International Journal of Research in Engineering and Science (IJRES)
ISSN (Online): 2320-9364, ISSN (Print): 2320-9356
www.ijres.org Volume 12 Issue 3 ǁ March 2024 ǁ PP. 174-180

Progressive Web App (PWA): Optimal Strategies & Challenges

Chakradhar Avinash Devarapalli, Software Developer
Email:avinashd7[at]gmail.com

Abstract - *Progressive Web Apps (PWAs) have recently emerged as an innovative approach to close the gap between web and native applications by incorporating both of their functionalities, offering enhanced user experiences with features such as offline functionality, push notifications, and app-like interactions within the web. This paper aims to explore the strategies and challenges involved in the development of PWAs, using insights from recent advancements and academic literature. The study identifies key usable/viable strategies, including service worker implementation, performance optimization, and security considerations, as essential pillars for successful PWA development and implementation. Additionally, the research highlights significant challenges such as browser compatibility, service worker complexity, security concerns and user engagement, underscoring the need for innovative solutions and best practices*.

Keywords – *Progressive Web App, PWA, web app manifest, PWA challenges, PWA strategies, service worker, cross-platform development, feature detection, Performance optimization*

Date of Submission: 05-03-2024 Date of acceptance: 18-03-2024

I. Introduction:

The dawn and rise of web development has experienced a very stimulating shift in recent years with the induction and advent of Progressive Web Apps (PWAs) [1]. Representing an amalgamation of web and mobile based tech, PWAs aim to provide users with an experience similar to native web applications while retaining and maintaining the accessibility and flexibility provided by live-hosted web applications and platforms [1]. Defined as "user experiences that have the reach of the web and are reliable, fast, and engaging" by Google Developers. PWAs embody a vast treasure of design principles, development methodologies, and technical/sub-technical standards aimed towards enhancing the performance, reliability, and engagement of web applications across diverse devices [3, 2, 4].

The conceptualization and inhibition of Progressive Web Apps traces back to the gradual work of Alex Russell and Frances Berriman in 2015 valuing a strife moment in the evolution of web development practices and methods [3]. Unlike conventional websites, which often grapple with performance bottlenecks, limited functionality, and the absence of offline support, PWAs offer users a seamless and immersive experience irrespective of network connectivity [5, 6]. PWAs resolve the challenges associated with native app development, such as platform-specific frameworks, app store approval processes, and installation barriers.

Previous studies provide insights into the technical aspects and implications of PWA. The Emergence of Progressive Web Apps delve into the technological advancements and design principles underpinning PWAs, emphasizing their potential/capability to close the gap between web and mobile experiences [3]. Insights on PWAs provides a comprehensive exploration of PWA development techniques, best practices, and case studies

The adoption of PWAs has been popular across diverse sectors and with developers working on the technology to make it better, with organizations leveraging this technology to deliver engaging web experiences to their user base. For instance, Twitter Lite, a PWA developed by Twitter, provides users with a lightweight, data-efficient alternative to its native mobile app, enabling seamless access to timelines and tweets even under low-bandwidth conditions.

In a similar manner, Forbes, a widespread media outlet focusing on financial data and company growth, implemented a PWA to enhance page load times and user engagement, resulting in a significant uplift in interactions and revenue, although it might not be a complete app as discussed but still passes as a PWA. Against this research, this paper aims to explore the implementation of Progressive Web Apps and their transformative impact on web applications. By synthesizing insights from academic literature, analyzing case studies, and evaluating the challenges and opportunities associated with PWA development, this paper seeks to put forward the future trajectory of web development and the potential of PWAs to redefine the digital landscape.

II. Analytical Review

Progressive Web Apps (PWAs) have proved themselves as a working and practical approach to close the gap between web and mobile experiences by offering a framework that allows both paradigms to work in a single domain, offering users the reliability and experience of native apps while withholding the accessibility and versatility of web-based platforms [4, 5, 1]. Despite the benefits, PWAs offers several challenges that persists in their implementation and adoption, requiring a deeper understanding of the hassles facing this innovative technology and the strategies employed to overcome them.

One of the most prominent and surfacing challenges in PWA development revolves around optimizing user experience and performance to deliver fast and responsive output to users. Poor performance can undermine user engagement and retention which can lead to poor results and bad turnover, particularly on mobile devices with limited processing power and network bandwidth which is common in areas with network limitations [9]. Strategies for performance optimization include code minification, lazy loading of assets, and efficient caching strategies using Service Workers as reviewed in past literature posted by various experienced authors [6, 7, 8]. Another relatively important aspect of PWA development obtained from past literature is providing robust offline support, enabling users to access content and functionality even in the absence of an internet connection by employing service workers and other JavaScript elements to keep the platform running. Implementing effective offline caching strategies using Service Workers is essential to ensure error-less offline experiences [9]. Managing offline data synchronization and conflict resolution can pose significant challenges, particularly in applications with dynamic content and user-generated data [10].

Another aspect relates towards ensuring cross-browser compatibility in PWA development to reach a broad audience across diverse devices and platforms which is main reasoning and objective of fairly well working PWAs. However, differences in browser capabilities and support for web standards can complicate development and testing efforts where an internet browser such as IE (Internet explorer) may not be up to date to ensure compatibility with JS features. Strategies for addressing cross-browser compatibility issues include feature detection, PolyFills, and progressive enhancement techniques [11].

Security and privacy factors are paramount in PWA development to protect user data and reduce the risk of security vulnerabilities as PWAs can be prone to external malware and brute attacks, which may hinder their overall abilities to perform as advertised. PWAs are served over HTTPS to ensure data integrity and confidentiality, but developers must also address security threats such as cross-site scripting (XSS) [12] and cross-origin resource sharing (CORS) issues [13]. Strategies for enhancing security include implementing content security policies (CSP) [12, 8, 13], enforcing secure communication protocols, and conducting regular security audits and penetration testing which is vital to ensure the working performance of a PWA.

PWAs most commonly rely on web-based distribution channels which are responsible for potential clientele outreach and lead turnovers, this results in raising challenges related to discoverability and installation friction. Users may struggle to find PWAs amid the vastness of the web unless they are advertised heavily by the publishing corporation, hindering adoption and engagement.

III. Identified Challenges

Identifying/observing the challenges related to the conversion of a web app into a Progressive Web App (PWA) is crucial for understanding the complexities involved in the process. Here are some of the key challenges associated with implementing PWAs:

3.1 Service Worker Complexities

The complexity of service workers lies in the intricate/in-depth nature of implementing and managing these JavaScript files within Progressive Web Apps (PWAs). Service workers play an extremely important role in allowing advanced features such as offline functionality, push notifications, and background synchronization, thus significantly improving the user experience and user satisfaction along with device support [1, 5]. In contrast, their implementation requires intricate attention to detail, including caching strategies, network interception, and event handling. Service workers also operate independently of the main browser thread and the web-app, introducing complexities in debugging, error handling, and version management.

3.2 Performance Optimization

Performance optimization inhibits a multifaceted challenge in the development of Progressive Web Apps (PWAs) which has the potential to halt development and even reduce responsiveness on older devices, given the ideology to deliver fast, responsive experiences across a diverse array of devices and network conditions [16]. The challenge arises from the need to balance rich functionality and immersive features with the imperative of minimizing loading times and optimizing resource usage that requires advanced frameworks and resource-heavy functions of JS or nay other scripting/programming language, some of which may not be supported of legacy or older browsers/devices [17]. Achieving optimal performance necessitates sharp attention

to various aspects of web development, including asset optimization, lazy loading of resources, and efficient caching strategies.

3.3 Security & Data Authentication

Security concerns portray a paramount challenge in the development and deployment of Progressive Web Apps (PWAs) on high- and low-end devices, where the fundamental goal is to ensure the protection of sensitive data and user privacy in an online environment as is suggested and required by law in some regions. This challenge is multivariate, encompassing various variables such as secure communication protocols, data encryption, and protection against common web security vulnerabilities [10, 9]. Using secure HTTPS protocols is required and important to safeguard data integrity and prevent tampering of important and sensitive data during transmission. PWAs must employ robust authentication mechanisms, such as token-based authentication or OAuth (Google OAuth2 etc.), to verify user identities and protect against unauthorized access [16]. Features such as push notifications and background synchronization introduce potential security risks, requiring strict and robust measures to reduce the threat of abuse or exploitation. Adhering to established security best practices, such as content security policies (CSP) and sub-resource integrity (SRI), is essential to prevent cross-site scripting (XSS) and cross-origin resource sharing (CORS) attacks and other common security vulnerabilities.

IV. Viable Strategies

4.1 Web App Manifest File

The Web App Manifest is a JSON file written in JavaScript that provides metadata about a Progressive Web App (PWA), which enables browsers to identify and install the web application on users' devices converting it into an effective PWA [9, 16]. The technical aspects of implementing a Web App Manifest includes highlighting its structure, key properties, and implications for PWA development. The Web App Manifest file contains a set of key-value pairs defined in JSON format (think of it as a NoSQL database to store values), specifying various attributes of the PWA, including its name, icons, colors, display preferences, and navigation behavior [2, 17]. The manifest file is typically named *manifest.json* and resides at the root of the application directory through which the actual web app runs.

The *manifest.json* file in *figure 1* outlines metadata for a PWA. Key properties include the app's *name*, *short name*, *description*, *icons*, *start URL*, *background color*, *theme color*, *display mode*, *orientation*, *scope*, and *service worker*. The icons array specifies multiple icon images at different sizes and resolutions, while the service worker properties define the location and scope of the service worker script which will be implemented to make the PWA include offline functionality.

4.2 Service Workers

Service workers are an important and fundamental component of modern web development practices, enabling developers and builders to create robust, responsive, and *offline-capable* web applications that can be converted into Progressive Web Apps (PWAs) [16]. Service workers are generally referred as some JavaScript files that run in the background of a web application, separate from the main browser thread [9, 18, 12], they are not in the active running directory of the web application but rather they work in the background. They act as programmable proxies, intercepting and handling network requests from the application, enabling features such as offline support, push notifications, and background synchronization for already loaded and to-be loaded content [13, 7]. Service workers operate independently of the web page, allowing developers to implement advanced caching strategies, perform background tasks, and manage application state even when the browser is closed which in actuality defines the purpose of a PWA.

The code-block in *figure 2* represents an extremely simple service worker which undertakes the responsibility of caching the defined URLs when the install event is triggered. This simple code-block can be implemented in the PWA in the form of a JS script file within the root directory of the main program and linked to run in offline conditions.

4.3 Progressive Enhancement

Progressive Enhancement in PWAs is a web development strategy used by most developers that prioritizes the implementation of a baseline experience accessible to all users, regardless of their device, browser, or network capabilities (basically adding responsiveness in a gradual manner). This technical discourse explores the foundational principles and implementation techniques of Progressive Enhancement, elucidating its role in fostering inclusive and resilient web experiences. There are several methods and ideologies that can be implemented when it comes to progressive enhancement such as graceful degradation, functionalities and layered enhancements.

Layered enhancement includes the involvement of additional layers of presentation, interactivity, and functionality using modern web technologies such as CSS3, JavaScript, and HTML5. These enhancements

should be applied progressively, based on the capabilities of the user's device and browser. Layered enhancement works on the principle of providing a viewpoint available to all respective devices.

Graceful degradation provides fallbacks or alternative experiences for users with less capable devices or older browsers that do not support modern web technologies which allows the users with older devices to access all the functionality of the PWA just with or without the full potential of the responsive UI. This ensures that all users can access and interact with the content, even if they cannot take advantage of the latest enhancements.

4.4 Feature Detection

Feature detection is a technique/framework used in web development to determine the capabilities of the user's device and browser and calculate the required resources needed for the user to access/use the said PWA according to which, the PWA can be developed, this allows developers to adaptively enhance the web experience based on these capabilities.

4.4.1 Native Feature Detection

NFD utilizes built-in JavaScript methods or APIs to detect support for specific features or APIs which might or might not be supported on an array of legacy devices. For example, developers can use the *typeof* operator to check for the existence of JavaScript objects or properties, or use feature detection methods provided by APIs such as the Document Object Model (DOM) or the Web APIs.

4.4.2 Polyfills

Polyfills or shims are used to provide fallback or legacy support for features that are not natively supported by the user's browser. Polyfills are JavaScript libraries that emulate the functionality of modern features in older browsers providing an alternative to the actual functionality of the PWA, allowing developers to use modern web standards while maintaining compatibility with legacy browsers. The code depicts a polyfill which works on the *fetch* API to see if it exists on a browser, if it does not, it creates a new *XMLHttpRequest* object to make the network request. It sets up event handlers for *onload* and *onerror* to handle the response or error then it opens the *XMLHttpRequest* with the provided method (defaulting to '*GET*') and *URL*.

V. Charts & Blocks

Table 1 Challenges & Impact Factor

Challenges	Factor of Impact
Service Worker Complexity [12, 10]	High - Impacts offline functionality, performance, and reliability
Performance Optimization [15]	High - Impacts user experience, engagement, and retention
Security Concerns [14]	High - Impacts user trust, data security, and compliance
Browser Compatibility [10]	Medium to High - Impacts reach and user experience across different browsers

Figure 1 PWA Code

```
{
 "name": "PWA",
 "short_name": "EPWA",
 "description": "An example Progressive Web App",
 "icons": [
   {
    "src": "/icons/ii.png",
    "sizes": "72x72",
    "type": "image/png"
   },
   {
    "src": "/icons/pp.png",
    "sizes": "512x512",
    "type": "image/png"
   }

],
```

```
  "start_url": "/index.html",
  "background_color": "#000000",
  "theme_color": "#1368E9",
  "display": "standalone",
  "orientation": "portrait",
  "scope": "/",
  "serviceworker": {
    "src": "/sw.js",
    "scope": "/"
  }
}
```

Figure 2 Service Worker API Code

```
const CACHE_VERSION = 'v1';
const CACHE_NAME = 'my-cache-' + CACHE_VERSION;
const urlsToCache = [
  '/',
  '/index.html',
  '/styles/main.css',
  '/scripts/main.js',
  '/images/logo.png'
];

self.addEventListener('install', event => {
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then(cache => {
        return cache.addAll(urlsToCache);
      })
  );
});
```

Figure 3 PolyFill Code

```
if (!window.fetch) {
  window.fetch = function(url, options) {
    return new Promise(function(resolve, reject) {
      var xhr = new XMLHttpRequest();
      xhr.onload = function() {
        if (xhr.status )= 200 && xhr.status < 300){
          resolve(xhr.response);
        } else {
          reject(new Error(xhr.statusText));
        }
};
      xhr.onerror = function() {
        reject(new Error("Network Error"));
      };
      xhr.open(options.method || 'GET', url);
      for (var header in options.headers || {}) {
        xhr.setRequestHeader(header, options.headers[header]);
      }
      xhr.send(options.body || null);
    });
  };
}
```

VI. Discussion

As the entire viewpoint of web development continues to evolve, Progressive Web Apps (PWAs) emerge as an important factor, offering the promise of native-like experiences on the web available to all in terms of responsiveness and legacy support. While the future of PWA development appears to be positive and inclined towards betterment, it is not without its challenges. Some challenges that were identified include browser fragmentation and ensuring consistent behavior and performance across diverse browser environments in terms of responsiveness and legacy browser support. Addressing the complexity of service workers, offline data synchronization, and background sync mechanisms portrays significant technical hurdles which seem difficult to mitigate. Moreover, balancing rich functionality with resource constraints, such as limited device memory and processing power, presents ongoing challenges in PWA optimization. Navigating the complexities of PWA implementation and discoverability, including optimizing the Web App Manifest, improving SEO, and streamlining the installation process, requires concerted efforts to enhance PWA visibility and adoption.

In a similar manner, effective strategies are essential to overcome the hurdles/complexities of PWA development and achieving the full potential of this framework. Some identified strategies include prioritizing performance optimization through efficient caching strategies (Optimizing service workers), lazy loading techniques, and responsive design principles. Implementing robust security measures, such as HTTPS adoption, content security policies (CSP), and encryption protocols, is important to safeguard user data and ensuring trust in PWAs as discussed and identified in the research.

VII. Conclusion

The future of PWA development holds an overwhelming potential for innovation and advanced usability, which would be driven by a concerted effort to overcome challenges and capitalize on emerging opportunities by implementing the most viable strategies to obtain the maximum output out of each PWA. By embracing effective strategies, leveraging technological advancements, and addressing challenges, developers can pave the way for the continued evolution of PWAs in terms of improving user satisfaction and shortening the learning curve to develop effective PWAs, this also offers immersive, reliable, and engaging web experiences that rival native applications in functionality and performance. Through collaboration between potential developing parties, user-centric design, and continuous corrective iterations of identified hurdles and hassles, the journey towards realizing the full potential of PWAs as a transformative force in the web ecosystem unfolds, shaping the future of digital experiences in the digital age.

References

[1] B. Frankston, "Progressive Web Apps [Bits Versus Electrons]," *IEEE Consumer Electronics Magazine*, vol. 7, pp. 106-117, March 2018.

[2] D. Sheppard, Beginning Progressive Web App Development, Apress, 2017.

[3] W. Tamire, *Evaluation of Progressive Web Application to develop an Offline-First Task Management App*, 2019.

[4] V. Aguirre, L. Delía, P. Thomas, L. Corbalán, G. Cáseres and J. Sosa, "PWA and TWA: Recent Development Trends," in *Computer Science – CACIC 2019*, Springer International Publishing, 2019, pp. 205-214.

[6] S. Tandel and A. Jamadar, "Impact of progressive web apps on web app development," *International Journal of Innovative Research in Science, Engineering and Technology*, vol. 7, no. 9, pp. 9439-9444, 2018.

[7] R. S. Mishra, "Progressive WEBAPP : Review," *International Research Journal of Engineering and Technology*, vol. 3, no. 6, pp. 3028-3032, 2016.

[8] A. Biørn-Hansen, T. Majchrzak and T.-M. 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*, 2017.

[9] T. Ater, Building progressive web apps: bringing the power of native to the browser, O'Reilly Media, Inc, 2017.

[11] D. Hume, "Trends in Web Standards and techniques," *VINE*, vol. 27, pp. 70-73, 2017.

[12] I. Malavolta, G. Procaccianti, P. Noorland and P. Vukmirovic, "Assessing the Impact of Service Workers on the Energy Efficiency of Progressive Web Apps," in *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, IEEE, 2017.

[13] T. Majchrzak, A. Biørn-Hansen and T.-M. Grønli, "Progressive Web Apps: the Definite Approach to Cross-Platform Development?," in *Proceedings of the 51st Hawaii International Conference on System Sciences*, 2018.

[14] S. Gupta and B. Gupta, "Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art," *International Journal of System Assurance Engineering and Management*, vol. 8, pp. 512-530, 2017.

[15] J. Lee, H. Kim, J. Park, I. Shin and S. Son, "Pride and Prejudice in Progressive Web Apps," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018.

[16] P. Thakur, "Evaluation and implementation of progressive web application," 2018.

[17] I. Malavolta, K. Chinnappan, L. Jasmontas, S. Gupta and K. Soltany, "Evaluating the impact of caching on the energy consumption and performance of progressive web apps," in *Proceedings of the IEEE/ACM 7th International Conference on Mobile Software Engineering and Systems*, 2020.

[18] J. M. Wargo, Learning Progressive Web Apps, Addison-Wesley Professional, 2020.

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

[20] R. Fransson and A. Driaguine, "Comparing Progressive Web Applications with Native Android Applications," 2017.

[21] K. Farrugia, "Model-theoretic semantics for the web," in *Proceedings of the twelfth international conference on World Wide Web - WWW '03*, 2016.