ISSN: 2473-4901
v9 n5944

Cross-Platform App Development: A Comparative Study of PWAs and React Native Mobile Apps

Veerendra Jagatha
jagathaveerendrasai1@cityu.edu

Ali Khamesipour
khamesipoural@cityu.edu

Sam Chung
chungsam@cityu.edu

School of Technology & Computing
City University of Seattle
Seattle, WA 98121, USA

Abstract

This paper compares the architecture, performance, and user experience of Progressive Web Apps (PWAs) and React Native mobile apps by converting a web application to both platforms using React framework. The study evaluates the challenges faced and modifications made during the conversion process and aims to provide developers with insights into the best approach for a specific use case. The paper offers valuable guidance to developers to make informed decisions when choosing between PWAs and React Native mobile apps. The benefits of the research include identifying the best platform for a specific use case and providing guidance to developers to make informed decisions. The paper fills a gap in the literature regarding direct comparisons between PWA's and React Native platforms in architecture, performance, and user experience. PWAs offer cross-platform compatibility and can deliver comparable performance, making them a strong choice for developers aiming for wide device accessibility. On the other hand, React Native apps provide a more native-like experience and have better access to native device capabilities. The choice between the two platforms should be guided by specific project requirements, such as architectural preferences, performance needs, and user experience goals. Additionally, considerations like development resources, time constraints, and target platform compatibility play a crucial role in the decision-making process.

Keywords: Mobile Apps, Cross-Platform Development, Progressive Web Apps, React Native, React

1. INTRODUCTION

The increasing popularity of Progressive Web Apps (PWAs) and React Native mobile applications led to a growing interest in understanding the strengths and weaknesses of each platform. Previous studies on cross-platform mobile development tried to compare usability (de Andrade Cardieri & Zaina, 2018) and quality assurance (Zohud & Zein, 2021). However, there needs to be a more direct comparison of the two platforms.

This paper aims to compare the architecture, performance, and user experience of PWA's and React Native mobile apps by converting a web application to a PWA and a React Native mobile app using React framework. By analyzing the challenges faced and modifications made during the conversion process, this case study provides

insights into the best approach for a specific use case by helping developers to make informed decisions when choosing between PWAs and React Native mobile apps.

The rise of mobile devices has caused a growing demand for mobile applications that provide a seamless user experience across multiple devices. While native apps offer high performance and accessibility, they require separate development for different platforms and can be expensive to maintain. A mobile app should ideally be compatible with all major mobile platforms (Zohud & Zein, 2021). PWAs and React Native mobile applications have emerged as popular alternatives which offer the advantages of cross-platform development and a native-like user experience.

The user interface (UI) is a crucial aspect of mobile app development that directly impacts the user experience. Effective interface design involves creating well-designed input and output mechanisms that satisfy the user's needs, capabilities, and limitations most efficiently (de Andrade Cardieri & Zaina, 2018). Interface elements serve as a communication channel between the user and the system, and their design can significantly impact the usability and performance of mobile apps. So, delivering a high-performing app is crucial for ensuring a positive user experience and maximizing user engagement. Furthermore, choosing the appropriate architectural approach is crucial in building scalable, maintainable, and extensible apps. By comparing the performance and architectural differences between PWAs and React Native mobile apps, developers can gain valuable insights, optimize performance, and design robust apps that meet their specific project requirements. Therefore, a comprehensive analysis of the architecture, performance, and user experience of PWAs and React Native mobile apps, as proposed in this study, can provide valuable insights into the effectiveness of these platforms' interface design, and inform future development practices.

We convert a web app to PWA and React Native mobile apps and evaluate them based on architecture, performance, and user experience. Based on the findings, we guide developers in making informed decisions when choosing between PWAs and React Native mobile apps, as there is a lack of research that directly compares the two platforms. The study involves

- developing both PWA and React Native versions of the web app,
- evaluating them based on the architecture, performance, and user experience, and
- comprehensively presenting the findings.

This study focuses on cross-platform mobile development, an area of substantial importance in today's digital landscape. By examining the architecture, performance, and user experience of both PWAs and React Native mobile apps, this research contributes valuable insights for educators, practitioners, and researchers seeking a comprehensive understanding of these platforms. The primary audience for this work includes educators and researchers in the field of digital technology, mobile application development, and cross-platform solutions. Additionally, developers and practitioners seeking informed decision-making guidance when choosing between PWAs and React Native mobile apps will find this study instrumental in their endeavors.

2. BACKGROUND

Over time, various approaches have been used to develop cross-platform applications as the development of mobile applications evolved. The creation of PWA utilizing web technologies like HTML, CSS, and JavaScript is a common method for developing cross-platform mobile applications. PWA can deliver app-like experiences on the web, with offline access and native-like performance. Love (2018) provides an example-based approach to building PWAs, while de Andrade Cardieri and Zaina (2018) analyze user experience in mobile web, native, and PWA.

Another popular cross-platform mobile application development approach uses frameworks such as React and React Native. React is a JavaScript library for building user interfaces, while React Native allows developers to create native apps for iOS and Android platforms with a single codebase. Subramanian (2019) provides a guide to building full-stack web applications using the MERN (Mongo DB, Express, React, and Node) stack, which includes React for the frontend development, while Boduch, Derks, and Sakhniuk (2022) offer a comprehensive guide to building cross-platform JavaScript applications with React and React Native.

3. RELATED WORK

Comparing PWAs and React Native platforms can help developers make informed decisions when choosing the best platform for their specific use case. While both platforms offer cross-platform app development, they have unique features and

ISSN: 2473-4901
v9 n5944

limitations. This comparison can also save time, effort, and resources by avoiding potential pitfalls or limitations of one platform.

Some researchers have compared the performance and usability of cross-platform approaches. Fournier (2020) compares the smoothness of PWAs and native mobile applications on Android, while Botella, Escribano, and Penalver (2016) discuss selecting the best mobile framework for developing web and hybrid mobile apps. In addition, there have been studies on the practical use of cross-platform mobile application development in the industry. Zohud and Zein (2021) present a multiple case-study analysis of cross-platform mobile app development in the industry. Dabit (2019) guides developing iOS and Android apps with JavaScript using React Native. Some experts have pointed out weaknesses and restrictions in earlier cross-platform mobile application development work. Lee et al. (2018) discuss obstacles in using PWA and offer suggestions to overcome them. An improved approach for enhancing web application and browsing performance using service workers is put forth by Pande et al. (2018).

Table 1 compares several research studies that conducted comparative studies of cross-platform mobile apps, focusing on aspects such as usability, security, and challenges of cross-platform development. However, despite the availability of such research, there is a gap in the literature in terms of direct comparisons between PWA and React Native platforms in the areas of architecture, performance, and user experience.

Criteria	Botella et al. (2016)	Zohud & Zein (2021)	de Andrade Cardieri & Zaina (2018)
Architecture	No	No	No
Performance	Yes	Yes	No
User Experience	Yes	No	Yes

Table 1: Related Work Summary

4. APPROACH

We compare two popular cross-platform app development approaches: PWAs and React Native mobile apps. We aim to evaluate these platforms regarding architecture, performance, and user experience. Then, we guide developers on which platform may best suit their specific needs.

To conduct this study, we follow the steps outlined below, summarized in Figure 1.

Step 1: Creating the Web App
Our first step is to create a simple web app using React framework. We reference a similar example, Cities, in Dabit's book (2019). The app allows users to add cities, countries, and locations to each city. We use this web app as the basis for our PWA and React Native mobile app versions.

Step 2: Converting the Web App to a PWA
We use modern web development techniques such as service workers, caching, and responsive design to convert the web app to a PWA. We use GitHub codespaces as the container to develop the PWA and ensure that it is installable, works offline, and is responsive across various devices. We also use tools such as Lighthouse to evaluate the performance and user experience of the PWA version of the app.

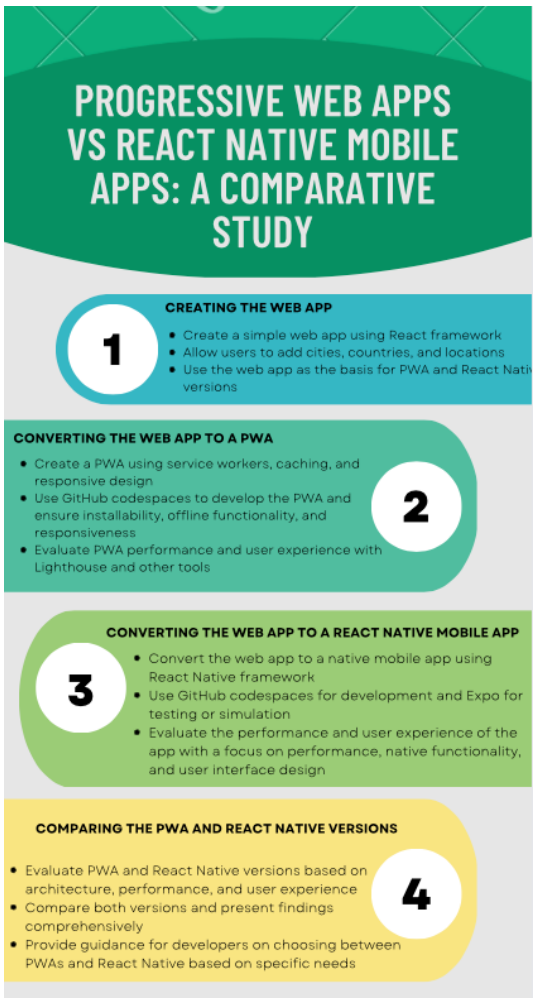

Figure 1 Progressive Web Apps vs React Native Mobile Apps: A Comparative Study

ISSN: 2473-4901
v9 n5944

Step 3: Converting the Web App to a React Native Mobile App

We use the React Native framework to convert the web app to a native mobile app that can run on iOS and Android devices. We use GitHub codespaces as the container to develop the React Native app and Expo as the client to test or simulate the app. We evaluate the performance and user experience of the React Native app, paying particular attention to aspects such as performance, native functionality, and user interface design.

Step 4: Comparing the PWA and React Native Versions
Once we have created both the PWA and React Native versions of the app, we evaluate them based on the criteria mentioned above. We compare both versions' architecture, performance, and user experience and comprehensively present our findings. We also guide developers in choosing between PWAs and React Native mobile apps based on their specific needs.

Design
The application follows a three-tier architecture with a Node.js and MongoDB backend and two frontend interfaces built using React for a PWA and a React Native as Mobile App. The backend and frontends communicate via RESTful Application Programming Interface (API) endpoints. Figure 2 illustrates the application's architecture:

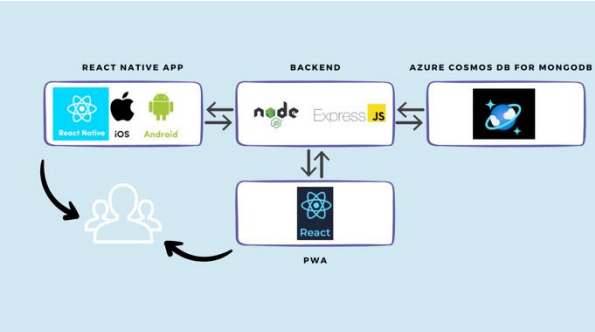

Figure 2 Architectural Diagram

Functionality
The application allows users to add new cities to the database, along with their country and a list of locations within the city. Each location has a name and additional information. Users can also view a list of all cities in the database and select a city to view its details, including the list of locations within the city. In addition, users can add new locations to an existing city. The React frontend provides a user interface for accessing the application through a web browser, while the React Native frontend allows users to access the application through a mobile device.

Deployment
The application is deployed on Azure App Service, with the backend API hosted on a Linux App Service Plan and the database hosted on Azure Cosmos DB for MongoDB. The React frontend is deployed as a progressive web app accessible through a web browser. In contrast, the React Native frontend is deployed as a mobile app accessible to IOS and Android platforms.

Implementation
The implementation of the application follows a three-tier architecture, consisting of a Node.js, a MongoDB backend, and two frontend interfaces built using React for the PWA and React Native for the mobile app. The backend and the frontend communicate with each other through RESTful API endpoints. The overall implementation involves the following steps: setting up the backend, developing the React web frontend, and creating the React Native mobile app frontend.

Backend Implementation
The backend is built using Node.js and Express, with the database hosted on Azure Cosmos DB for MongoDB. The backend API provides the necessary endpoints to handle various operations, such as adding a city, adding a location to a city, retrieving a list of cities, and retrieving city details.

The Node.js runtime environment is installed, and the Express framework is used to handle API routing and request handling. The Azure Cosmos DB for MongoDB is provisioned as the database service, as shown in Figure 3, providing a fully managed cloud database solution. The backend is connected to the database through Mongoose for data modeling and interaction.

ISSN: 2473-4901
v9 n5944

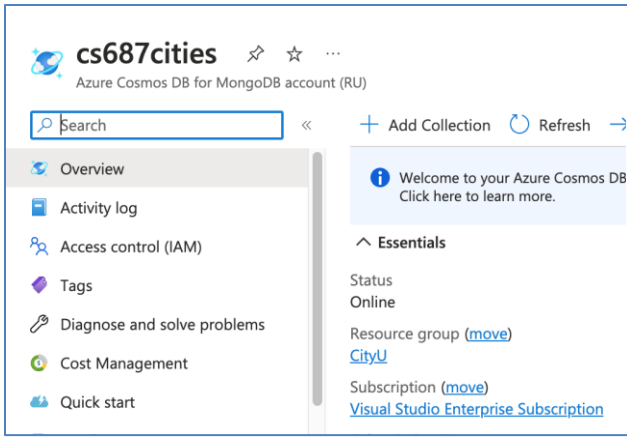

Figure 3 Azure Cosmos DB for MongoDB (cities)

The API routes are implemented using Express, with appropriate route handlers defined for each endpoint. The routes handle incoming requests, interact with the database using Mongoose models, and return the corresponding responses. The backend structure is shown in Figure 4.

React Web Frontend Implementation

The React web frontend is developed using Create React App, a popular toolchain for building React applications. The frontend provides a user-friendly interface accessible through a web browser. It allows users to interact with the application, view a list of cities, add new cities, view city details, and add locations to cities.

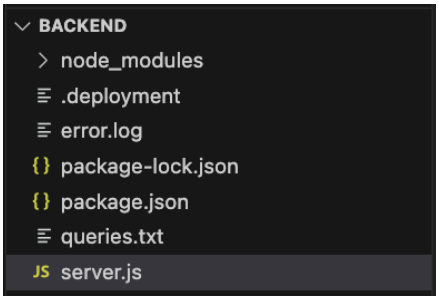

Figure 4 Backend structure

The frontend utilizes React Router for client-side routing, enabling navigation between different views without page refresh. The routing configuration is set up to correspond to the different application features, such as displaying the list of cities, showing city details, and adding new cities and locations. The frontend structure is shown in Figure 5.

React components are created for each view and functionality. These components are integrated with the backend API endpoints to fetch and update data as necessary.

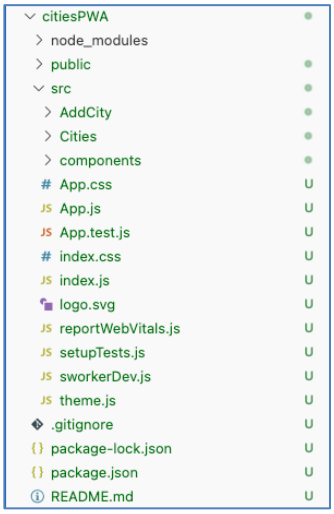

Figure 5 Frontend Structure

Mobile App Frontend Implementation

The React Native mobile app frontend is developed using Expo, a framework that simplifies the creation of cross-platform mobile applications. The frontend provides a native mobile app experience for users, accessible on both iOS and Android platforms. It offers similar functionalities to the web frontend, allowing users to view cities, add new cities, view city details, and add locations to cities.

React Navigation is utilized for client-side routing within the mobile app. The app is organized into screens, each corresponding to a specific view or functionality. The navigation stack is set up to handle the transition between screens, enabling smooth navigation and interaction. The project structure for the mobile app is like the react structure, which is shown earlier in Figure 5.

Similar to the web frontend, react components are created for each screen. These components communicate with the backend API endpoints to fetch and update data, ensuring consistency and synchronization between the mobile app and the backend.

Deployment

The application is deployed on Azure App Service, a fully managed platform for hosting web applications. The backend API is hosted on a Linux App Service Plan as shown in Figure 6, while the React web frontend is deployed as a progressive web app accessible through web browsers. The React Native mobile app frontend is deployed through Expo publish. Expo provides this straightforward way to share our React Native app without going through the app store approval process.

ISSN: 2473-4901
v9 n5944

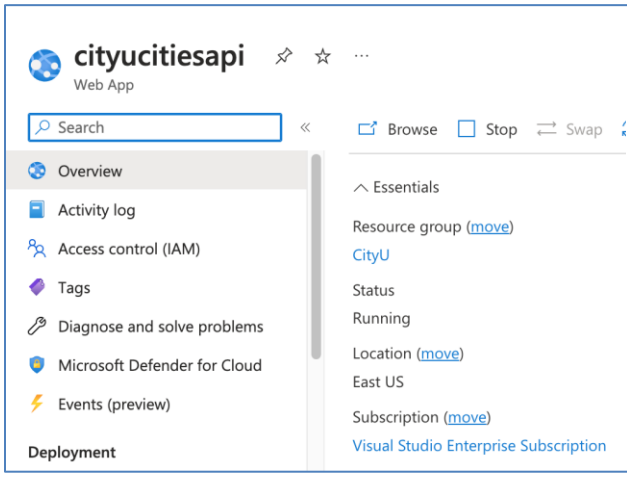

Figure 6 Backend API

The deployment involves configuring the Azure App Service with the necessary runtime environment and dependencies, as well as setting up the appropriate deployment pipelines.

Screens

The application consists of multiple screens that provide a user-friendly interface for interacting with the system. Each screen serves a specific purpose and allows users to perform various actions and view relevant information. Figures 7 and 8 are the screens implemented in the application for adding a city for each platform:

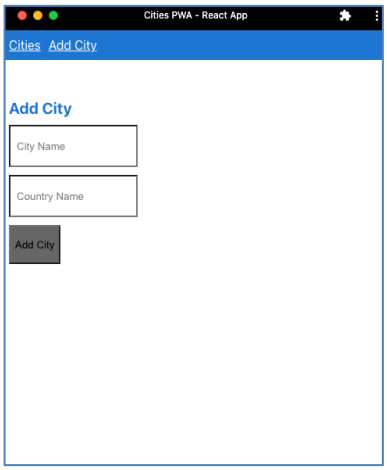

Figure 7 Add City (PWA)

Figure 8 Add City (IOS mobile)

5. DATA COLLECTION

Architecture

To examine the architectural differences between PWAs and React Native mobile apps, we conduct a thorough data collection process. The objective is to gather insights into the architectural aspects that distinguish these two cross-platform app development approaches.

1. Literature Review: We extensively reviewed academic papers and documentation from reputable sources to understand the underlying architectural principles and design patterns associated with PWAs and React Native apps. This literature review provided a foundation for identifying the key architectural differences between the two platforms.
2. Experimental Setup: To gain practical insights, we set up a controlled environment where we develop the cities application as both PWA and React Native mobile app.
3. Development Process: For PWA development, we utilize modern web development technologies such as HTML, CSS, and JavaScript. We follow best practices for implementing responsive design, service workers for offline functionality, and caching mechanisms to enhance performance. The development process takes advantage of GitHub codespaces as a container for seamless collaboration and version control. For the React Native mobile app development, we employ the React Native framework along with JavaScript and native components. Expo, a popular toolchain, is used as the client to test and simulate the app on both iOS and Android devices. The

development process leverages GitHub codespaces to streamline the workflow and facilitate code sharing and synchronization.
4. Metrics and Measurements: To gather quantitative data for architectural differences, we measure various performance and memory utilization metrics.

Following this data collection process, we can obtain comprehensive information about the architectural differences between PWAs and React Native mobile apps. The collected data serve as a foundation for the subsequent data analysis phase, where we examine and compare the architectural aspects of these platforms in detail.

User Experience
The objective is to gather insights into both platforms' user-centric features and characteristics. The data collection phase involves the following steps:

1. User Experience Design: We review user experience design principles and best practices specific to PWAs and React Native apps. This step includes an examination of design guidelines and usability standards. The purpose is to understand the fundamental principles of creating engaging and intuitive user experiences on each platform.
2. Performance and Responsiveness: To evaluate the performance and responsiveness of PWAs and React Native apps, we collect data on various aspects. This step included measuring the app's loading speed, rendering time, and responsiveness to user interactions. We use tools such as Lighthouse to gather performance-related metrics.
3. Native-Like features: We compare the native-like features of PWAs and React Native apps. This step involves identifying and assessing the functionality and behavior that closely resembles that of a native mobile app. We examine features such as offline capabilities and background sync of the data.

Following this data collection process, we can obtain comprehensive information about the user experience aspects of both PWAs and React Native mobile apps.

Performance
To evaluate the performance of PWAs and React Native mobile apps, we measure key performance metrics using the Lighthouse tool. The following metrics are collected for both platforms:

1. First Contentful Paint (FCP): This metric measures the time taken for the first piece of content to appear on the screen. It indicates how quickly the app starts rendering content to the user.
2. Largest Contentful Paint (LCP): LCP measures the time it takes for the largest element on the screen to become visible. It provides insights into the perceived loading speed and responsiveness of the app.
3. Total Blocking Time (TBT): TBT measures the amount of time the main thread is blocked and is unable to respond to user interactions. A lower TBT indicates a smoother and more responsive user experience.
4. Cumulative Layout Shift (CLS): CLS measures the visual stability of the app by tracking unexpected layout shifts. It quantifies the extent to which elements on the screen move or shift unexpectedly, potentially causing user frustration.
5. Speed Index: The Speed Index measures how quickly the contents of a web page are visually populated. It provides an overall measure of the app's perceived loading speed.

6. DATA ANALYSIS

Architecture
We examined the collected data to identify and analyze the architectural differences between PWAs and React Native mobile apps. The data are analyzed using qualitative and quantitative methods to gain insights into each platform's underlying architectural principles and design patterns.

1. Based on the literature review, we identify key architectural differences between PWAs and React Native apps. We analyze the identified architectural principles and design patterns to understand how each platform handles aspects such as component structure, data management, navigation, and integration with device capabilities.
2. The controlled environment in which we develop the PWA and React Native mobile app versions of the cities application allows us to analyze the architectural choices made during development. We examine the overall application structure, code organization, and libraries.

User Experience
To comprehensively evaluate and compare the user experience aspects of Progressive Web Apps (PWAs) and React Native mobile applications, we

ISSN: 2473-4901
v9 n5944

conducted a thorough assessment focusing on features, performance, responsiveness, and other key metrics.

The evaluation involved an in-depth analysis of various aspects pertaining to user experience. This encompassed a detailed review of design principles, best practices, and platform-specific functionalities relevant to PWAs and React Native apps. We compared loading speed, rendering time, and responsiveness to user interactions between PWAs and React Native apps. This analysis provided invaluable insights into how each platform handles performance optimizations and contributes to the delivery of a seamless and satisfying user experience.

We also assessed and compared the native-like features of PWAs and React Native apps, with a specific focus on functionalities such as offline capabilities and background synchronization of data. Through this examination, we gauged the effectiveness of each platform in providing these features and evaluated their overall impact on the user experience.

Performance

Based on the collected data, we compare the performance of PWAs and React Native mobile Apps. The findings provide valuable insights into how each platform performs regarding initial loading speed, content rendering, responsiveness, and overall stability.

To evaluate the performance of the PWA and React Native app, we collect and analyze various performance metrics using the Lighthouse tool. Metrics for PWA are shown in Figures 9 and 10.

Figure 9 PWA performance metrics

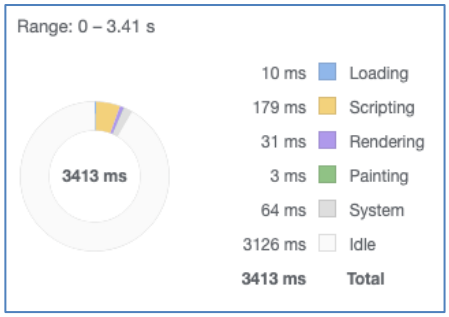

Figure 10 PWA Summary

Metrics for React Native App are shown in Figures 11 and 12.

The performance analysis of the PWA and React Native app reveals that the PWA exhibits faster First Contentful Paint (FCP), Speed Index, and comparable scores in metrics such as Largest Contentful Paint (LCP), Total Blocking Time (TBT), and Cumulative Layout Shift (CLS). The PWA demonstrates a quicker rendering of initial content, faster overall loading and rendering speed, and achieves a good performance score in Lighthouse. While both platforms show similar patterns in terms of loading, rendering, and painting times, the PWA showcases better performance overall.

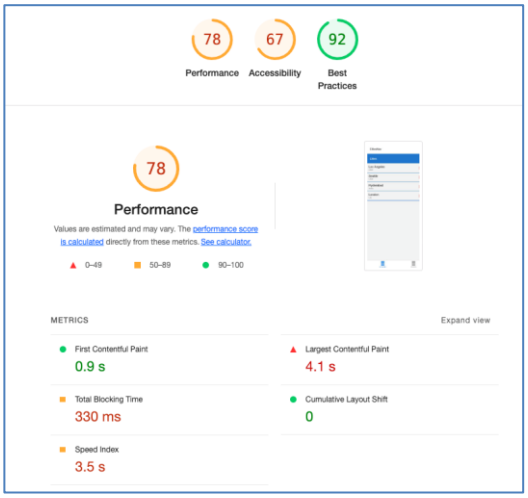

Figure 11 React Native performance metrics

ISSN: 2473-4901
v9 n5944

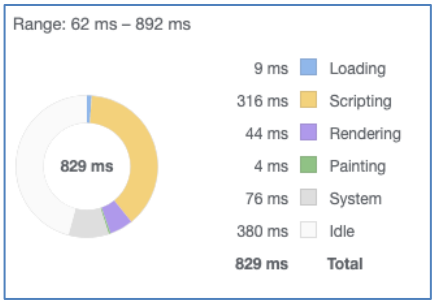

Figure 12 React Native Summary

7. FINDING

Architecture

1. Component Structure: In terms of component structure, we find that React Native apps follow a more native-like approach, where the UI components are rendered using native platform components. On the other hand, PWAs use web components and HTML elements to build the user interface.
2. Data Management: React Native apps typically leverage state management libraries like Redux to handle data flow and state updates. PWAs, on the other hand, can use similar state management approaches but also have the option to utilize client-side storage mechanisms like IndexedDB and localStorage. This option allows PWAs to store data locally and provide offline capabilities.
3. Navigation: React Native provides a built-in navigation library (React Navigation) that offers a seamless and intuitive way to handle app navigation. PWAs often rely on browser-based navigation using URLs and anchor tags. While both approaches can achieve effective navigation, React Native's built-in navigation library provides more control and flexibility for complex navigation scenarios.

User Experience

1. Design and User Interface: Both PWAs and React Native apps can create visually appealing and engaging user interfaces. However, PWAs often face challenges in achieving the exact look and feel of native mobile apps due to limitations in accessing native device APIs and UI components. React Native apps, being closer to the native platform, can deliver a more native-like user interface and user experience.
2. Native-Like Features: While PWAs have made significant progress in bridging the gap with native apps, there are still some limitations in accessing certain native-like features. Although PWAs have made strides in incorporating these features through browser APIs, they may still encounter restrictions and variations across different browsers and devices.

Performance

Our data analysis reveals that PWAs and React Native apps can provide comparable performance and responsiveness. However, PWA shows slightly better performance when compared to React Native App. These results suggest that well-optimized PWA can provide a similar experience as native Apps in terms of performance. It is important to note that performance can vary depending on various factors, including the specific app, its complexity, the target devices, and the optimizations implemented.

Overall, our findings indicate that React Native apps offer a more native-like experience and have better access to native device capabilities. PWAs, on the other hand, provide the advantage of cross-platform compatibility and the ability to deliver web-based experiences across multiple devices.

Based on our findings, we provide recommendations to assist developers in choosing the most suitable platform for their specific use case. These recommendations consider architectural preferences, performance requirements, and user experience goals. Additionally, we highlight the importance of considering factors such as development resources, time constraints, and target platform compatibility when making the decision.

By providing this guidance, we aim to empower developers to make informed decisions and optimize their development processes, resulting in successful and efficient app deployments. The insights presented in this paper contribute to the broader understanding of cross-platform app development and assist developers in maximizing the potential of their projects. Also, by incorporating these findings into their teaching, educators can equip students with the knowledge and skills to make informed decisions in real-world development scenarios.

By following the recommended guidelines, developers can select the platform that aligns with their project goals, resources, and target audience, leading to the development of high-quality apps that deliver exceptional user experiences.

ISSN: 2473-4901
v9 n5944

9. FUTURE WORK

While this study provides valuable insights into the architectural differences, performance, and user experience aspects of PWAs and React Native mobile apps, there are specific areas that warrant focused investigation in future research:

1. Platform-Specific Functionality Testing: A detailed examination of unique features and functionalities specific to each platform could provide deeper insights into their capabilities and enable developers to leverage them effectively.
2. Native Device API Integration: Exploring in-depth integration with native device APIs can unlock additional potential for both PWAs and React Native apps. This could involve optimizing access to device-specific resources and services.
3. Performance Optimization Strategies: Further research into advanced performance optimization techniques tailored to each platform could yield valuable strategies for developers aiming to enhance the responsiveness and efficiency of their applications.

By further exploring these specific areas we plan to expand our understanding of PWAs and React Native mobile apps, offering developers actionable insights to unlock their full potential and deliver exceptional app experiences.

10. REFERENCE

Boduch, A., Derks, R., & Sakhniuk, M. (2022). React and React Native: Build Cross-Platform JavaScript Applications with Native Power for the Web, Desktop, and Mobile (4th ed.). Packt Publishing.

Botella, F., Escribano, P., & Penalver, A. (2016). Selecting the Best Mobile Framework for developing web and Hybrid Mobile Apps. In Proceedings of the XVII International Conference on Human Computer Interaction (pp. 102-109). https://doi.org/10.1145/2998626.2998648

Dabit, N. (2019). React native in action: Developing IoS and Android apps with JavaScript. Manning.

de Andrade Cardieri, G., & Zaina, L. M. (2018). Analyzing user experience in mobile web, Native and progressive web applications. Proceedings of the 17th Brazilian Symposium on Human Factors in Computing Systems. https://doi.org/10.1145/3274192.3274201

Domes, S. (n.d.). Progressive web apps with react: Create lightning fast web apps with native power using react and Firebase. Packt.

Fournier, C. (2020). Comparison of Smoothness in Progressive Web Apps and Mobile Applications on Android (Dissertation). Retrieved from http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-283653

Hansson, N., & Vidhall, T. (2016). Effects on performance and usability for cross-platform application development using React Native (Dissertation). Retrieved from http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-130022

Lee, J., Kim, H., Park, J., Shin, I., & Son, S. (2018). Pride and prejudice in progressive web apps. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (pp. 706-721). https://doi.org/10.1145/3243734.3243867

Love, C. (2018). Progressive web application development by example: Develop fast, reliable, and engaging user experiences for the web. Packt Publishing.

Pande, N., Somani, A., Prasad Samal, S., & Kakkirala, V. (2018). Enhanced web application and browsing performance through service-worker infusion framework. 2018 IEEE International Conference on Web Services (ICWS). https://doi.org/10.1109/icws.2018.00032

Subramanian, V. (2019). Pro Mern Stack: Full Stack Web App Development with Mongo, express, react, and node.

Zohud, T., & Zein, S. (2021). Cross-platform mobile app development in industry: A multiple case-study. International Journal of Computing, 46–54. https://doi.org/10.47839/ijc.20.1.2091