experiences, providing a more efficient, faster-loading, and user-friendly solution without the need for extensive storage or consistent network connectivity.

Stefan et al., [24] compares the energy usage of different mobile development techniques with Progressive Web Apps (PWAs), with an emphasis on UI rendering and interaction scenarios. The study discovered that although PWAs use more energy than other mobile cross-platform development techniques, they are still a competitive alternative to native development, which uses the least energy. The PWA's energy footprint is greatly affected by the kind of web browser that is used to run it. Because consumers are cognizant of their smartphones' energy efficiency, the study highlights the significance of energy-efficient apps. The report also emphasizes how difficult it is for developers to design mobile apps that are energy-efficient due to the lack of tools for diagnosing and analyzing energy-related problems.

Malavolta et al., [25] evaluates how service providers affect two mobile devices' energy efficiency as well as various network scenarios for Progressive Web Apps (PWAs). The two main elements of the empirical experiment utilised in the study were the use of service workers and the type of network (2G or WiFi). The study found that there was no interaction between the two factors and that service staff had no appreciable effect on the energy usage of the two devices, regardless of network conditions. The study's conclusions point to the possibility of enhancing energy efficiency through PWA and service worker technologies, which is a positive move in narrowing the user experience gap between native apps and mobile web apps.

Rensema et al.,[26] examines the core components of PWAs with an emphasis on security, privacy, compatibility, performance, and the effect on users and businesses. Compatibility tests across eight browsers on four different operating systems are part of the research, along with analyses of Service Worker, Web App Manifest, add-to-home-screen features, and offline capabilities. The findings indicate that while iOS has limited support for PWA capabilities, the majority of them are supported by major browsers, particularly on Android and Chromium-based platforms. The implementation technique has a considerable impact on performance, and proper optimization can yield large benefits. However, PWAs are a tempting option for contemporary web development because they have been shown to increase user re-engagement and revenue for large enterprises.

Steiner et al., [27] examines whether Web Views, which are in-app web experiences featured in applications rather than standalone web browsers, support Progressive Web App (PWA) features. PWAs can work offline, get push alerts, and synchronize data in the background thanks to Service Workers APIs. Although a few standalone Android browsers support Service Workers, the support for them in Web Views varies greatly. The PWA Feature Detector is an open-source tool created by the authors to assess PWA feature support on various devices and Web Views. Although the study indicates that there are significant differences between different Web View technologies and the browser engines that power them, it also finds that on Android , the results are consistent irrespective of the version of the operating system, which is helpful considering the inconsistent update policies of many manufacturers.

Jiyeon et al. [28] examines the special security and privacy issues associated with Progressive Web Apps (PWAs), which provide offline surfing and native app-like functionality using HTML5 capabilities like caching, push notifications, and service workers. The analysis finds design faults in widely used third-party push services and vulnerabilities in key browsers that raise the danger of phishing. In addition, a demonstration of a bitcoin mining attack that takes advantage of service workers is presented, coupled with a new side-channel attack that exploits offline caching to infer users' history of visited PWAs. The research makes a number of recommendations and countermeasures to lessen these dangers, with a focus on stronger push notification systems, enhanced browser security, and tactics to stop cache-based exploits.

David et al., [29] explores the potential of web apps to run on all devices, as opposed to the traditional native mobile app development for each platform, which can be expensive. With advancements in web technologies, web apps can now offer more features and capabilities, making them a viable option for mobile app development. The paper introduces the concept of Progressive Web Apps (PWA), created by Google, which aims to standardize web development. It highlights the advantages of developing apps centrally as a PWA, comparing it to developing for each mobile platform. The paper also discusses

the current state of web technologies and the scenarios where PWAs are a strong alternative to native mobile apps.

Andreas et al.,[30] advocates for PWAs as a possible technology that could bridge the gap between native and online apps. The study compares the performance of two cross-platform mobile apps with a Progressive Web App after providing an outline of PWA characteristics. These programs were created to confirm findings and made available for validation in an open-source repository.

4. METHODOLOGY

The research paper employs a distinct and comprehensive methodology to evaluate the performance of Progressive Web Applications (PWAs), aiming to provide a thorough comparison with native and cross-platform applications.

The methodology involves a detailed comparison of the performance metrics of three types of applications: Native Applications, Cross-Platform Applications, and Progressive Web Applications (PWAs). Specifically, the performance metrics under scrutiny are Largest Contentful Paint (LCP), Installation Size, and Launch Time. To ensure a robust and meaningful comparison, two types of applications are developed for each method. The first type of application is designed to display static and predefined content, which serves as a controlled environment to measure basic performance attributes without the variability of network conditions. The second type of application is more complex and dynamically loads data from the internet, providing insights into how each application handles real-world data retrieval and rendering tasks. This dual-application approach allows for a comprehensive assessment of how static versus dynamic content impacts performance across different application types.

4.1 Device Details

For this evaluation, all applications—Native Applications, Cross-Platform Applications, and Progressive Web Applications (PWAs)—are tested using the same device to ensure consistency and reliability in the performance metrics. The device is carefully configured to minimize any background activity that might interfere with the evaluation methodology. This controlled environment is crucial to maintain the integrity of the collected data and to provide an accurate comparison of the performance metrics across different application types and web browsers.

Table -1: Device Specification

Device Name	Nothing Phone 2A
RAM	8GB
Processor	Dimensity 7200
Operating System	Android 14

4.2 Application Details

To rigorously evaluate the performance metrics of different application development methods, a total of six applications were developed. These applications were strategically designed to encompass both static and dynamic content scenarios across three distinct development approaches: Native Applications, Cross-Platform Applications, and Progressive Web Applications (PWAs). For every development method, two apps were specifically made to enable a thorough and equitable comparison.

4.2.1 Development of Native Application

For the first methodology, two native applications were developed for the Android platform to evaluate performance in static and dynamic content scenarios. The first application displayed a static heading and content of approximately 1200 characters, providing a controlled environment for assessing metrics such as LCP, Installation Size, and Launch Time. The second application dynamically loaded the same content from the internet, simulating real-time data fetching and rendering. Both applications were developed using Kotlin and Jetpack.

4.2.2 Development of Cross Platform Application

Two cross-platform applications were developed to compare with the native applications. These apps, designed to run on both Android and iOS, were built using the React Native framework. The first application displayed static content, including a heading and 1200 characters of text, to benchmark performance in a controlled environment. The second application dynamically loaded the same content from the internet, testing performance in dynamic scenarios. React Native and JavaScript were chosen for their ability to deliver a near-native user experience.

4.2.3 Development of Progressive Web Application

For the PWAs, two web applications were created to evaluate performance under static and dynamic content scenarios. The first PWA displayed a static heading and 1200 characters of content, providing a controlled environment to measure performance metrics like load time and rendering speed. The second PWA dynamically loaded the same content from the internet, reflecting real-world usage patterns. These PWAs were developed using React.js and JavaScript, chosen for their efficiency in handling dynamic data and creating responsive user interfaces.

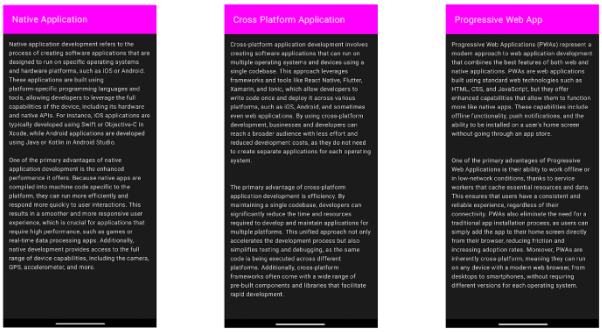

Fig -2: Developed Applications (Native, Cross Platform, PWA)

5. RESULTS

The methodology employed in this comparative analysis involved systematically measuring and evaluating key performance metrics across three different types of mobile applications: Native Applications, Cross-Platform Applications, and Progressive Web Applications (PWAs). A detailed and structured approach was adopted to ensure the accuracy and reliability of the performance measurements, utilizing appropriate tools and techniques tailored to each

metric under consideration. The primary metrics of interest were Installation Size, Launch Time, and Largest Contentful Paint (LCP).

5.1 Installation Size

The installation size represents the amount of storage space required on a device to install the application. For PWAs, this metric indicates the total storage space needed to download the PWA and its assets completely. In the case of applications displaying static content, the PWA exhibited a significantly smaller installation size of 1.43 MB compared to the native app and the cross-platform app, which required 7.39 MB and 12.7 MB of storage space, respectively. This notable difference highlights a key advantage of PWAs: their ability to deliver lightweight applications that consume minimal device storage, facilitating faster downloads and installations and thereby enhancing the user experience.

For applications loading dynamic data, the PWA had an installation size of 2.4 MB, whereas the native app and cross-platform application required 8.9 MB and 13.5 MB of storage space, respectively. This further underscores the primary advantage of PWAs in providing robust applications with minimal device storage requirements

5.2 Launch Time

Launch time refers to the duration taken by an application to open and become fully functional after being launched by the user. For applications with static data, the native application demonstrated the fastest launch time of 760 milliseconds, followed by the cross-platform app with a launch time of 960 milliseconds, and the PWA with a launch time of 1240 milliseconds. The longer launch time of the PWA reflects the necessity of loading and rendering the web content completely. However, it is important to note that subsequent launches are quicker due to browser caching of PWAs for offline availability.

For applications with dynamic data, the native application again showed the fastest launch time of 890 milliseconds, followed by the cross-platform app at 1080 milliseconds, and the PWA at 1400 milliseconds. This performance pattern highlights the need for PWAs to complete a full initial load, resulting in a longer initial launch time.

5.3 Largest Contentful Paint

The Largest Contentful Paint (LCP) metric evaluates the time it takes for the largest content piece in the viewport to become visible to the user during page load. In applications with static content, the native application outperformed both the cross-platform and PWA applications, achieving an LCP of 600 milliseconds. The cross-platform app had an LCP of 780 milliseconds, while the PWA registered an LCP of 1000 milliseconds. The relatively larger LCP of the PWA indicates the requirement for complete downloading and rendering of the application, leading to longer LCP times compared to native and cross-platform applications.

For applications with dynamic data, the native application maintained the lowest LCP of 690 milliseconds, followed by the cross-platform application at 890 milliseconds, and the PWA at 1120 milliseconds. This consistency in performance metrics further illustrates the efficiency of native applications in rendering content swiftly, while PWAs, due to their comprehensive loading processes, exhibit longer LCP times.

Table -2: Performance metrics for Static Data Application

Measure	Native App	Cross Platform App	PWA
Installation Size	7.39 MB	12.7 MB	1.43Mb
Launch Time	760 ms	960 ms	1240 ms
LCP	600 ms	780 ms	1000 ms

Table –3 : Performance metrics for Dynamic Data App

Measure	Native App	Cross Platform App	PWA
Installation Size	8.9 MB	13.5 MB	2.4 MB
Launch Time	890 ms	1080 ms	1400 ms
LCP	690 ms	890 ms	1120 ms

6. CONCLUSION

6.1 Conclusion

The comparative analysis of Native Applications, Cross-Platform Applications, and Progressive Web Applications (PWAs) reveals distinct performance characteristics across key metrics. Native applications consistently demonstrated superior performance in terms of launch time and Largest Contentful Paint (LCP), showcasing their efficiency in rendering and quick access. Cross-platform applications, while slightly lagging behind native apps, offered a balanced performance across metrics due to their versatile framework. PWAs, despite having the smallest installation size advantage, exhibited longer launch times and LCP due to the need for complete loading and rendering of web content. These findings highlight that while PWAs are advantageous for their lightweight storage requirements and broad accessibility, native applications remain the optimal choice for performance-critical scenarios. Cross-platform applications provide a viable middle ground, balancing performance and development efficiency.

6.2 Future Work

Future studies should look into ways to improve the performance of web technologies and sophisticated caching techniques to minimize the startup time and Largest Contentful Paint (LCP) of Progressive Web Applications. Furthermore, broadening the scope of the research to encompass a greater range of devices and network configurations may yield a more thorough comprehension of performance dynamics. Examining the effects of employing distinct development frameworks and libraries for cross-platform apps may also provide valuable information for enhancing their efficiency.

REFERENCES

1. Proske, Marina & Poppe, Erik & Jaeger-Erben, Melanie. (2020). The smartphone evolution - an analysis of the design evolution and environmental impact of smartphones.

2. Youn, Dongliang & Hu, Minjie. (2021). A Comparative Study of Cross-platform Mobile Application Development.

3. Google. (n.d.). Command-Line tools : Android studio : Android developers. Android Developers. https://developer.android.com/tools

4. Android Developers. (n.d.). Android Studio. Retrieved May 27, 2024, from https://developer.android.com/studio

5. Apple Inc. (n.d.). Xcode. Retrieved May 27, 2024, from https://developer.apple.com/xcode/

6. Oracle. (n.d.). Java. Retrieved May 27, 2024, from https://www.java.com/en/

7. JetBrains. (n.d.). Kotlin programming language. Retrieved May 27, 2024, from https://kotlinlang.org/

8. JetBrains. (n.d.). JetBrains: Essential tools for software developers and teams. Retrieved May 27, 2024, from https://www.jetbrains.com/

9. Swift. (n.d.). Swift programming language. Retrieved May 27, 2024, from https://www.swift.org/

10. Apple Inc. (n.d.). App Store - Apple (IN). Retrieved May 27, 2024, from https://www.apple.com/in/app-store/

11. Google. (n.d.). Google Play. Retrieved May 27, 2024, from https://play.google.com/

12. Facebook. (n.d.). React Native. Retrieved May 27, 2024, from https://reactnative.dev/

13. JavaScript.com. (n.d.). JavaScript. Retrieved May 27, 2024, from https://www.javascript.com/

14. Facebook. (n.d.). React documentation. Retrieved May 27, 2024, from https://reactjs.org/.

15. Mozilla. (n.d.). Service Worker API - Web APIs | MDN. Retrieved May 27, 2024, from https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API.

16. S. M. Gaikwad and K. R. Kulkarni, "Data collection system based on pwa (progres- sive web app) as saas," 2022 5th International Conference on Advances in Science and Technology (ICAST), pp. 270–273, 2022. [Online]. Available: https://api. semanticscholar.org/CorpusID:256878711.

17. Z. Tahir, A. A. Ilham, M. Niswar, Adnan, and A. A. Fauzy, "Progressive web apps development and analysis with angular framework and service worker for e- commerce system," 2021 IEEE International Conference on Computing (ICOCO), pp. 192–195, 2021. [Online]. Available: https : / / api . semanticscholar . org / CorpusID:246039015.

18. A. O. Josephe, C. Chrysoulas, T. Peng, B. E. Boudani, I. Iatropoulos, and N. Pitropakis, "Progressive web apps to support (critical) systems in low or no connec- tivity areas," 2023 IEEE IAS Global Conference on Emerging Technologies (Glob-ConET), pp. 1–6, 2023. [Online]. Available: https://api.semanticscholar.org/ CorpusID:259179353.

19. O. Jerome and A. Onway, "Effective cross-platform mobile app development using progressive web apps, deep learning and natural language processing," International Journal of Engineering Applied Sciences and Technology, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:260328925.

20. F. Correia, ´O. Ribeiro, and J. C. Silva, "Progressive web apps development: Study of caching mechanisms," 2021 21st International Conference on Computational Science and Its Applications (ICCSA), pp. 181–187, 2021. [Online]. Available: https: //api.semanticscholar.org/CorpusID:247476934.

21. A. Akpan, S. Mmeah, and B. Baah, "E-wallet system implementation: Impact on small scale business," International Journal of Computer Applications, vol. 8, pp. 2277–128, 2018.

22. P. Thacker and D. Dharani, "Realization of native apps using progressive web apps," Journal of emerging technologies and innovative research, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:218774075.

23. J. Muman, "Progressive web apps: An optimistic approach to traditional application development," 2021. [Online]. Available: https://api.semanticscholar.org/ CorpusID:237396884.

24. S. Huber, L. Demetz, and M. Felderer, "A comparative study on the energy con- sumption of progressive web apps," Inf. Syst., vol. 108, p. 102 017, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:247334626.

25. I. Malavolta, K. Chinnappan, L. Jasmontas, S. Gupta, and K. A. K. Soltany, "Evaluating the impact of caching on the energy consumption and performance of progressive web apps," 2020 IEEE/ACM 7th International Conference on Mo- bile Software Engineering and Systems (MOBILESoft), pp. 109–119, 2020. [Online] .

26. D.-J. Rensema, "The current state of progressive web apps: A study on the per- formance, compatibility, consistency, security and privacy, and user and business impact of progressive web apps," Dissertation, Karlstad University, 2020. [Online]. Available: https://urn.kb.se/resolve?urn=urn:nbn:se:kau:diva-78904.

27 T. Steiner, "What is in a web view: An analysis of progressive web app features when the means of web access is not a web browser," Companion Proceedings of the The Web Conference 2018, 2018. [Online]. Available: https://api.semanticscholar. org/CorpusID:12592264.

28. 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 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18), New York, NY, USA: Association for Computing Machinery, 2018, 1731–1746. doi: 10.1145/3243734.3243867.

29. D. A. Hume, "Progressive web apps," 2017. [Online]. Available: https : / / api . semanticscholar.org/CorpusID:67337068.

30. A. Biørn-Hansen, T. Majchrzak, and T. Grønli, "Progressive web apps: The possible web-native unifier for mobile development," in Proceedings of the 13th International Conference on Web Information Systems and Technologies - Volume 1: WEBIST, 2017, pp. 344–351, isbn: 978-989-758-246-2. doi: 10.5220/0006353703440351.