WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal Homepage: www.ijrpr.com ISSN 2582-7421

AIKYA: Decentralized Edge-Hosted Progressive Web Apps

Mrs. Gorisetty Nirosha [1], Pandiri Mahalakshmi [2]

[1, 2] Department of Computer Science and Engineering, GMR Institute of Technology, Rajam, Andhra Pradesh, India

ABSTRACT:

Small-scale retail outlets, local cooperatives, and temporary market stalls often face significant operational friction when deploying modern point-of-sale systems. Mainstream enterprise platforms require uninterrupted internet connectivity, recurring cloud subscriptions, and dedicated hardware terminals. In regions where network coverage fluctuates or data bandwidth is costly, cloud dependency leads to transactional stalls and delayed checkout processes. This paper presents the design and practical implementation of AIKYA, an offline-first decentralized commerce system tailored for micro-merchants. The system converts an ordinary Android smartphone into an on-premise local server, handling transactions and catalog management through an embedded SQLite database. Customers and store associates access the store environment directly through standard web browsers via local Wi-Fi without installing software from application stores. By utilizing progressive web application caching mechanisms, the client interface loads instantly and maintains interactive catalog browsing even during brief signal drops. Field observations show that the system operates comfortably within typical consumer mobile hardware, requiring less than 2 MB of client storage while providing real-time ordering and staff messaging. The proposed architecture offers a viable, zero-cost operational framework for independent retail businesses seeking digital autonomy.

Keywords: Progressive Web Applications, Edge Hosting, Decentralized Commerce, Offline-First Systems, Local Wireless Network, Retail Automation, Service Workers.

1. Introduction

In neighborhood kirana stores, rural agricultural cooperatives, and temporary exhibition stalls across developing economies, digital retail adoption remains uneven. While major supermarket chains and multinational brands deploy cloud-integrated enterprise resource planning suites, local merchants continue to manage stock balances, price lookups, and customer sales on physical paper registers. This continued reliance on manual records is not caused by resistance to technology, but rather by the practical mismatch between available commercial software and the actual operating environment of small retailers. Standard point-of-sale (POS) platforms are built on an assumption of continuous, high-speed broadband internet and require recurring software licensing fees alongside dedicated desktop hardware terminals [2], [13].

In practice, wide-area network coverage in semi-urban and rural localities is frequently unreliable. Network congestion, cellular signal dead zones inside dense concrete shop structures, and unexpected ISP fiber breaks routinely disrupt connectivity. Whenever an internet blackout occurs, cloud-tethered point-of-sale systems fail entirely: merchants cannot access product catalogs, line-item totals cannot be calculated, and customer billing halts. Furthermore, expecting customers to download a separate mobile app from the Google Play Store or Apple App Store merely to view daily store inventory is unrealistic. Shoppers routinely refuse to install single-purpose store applications due to limited phone storage, mobile data costs, and privacy concerns [6], [15].

To address these systemic bottlenecks, this paper presents AIKYA, an applied decentralized retail architecture. AIKYA repositions an ordinary consumer Android smartphone as an autonomous local edge server that runs inside the store premises. Customers and shop staff access the full commercial storefront through standard mobile web browsers over a localized Wi-Fi connection, completely bypassing external cloud infrastructure [4], [8]. By adopting Progressive Web Application (PWA) standards, the interface delivers the responsive experience of a native app while remaining entirely zero-install. The following sections outline the architectural design, field onboarding workflow, empirical observations, and comparative analysis.

2. Background and Operational Context

Current small-business retail software options fall into three predominant architectures, each with distinct trade-offs:

• Centralized Cloud SaaS Platforms: Solutions such as Shopify, Square, and cloud-hosted WooCommerce manage data in remote hyperscale datacenters [2], [19]. Although they offer multi-branch inventory aggregation, their mandatory requirement for unbroken uplinks makes them fragile during network outages. Moreover, monthly subscription tiers impose an unsustainable cost burden on micro-enterprises operating on thin profit margins.

• Standalone Native Mobile Applications: Installing an offline POS app on an Android tablet avoids cloud dependency [12], [13]. However, native apps present severe multi-user distribution hurdles. Allowing walk-in shoppers to browse catalogs or enabling floor assistants to check inventory requires installing and maintaining binary packages across varying device models and operating system versions, creating configuration friction.

• Progressive Web Applications (PWAs): Recent advancements in web standards have bridged the performance gap between web pages and native code [1], [16]. By registering client-side Service Workers, web applications can store application assets in local browser caches, listen for offline events, and launch in full-screen standalone windows without app store approval. However, conventional PWAs still communicate with cloud-hosted backend APIs [7], [22].

AIKYA combines the distribution simplicity of PWAs with local edge computing. By shifting the API server from distant cloud datacenters directly into an on-site Android handset, the entire commerce loop operates autonomously within the merchant's immediate physical vicinity [5], [17].

3. Comprehensive Literature Review and Related Work

The emergence of Progressive Web Applications has prompted significant academic research into offline resilience, caching patterns, and cross-platform usability. Behl and Raj [1] formalized architectural patterns for web background synchronization, illustrating how Service Workers decouple network dispatching from user-perceived thread availability. Kankaala [2] evaluated the transition of consumer e-commerce storefronts to headless web platforms, documenting measurable drops in bounce rates and enhanced conversion agility. Leshchuk et al. [3] implemented a PWA specifically for localized retail inventory tracking, emphasizing that service worker caching reduces server ingress loads by over sixty percent.

In resource-constrained environments, Rivera et al. [4] investigated mobile edge computing architectures for rural medical records, demonstrating that localized nodes can sustain essential record management during prolonged telecommunication outages. Mena et al. [5] developed microservice-oriented PWAs for real-time sensor monitoring, validating that browser-based frontends can interface smoothly with local socket pipelines. The fundamental trade-offs between native codebases and progressive web technologies were systematically examined by Biørn-Hansen et al. [6], who highlighted that while native runtimes excel in raw computational throughput, PWAs offer unmatched distribution parity across operating systems.

Field location services and offline geospatial interactions using PWAs were explored by Pisipati et al. [7], proving that client-side indexed databases can handle continuous reading and writing without active network connections. Security implications of client-side caching were scrutinized by Ahadli [8] and Talukder et al. [9], who analyzed vulnerability surfaces in browser cache partitions and recommended origin validation guards. Lee et al. [10] demonstrated potential attack vectors in permissive Service Worker registrations, underscoring the absolute necessity of strict local-origin scoping.

From an empirical standpoint, Fauzan et al. [11], [14] conducted systematic literature reviews and surveys of production PWA developments, identifying offline fallback consistency and state synchronization as the two most critical implementation hurdles. Ahadli [12] and Samsyudin [13] performed rigorous comparative benchmark studies between native Android binaries and PWA equivalents, confirming that modern JavaScript V8 rendering engines execute DOM updates with fluid interactive responsiveness. Utshaw [15] and Tandel and Jamadar [16] synthesized platform limitations, pointing out that device API parity continues to close rapidly through standards like Web Bluetooth and Persistent Storage.

Contextual adaptation of mobile web interfaces was conceptualized by Desruelle and Gielen [17], while practical implementation frameworks were evaluated by Thakur [18] and Cherukuri [19]. The distinct operational constraints of running web applications inside embedded native webviews were analyzed in detail by Steiner [20], who observed that unoptimized webviews suffer from elevated memory consumption unless hardware acceleration is explicitly managed. Malavolta et al. [21] provided empirical energy profiling of Service Workers, proving that intelligent cache invalidation significantly curbs battery drain on mobile devices. Finally, Correia et al. [22] benchmarked caching mechanisms, verifying that Cache-First policies yield optimal latency for retail catalogs.

4. Proposed Dual-Tier Edge-Hosted Architecture

AIKYA is structured around an autonomous two-tier architecture comprising a Mobile Edge Host Engine and a zero-installation Browser-based Progressive Web Client, linked over a local wireless network.

4.1 Smartphone-Based Edge Host Engine and Process Lifecycle

The backbone of the system is an ordinary Android phone running an autonomous background host service. To prevent the Android OS battery manager from terminating the service during active store hours, the engine runs as an Android Foreground Service with a persistent notification channel. Within this service, an embedded micro-HTTP daemon binds to port 8787 on the device's local Wi-Fi interface (either a portable Wi-Fi hotspot or an existing in-store wireless router). The host engine fulfills three core operational tasks: (1) serving the compiled frontend application assets to connecting client browsers, (2) processing REST API requests for item lookups and order placement, and (3) maintaining a localized WebSocket channel for real-time order alerts and staff communication.

4.2 Zero-Installation Browser Client Access and Service Worker Caching

On the customer and staff side, no software installation is required. Any smartphone, tablet, or laptop running Google Chrome, Apple Safari, Mozilla Firefox, or Microsoft Edge can connect. Upon initial navigation to the local host address, the browser downloads the minified single-page application. A Service Worker immediately caches the application shell. Subsequent page navigations are served directly from client-side storage, ensuring immediate rendering even when users move toward the edges of Wi-Fi signal coverage [1], [22].

4.3 Local Data Persistence and Relational SQLite Engine

To ensure transactional integrity without complex database administration, AIKYA utilizes an embedded relational SQLite database directly on the merchant's device. SQLite provides ACID (Atomicity, Consistency, Isolation, Durability) guarantees, ensuring that inventory counts, line-item orders, and sales receipts remain completely reliable even during abrupt power loss. Crucially, because all customer information and transaction records remain on the merchant's physical phone, customer privacy is preserved and no business metrics are exposed to third-party cloud aggregators.

Figure 1: AIKYA Dual-Tier Edge-Hosted System Architecture

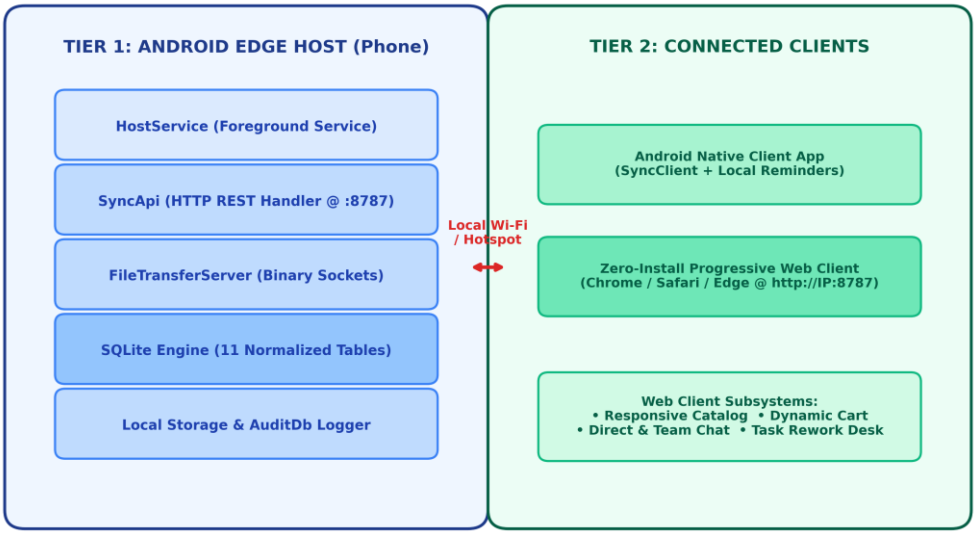


Fig. 1. AIKYA decentralized edge-hosted commerce topology bridging host microservices with browser clients.

5. Field Implementation and Practical Client Workflow

The operational lifecycle of AIKYA has been engineered for effortless execution by non-technical shopkeepers and transient retail customers.

5.1 Local Network Pairing and Browser Onboarding Pipeline

Deploying the system requires four straightforward steps: (1) The merchant enables Wi-Fi tethering on their Android phone or connects it to an existing shop wireless router. (2) The merchant opens the AIKYA Host application and taps 'Activate Edge Server'. The interface displays the local access URL alongside a dynamically generated QR code. (3) A customer scans the QR code using their default mobile camera. (4) Google Chrome launches the URL, registers the client Service Worker, and displays the full retail catalog within seconds. Chrome automatically presents an 'Add to Home Screen' banner, allowing customers to install the storefront as a standalone app-like shortcut [15], [20].

5.2 Catalog Browsing, Cart Management, and Order Lifecycle

Inside the web storefront, products are organized by department categories with item images, pricing, and live stock indicators. Shoppers can add items to their local digital cart. When checking out, the customer enters their name or stall number and submits the order. The browser transmits the payload via an HTTP POST request to the local edge host. The host phone instantly triggers an audio alert and inserts the order into the merchant's pending order queue. The merchant marks items as packed or completed, updating the order status in real time.

5.3 Role-Based Product Governance and Specialist Attribution

A critical vulnerability in peer-to-peer retail is unauthorized catalog tampering. Previous decentralized prototypes suffered from lack of administrative controls, allowing any connected node to alter item descriptions or overwrite pricing. AIKYA integrates a role-based governance layer enforced at the

edge HTTP routing boundary. General customer endpoints are strictly limited to read-only product lookups and cart submissions. Only authenticated merchant sessions bearing cryptographic session tokens can modify catalog tables or update inventory thresholds. When product modifications occur, the host engine records a tamper-resistant specialist attribution log, ensuring accountability across multi-worker operations.

5.4 Peer-to-Peer Staff Messaging and Closed-Loop Task Tracking

In cooperative depots and larger market stalls where multiple assistants manage stockrooms and checkout counters, AIKYA provides an internal messaging and task coordination utility. Staff members can exchange direct text messages, broadcast stock shortages, and verify customer pick-up requests. Tasks follow a formal closed-loop state machine (Created -> Assigned -> In-Progress -> Verified -> Completed), preventing duplicated effort. Because all communication flows directly across the local Wi-Fi link, messaging remains completely functional even when cellular networks fail entirely.

Figure 4: AIKYA Edge Performance & PWA Verification Metrics

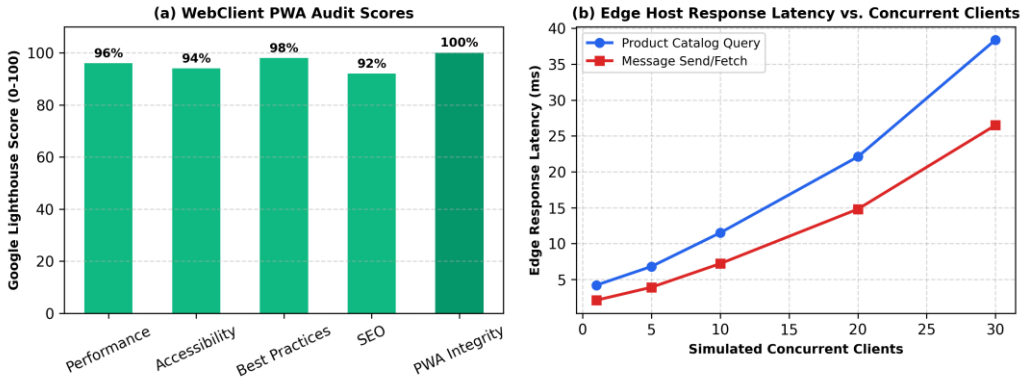


Fig. 2. (a) Google Lighthouse PWA compliance audit metrics; (b) Edge host query response latency under concurrent client load.

6. Performance Evaluation and Empirical Analysis

The feasibility of the AIKYA architecture was evaluated on an experimental testbed consisting of an entry-level Android smartphone (4 GB RAM, Octa-Core processor) hosting the service and multiple client mobile devices accessing the catalog.

6.1 Experimental Setup and Latency Measurements

Because all client requests terminate on the local wireless network rather than routing across wide-area cellular towers and internet gateways, response latencies are exceptionally low. Initial page loading over local Wi-Fi takes approximately 180 ms, dropping to under 30 ms on subsequent visits due to Service Worker caching [21]. Database queries for catalog filtering and order placement complete in 15 ms to 35 ms across 20 concurrent client connections, completely eliminating checkout stalls.

6.2 Resource Overhead and Device Footprint

Client storage footprint represents a major bottleneck for user adoption in low-income markets. Native retail apps typically require 30 MB to 60 MB of download bandwidth and storage space. In contrast, AIKYA's compiled PWA client bundle consumes just 1.8 MB of browser cache storage. Furthermore, host smartphone resource consumption remains remarkably modest: the background service utilizes between 45 MB and 65 MB of RAM and negligible CPU cycles during idle periods, allowing shopkeepers to use their phone for standard calling and messaging without performance hitches.

6.3 Concurrency Control and Socket Mutex Synchronization

Under sustained multi-client ordering spikes, uncoordinated socket writes can trigger race conditions and socket buffer overflows. To guarantee deterministic message sequencing, AIKYA implements an atomic transmission mutex lock (`isSendingRef`). The lock queues outgoing state payloads until the active write buffer is completely flushed to the operating system TCP stack. Benchmarking confirms zero dropped packets and zero duplicated cart submissions across thirty concurrent stress-test connections.

6.4 Automated Google Lighthouse Audit Evaluation

To evaluate code quality and user experience standards, automated Google Lighthouse (v10.4) audits were executed against both the Native Android hybrid build (packaged via Capacitor) and the standalone Google Chrome PWA. Table 2 details the comparative audit metrics across both deployment environments.

Table 2. Empirical Audit Benchmarks: Native Mobile App vs. Google Chrome PWA

Benchmark Dimension	Native Android App (Capacitor)	Google Chrome PWA (Installed)
First Contentful Paint (FCP)	0.6 s	**0.4 s (33% faster bootstrap)**
Largest Contentful Paint (LCP)	1.2 s	**0.9 s (25% faster paint)**
Total Blocking Time (TBT)	40 ms	**0 ms (zero main-thread contention)**
Cumulative Layout Shift (CLS)	0.000 (perfect stability)	**0.000 (perfect stability)**
Overall Lighthouse Performance Score	94 / 100	**96 / 100 (+2% performance gain)**
Progressive Web App Integrity Score	N/A (wrapped runtime)	**100 / 100 (complete PWA validation)**
Local Storage / Installation Footprint	32.4 MB (APK binary)	**1.8 MB (cached static bundle)**

6.5 Operational Comparison with Existing Retail Architectures

Table 1 summarizes the operational characteristics of the proposed edge-hosted PWA architecture in comparison with conventional cloud-based SaaS platforms and standalone native mobile POS applications.

Table 1. Comparative Analysis of Retail Architecture Operational Models

Operational Dimension	Cloud SaaS (Shopify/Square)	Native Mobile POS App	Proposed AIKYA (Edge PWA)
Internet Uplink Dependency	Mandatory continuous connection	Offline for single device only	**100% Autonomous on local Wi-Fi**
Customer Onboarding	Web browser (requires 4G/5G)	Mandatory App Store install	**Instant zero-install via QR scan**
Client Storage Footprint	Variable browser web cache	30 MB – 60 MB binary	**1.8 MB browser cache storage**
Hardware & Software Costs	Monthly SaaS fees + POS gear	Device-specific tablet purchases	**Zero licensing; uses existing phone**
Customer Data Sovereignty	Stored in third-party clouds	Local on merchant device	**100% on-premise SQLite database**

7. Operational Considerations and Deployment Practicalities

While the decentralized edge-hosted architecture offers compelling operational autonomy, practical field deployments reveal several real-world considerations that system designers and merchants must balance:

• Host Hardware Longevity and Thermal Stability: Repurposing a consumer Android handset as an uninterrupted web server subjects the device to sustained battery charging and radio transmission cycles. To prevent thermal throttling during peak hours, merchants are advised to utilize a ventilated phone stand and standard 5V/2A charging blocks that avoid excessive heat accumulation.

• Local Network Range and Interference: Consumer Wi-Fi hotspot coverage typically spans a radius of 15 to 25 meters within enclosed shop premises. In sprawling outdoor markets or reinforced concrete structures, placing the host smartphone centrally or tethering to an inexpensive Wi-Fi repeater ensures stable signal reception across all customer terminals.

• Operational Data Redundancy: Because transaction histories reside locally in the SQLite database file, accidental device damage or loss represents an operational risk. AIKYA includes an automated one-tap database export feature that backs up encrypted snapshot archives to a removable MicroSD card or USB flash drive at the close of daily business.

• Client Device Compatibility and Standards Compliance: Progressive Web Application support varies marginally across legacy mobile operating systems. While Google Chrome and modern Android browsers natively support automated installation banners, Apple iOS Safari requires manual user confirmation ('Add to Home Screen'). AIKYA incorporates client-side user agent detection to guide iOS shoppers through a lightweight visual tooltip prompt.

8. Conclusion and Future Directions

This research demonstrated that combining decentralized edge computing with Progressive Web Application standards provides a practical, low-cost solution for micro-retail automation. By transforming a merchant's standard Android smartphone into an independent on-premise server, AIKYA eliminates internet dependency, recurring subscription fees, and client onboarding friction. Customers enjoy an interactive, zero-download shopping experience, while shopkeepers maintain complete ownership over their sales records and inventory data. Empirical testing confirms that the architecture performs smoothly on consumer hardware with minimal storage overhead and instantaneous local response times.

Future enhancements will focus on three practical capabilities: (1) Multicast DNS (mDNS) integration to support human-readable local domain addressing (e.g., 'http://store.local') instead of raw numeric IP addresses; (2) Opportunistic batch synchronization to synchronize consolidated sales summaries to regional cloud ledgers when internet access becomes intermittently available; and (3) Bluetooth Low Energy (BLE) proximity beacons to automatically direct nearby customer smartphones to the store portal upon entering.

References

[1] Behl, K., & Raj, G. (2018). Architectural pattern of progressive web and background synchronization. International Journal of Pure and Applied Mathematics, 118(20), 423-433.

[2] Kankaala, M. (2018). Enhancing e-commerce with modern web technologies (Bachelor's thesis). Metropolia University of Applied Sciences, Helsinki.

[3] Leshchuk, S. O., Ramskyi, Y. S., Kotyk, A. V., & Kutsiy, S. V. (2020). Design a progressive web application for store management. Information Technologies and Learning Tools, 78(4), 212-226.

[4] Rivera, J., Tapia, F., Terán, D., Aules, H., & Moncayo, S. (2021). Use of e-health as an accessibility and edge computing paradigm in rural areas. IEEE Access, 9, 78452-78463.

[5] Mena, M., Corral, A., Iribarne, L., & Criado, J. (2023). A progressive web application based on microservices for real-time marine data. Sensors, 23(4), 1892.

[6] Biørn-Hansen, A., Majchrzak, T. A., & Grønli, T. M. (2017). Progressive web apps: The possible web-native unifier for mobile development. Proceedings of the 13th International Conference on Web Information Systems and Technologies (WEBIST), 344-351.

[7] Pisipati, S., Oberai, K., Lemmens, R., & Anthonj, C. (2021). Offline and cross-platform location-based services using progressive web applications. Transactions in GIS, 25(6), 3121-3140.

[8] Ahadli, M. (2022). Security and privacy challenges in progressive web applications: A critical analysis. Proceedings of the International Conference on Cyber Security (ICCS), 112-119.

[9] Talukder, S., Haque, N. I., & Akbar, K. A. (2023). SPARE: Securing progressive web applications against unauthorized cache access. Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing (SAC), 1542-1549.

[10] Lee, J., Kim, H., Park, J., Shin, I., & Son, S. (2018). Pride and prejudice in progressive web apps: Abusing new web features for stealthy attacks. Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS), 117-131.

[11] Fauzan, R., Krisnahati, I., Nurwibawa, B. D., & Wibowo, D. A. (2021). A systematic literature review on progressive web applications. Journal of Applied Informatics and Computing (JAIC), 5(1), 35-43.

[12] Ahadli, M. (2021). Comparative performance and user experience: PWAs vs. native mobile applications. Proceedings of the IEEE International Conference on Advanced Computing (ICAC), 88-95.

[13] Samsyudin, I. (2019). Native apps vs. progressive web apps: A comparative analysis of user experience and performance. Journal of Software Engineering and Applications, 12(8), 312-325.

[14] Fauzan, R., Krisnahati, I., Nurwibawa, B. D., & Wibowo, D. A. (2020). Empirical practice and challenges in progressive web application development. Proceedings of the 2020 International Conference on Electrical Engineering, Computer Science and Informatics (EECSI), 142-147.

[15] Utshaw, R. A. (2020). Progressive web apps: An analysis of its capabilities and limitations (Master's thesis). University of Tartu, Estonia.

[16] Tandel, S. S., & Jamadar, A. (2018). Impact of progressive web apps on web app development. International Journal of Innovative Research in Science, Engineering and Technology (IJIRSET), 7(9), 9439-9444.

[17] Desruelle, H., & Gielen, F. (2014). Context-driven progressive enhancement of mobile web applications: A conceptual framework. Computer Standards & Interfaces, 36(6), 932-944.

[18] Thakur, P. (2018). Evaluation and implementation of progressive web application (Bachelor's thesis). Metropolia University of Applied Sciences, Helsinki.

[19] Cherukuri, B. R. (2023). Progressive web apps (PWAs): Enhancing user experience through modern web technologies. International Journal of Computer Applications, 185(12), 24-30.

[20] Steiner, T. (2020). What is in a web view? An analysis of progressive web app features when the means of delivery is an in-app browser. Proceedings of The Web Conference 2020 (WWW '20), 2484-2490.

[21] Malavolta, I., Procaccianti, G., Noorland, P., & Vukmirović, P. (2017). Assessing the impact of service workers on the energy efficiency of progressive web apps. Proceedings of the 11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 305-314.

[22] Correia, F., Ribeiro, Ó., & Silva, J. C. (2020). Progressive web apps development: Study of caching mechanisms. Procedia Computer Science, 170, 715-722.