**Indian Journal of Computer Science and Technology**
https://www.doi.org/10.59256/indjcst.20260503137
Volume 5, Issue3 (September-December 2026), PP: 1133-1140.
www.indjcst.com
**ISSN No: 2583-5300**

# Queueless: A Full-Stack Virtual Queue Management and Waiting-Time Optimization System

**Chetan Kumar G S[1], Prasanna Kumar Naik R[2], Sujal Soans[3], G S Guruswamy[4], Soubhagya R K[5] , Sandhya V[6]**

*[1,2]Assitant Professor, Department of Master of Computer Applications, University BDT College of Engineering (A Constituent College of VTU, Belagavi), Davangere, Karnataka, India.*
*[3, 4, 5, 6] Student, Department of Master of Computer Applications, University BDT College of Engineering (A Constituent College of VTU, Belagavi), Davangere, Karnataka, India.*

***To Cite this Article:*** *Chetan Kumar G S[1], Prasanna Kumar Naik R[2], Sujal Soans[3], G S Guruswamy[4], Soubhagya R K[5], Sandhya V[6], "Queueless: A Full-Stack Virtual Queue Management and Waiting-Time Optimization System", Indian Journal of Computer Science and Technology, Volume 05, Issue 03 (September-December 2026), PP: 1133-1140.*

Copyright: ©2026 This is an open access journal, and articles are distributed under the terms of the Creative Commons Attribution License; Which Permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

**Abstract:** *Physical waiting lines remain the default way of managing walk-in visitors across hospitals, banks, public offices, and retail service centers. When customer arrivals surge or service times fluctuate, conventional lines quickly cause crowded lobbies, frustration over unpredictable waits, and unnecessary manual coordination for counter staff. Traditional queue handling relies heavily on on-site presence, paper slips, and verbal announcements, leaving customers with little visibility into how long they must wait.This paper presents* ***QueueLess****, a full-stack web-based virtual queue management system that enables customers to discover an organization, choose a branch and service, claim a digital token, and monitor queue progress remotely from any browser. The documented platform supports distinct operational roles for customers, counter staff, organization administrators, and system administrators. Its architecture pairs React, Vite, and Redux Toolkit on the frontend with Node.js, Express.js, MongoDB, and Mongoose on the backend, secured through JSON Web Tokens (JWT), HTTP-only cookies, and bcryptjs password hashing. In addition to detailing the hierarchical data model, six-layer system architecture, token lifecycle, and baseline waiting-time estimation equations, this work defines a reproducible evaluation framework covering estimation error, update latency, concurrency stability, and role-based access testing. Because the public repository README describes the intended feature set while the accompanying project-status document still records the core queue engine as pending, no unverified numerical performance claims are reported here; all evaluation metrics are framed for controlled validation once the live build is* verified.

**Key Word:** *Virtual Queue Management, Digital Token Tracking, Waiting-Time Estimation, Role-Based Access Control (RBAC), Full-Stack Web Application, React, Node.js, MongoDB, Express.js, Mongoose.*

## I. INTRODUCTION

Waiting is an unavoidable part of everyday service delivery, whether someone is visiting a medical clinic, a bank branch, or a municipal counter. In a standard physical line, a visitor has to remain on-site simply to protect their place in order. Whenever arrival rates climb or individual transactions run longer than expected, waiting rooms fill rapidly and customers are left with almost no reliable sense of how soon they will be served.

Most legacy service centers rely on handwritten registers, printed paper slips, or verbal callouts to keep lines moving. Although straightforward to set up, manual systems break down easily during peak hours. Customers cannot check queue length before arriving, counter personnel lose time managing disruptions or answering repeated status inquiries, and branch managers rarely have structured records to compare service efficiency across locations.

Virtual queuing addresses these pain points by separating a person's place in the service order from their physical presence inside the waiting room [1], [2]. By moving token issuance and status tracking onto the web, organizations can keep lobbies less crowded while giving staff an organized digital workspace to process requests sequentially.

The project titled **"QueueLess: A Full-Stack Virtual Queue Management and Waiting-Time Optimization System"** introduces a browser-based application tailored for organizations operating one or more branches and services. A customer selects an organization, branch, and active service to join a queue remotely, receiving a unique digital token along with live queue position and an estimated waiting time. On the operational side, counter staff use a dedicated dashboard to call the next token, mark services complete, record no-shows, or temporarily pause a queue, while administrators oversee branches, service parameters, and staff accounts [3].

Built with React, Vite, and Redux Toolkit on the client and Node.js, Express.js, MongoDB, and Mongoose on the server, QueueLess combines remote queue participation with structured multi-role operations and persistent record keeping. Rather than proposing a new stochastic queuing theorem, the project focuses on practical full-stack application engineering—

uniting customer self-service, staff workflow control, role-based security, and a transparent mathematical framework for evaluating waiting-time estimates.

## II. LITERATURE REVIEW

Research on queue management spans virtual ticketing systems, mobile queue sensing, customer wait psychology, discrete-event simulation, and web access-control models. Together, these areas provide the technical and operational grounding behind QueueLess.

### 2.1 Review of Existing Studies

Ngugi, Mwai, and Njomo (2018) examined replacing physical waiting lines with online tickets and virtual queue administration, establishing a direct conceptual precedent for remote queue participation [1]. QueueLess extends that foundation by adding an explicit organization, branch, service, and multi-role management structure. Taking a sensing-based perspective, the QueueSense study (2016) used collaborative smartphone sensing to detect physical queues and estimate waiting durations automatically [2]. While mobile sensing is effective for passive detection in open spaces, QueueLess focuses on active token management and staff-controlled state updates inside structured service providers.

Customer perception plays an equally vital role in whether virtual queues succeed. Yu, Zhang, and Zhou (2022) conducted a large-scale field experiment showing how visible delay estimates influence customer abandonment behavior, underscoring both the value of displaying wait times and the need to verify estimate accuracy empirically [3]. Similarly, Jo, Chi, and Kim (2025) compared physical and virtual queues in restaurant settings, highlighting perceived fairness and waiting cost as primary drivers of customer satisfaction [5]. On the capacity-planning side, Gulhane (2020) applied discrete-event simulation to study hospital queue efficiency and server utilization, offering a useful experimental model for testing multi-counter scenarios [4]. More recently, Awale et al. (2026) synthesized literature across digital queue management, stressing that wait estimates shift under live queue conditions and must be assessed alongside update reliability and staff workload [10].

Because QueueLess serves customers, counter staff, and administrators inside one shared platform, strong authorization and credential protection are essential. Ferraiolo and Kuhn (1992) formalized Role-Based Access Control (RBAC) by binding permissions to functional roles instead of ad hoc user checks [9]. Ferraiolo, Barkley, and Kuhn (1999) extended this model with a web-oriented reference architecture [6], and Sandhu, Ferraiolo, and Kuhn (2000) consolidated it into the NIST RBAC standard [7]. Finally, Provos and Mazières (1999) introduced bcrypt as an adaptive password-hashing scheme with tunable computational cost [8], directly motivating QueueLess's documented choice of bcryptjs for password security.

### 2.2 Comparison of Existing Approaches

The studies reviewed above address distinct layers of digital queuing, behavioral transparency, capacity modeling, and web security:

| Year | Authors | Technology | Main Focus |
|---|---|---|---|
| 1992 | D. F. Ferraiolo & D. R. Kuhn [9] | RBAC Model | Role-based permission formalization |
| 1999 | D. F. Ferraiolo, J. Barkley & D. R. Kuhn [6] | Web RBAC | Role-based access control for web applications |
| 1999 | N. Provos & D. Mazières [8] | bcrypt Hashing | Adaptive, computationally expensive password hashing |
| 2000 | R. Sandhu, D. Ferraiolo & D. R. Kuhn [7] | NIST RBAC | Unified standard for role-based authorization |
| 2016 | QueueSense Study [2] | Mobile Sensing | Smartphone-based physical queue detection |
| 2018 | N. M. Ngugi, J. K. Mwai & M. Njomo [1] | Virtual Queueing | Online ticket generation replacing physical lines |
| 2020 | K. Gulhane [4] | Discrete-Event Sim. | Hospital queue efficiency and counter capacity |
| 2022 | Q. Yu, Y. Zhang & Y.-P. Zhou [3] | Field Experiment | Impact of delay information on queue abandonment |

| Year | Authors | Technology | Main Focus |
|---|---|---|---|
| 2025 | E. H. Jo, O. H. Chi & J. Kim [5] | Behavioral Analysis | Waiting fairness in physical vs. virtual queues |
| 2026 | A. S. Awale et al. [10] | Digital Queue Review | Wait estimation accuracy, reliability, and workload |

### 2.3 Research Gap

Although prior studies establish strong building blocks for virtual ticketing, queue sensing, delay transparency, simulation, and RBAC, these components are frequently explored in isolation. Many local service organizations still lack a unified browser platform that connects remote customer booking with multi-branch administration and live counter operations.

Furthermore, delay estimates in operational software are often assumed to be accurate without defining transparent baseline equations or formal regression error metrics. Multi-role isolation is also overlooked in lightweight token apps, creating operational risks when staff controls and administrative privileges are not strictly separated.

Finally, QueueLess maintains a clear scope boundary: it is engineered as an integrated full-stack queue management system rather than an AI forecasting product, as machine-learning-based predictive waiting times are explicitly documented in the repository as a future enhancement [11].

### 2.4 Contribution of the Proposed System

QueueLess closes these practical gaps by uniting customer queue discovery, digital token issuance, live staff queue controls, and multi-branch administration inside a single web application. It enforces an explicit Organization → Branch → Service → Queue → QueueToken data structure backed by MongoDB and protected through JWT HTTP-only cookies, bcryptjs, and RBAC.

In addition to system integration, this paper contributes a transparent waiting-time mathematical baseline alongside a structured, reproducible evaluation protocol so that system accuracy, concurrency behavior, and security can be rigorously audited once live deployment testing is completed.

## III. EXISTING SYSTEM

Conventional service centers typically manage incoming traffic through physical standing lines, paper ticket dispensers, or handwritten registers at a reception desk. Under this baseline model, a customer travels to the branch, obtains a paper slip or waits in a designated area, listens for verbal announcements or checks a lobby screen, and proceeds to a counter once their number is called.

While simple to operate at low volume, manual queue setups do not scale smoothly when foot traffic increases or when an organization operates across multiple branches. Even basic digital display systems often remain disconnected from remote joining, customer status visibility, and persistent cross-branch reporting.

**The major shortcomings of the existing system are:**

- **Mandatory Physical Presence:** Customers must arrive early and remain inside crowded waiting areas simply to hold their place in line.
- **Poor Waiting-Time Visibility:** Visitors rarely know how many active tokens stand ahead of them or how long service will take, increasing frustration and abandonment.
- **Manual Staff Overhead:** Counter staff spend time calling numbers verbally, reconciling skipped visitors, and logging entries by hand.
- **Weak Multi-Branch Administration:** Management lacks a centralized system to configure services, monitor branch load, or compare historical service durations.
- **Limited Access Accountability:** Paper registers and basic shared logins lack strict role isolation between customers, counter personnel, and administrators. These limitations highlight the clear need for an integrated, browser-accessible virtual queue platform capable of remote joining, transparent progress tracking, and role-governed service operations.

## IV. PROPOSED SYSTEM

The proposed system, **QueueLess**, is a full-stack web application designed to modernize how organizations manage multi-branch service queues. Rather than merely replacing paper tickets with a screen, QueueLess links every customer's digital token to server-validated staff transitions and a persistent database hierarchy.

The client application is developed using **React**, **Vite**, and **Redux Toolkit** to deliver responsive interfaces and predictable client-side state management. The backend uses **Node.js** and **Express.js** to expose RESTful API routes organized cleanly into controllers, queue-engine logic, and **Mongoose** models connected to **MongoDB** [11], [12]. Authentication relies on JSON Web Tokens (JWT) stored inside HTTP-only cookies alongside bcryptjs password hashing, while Role-Based Access Control separates customer, counter staff, organization admin, and system admin actions.

**Main features of the proposed system:**

- **Remote Queue Discovery and Joining:** Customers browse organizations, branches, and active services directly from a web browser to join queues remotely.
- **Digital Token Tracking:** Every service request generates a unique QueueToken storing token number, status, timestamps, queue position, and estimated waiting time.
- **Transparent Wait-Time Estimation:** Computes baseline waiting estimates from active queue position, service average duration, and available counters.
- **Dedicated Staff Dashboard:** Equips counter personnel to call the next token, complete active services, flag no-shows, and pause or resume queues.
- **Multi-Branch Administration:** Allows administrators to configure organizations, physical branches, branch services, and staff accounts from one portal.
- **Role-Based Security Architecture:** Protects restricted routes using JWT HTTP-only cookies, bcryptjs password hashing, and server-side RBAC checks.
- **Persistent Service History:** Archives completed token records and service durations in MongoDB (QueueHistory) for auditing and future analytics.

By combining customer self-service, structured staff workflows, and centralized branch management, QueueLess aims to reduce physical congestion and make waiting more transparent for both visitors and service providers.

## V. METHODOLOGY

The engineering methodology for QueueLess follows a structured lifecycle from domain modeling and security enforcement through token workflows, waiting-time math, and controlled testing.

### A. Domain Hierarchy Modeling

Multi-location service delivery is represented through a five-level operational hierarchy: Organization → Branch → Service → Queue → QueueToken. Structuring entities in this sequence isolates branch operations cleanly while allowing centralized administrative reporting across an entire organization.

### B. Database Schema Design

The backend defines eight MongoDB collections managed via Mongoose ODM: User, Organization, Branch, Service, Queue, QueueToken, Notification, and QueueHistory [4], [12]. Each Service record stores an averageServiceTime attribute, while each QueueToken captures tokenNumber, userId, queueId, status, joinedAt, calledAt, completedAt, position, and estimatedWait. Completed tokens transition into QueueHistory to preserve observed service durations [4].

### C. Authentication and Access Control

User credentials are salted and hashed with bcryptjs before database storage [8]. Authenticated sessions issue signed JWTs stored in HTTP-only cookies to reduce exposure to client-side script attacks. Before executing any sensitive controller action, RBAC middleware verifies that the caller's assigned role permits the requested operation [6], [7], [9].

### D. Token Lifecycle Management

When a customer submits a validated queue-join request, the server creates a new token in the waiting state and computes its initial position. As counter staff operate their dashboard, server-enforced rules govern transitions across waiting, called, completed, skipped/no-show, and paused states, ensuring unauthorized clients cannot alter another user's token.

### E. Waiting-Time Estimation Model

Using the schema's averageServiceTime ($\mu_s$) and estimatedWait ($\widehat{W}$) fields, the transparent baseline estimate for a single stream with $n$ customers ahead—and its approximation across $c$ equivalent active counters—is given by:

$$\widehat{W} = n \times \mu_s, \qquad \widehat{W} \approx \frac{n \times \mu_s}{c}$$

Actual waiting time ($A_i$) is measured consistently from joinedAt to calledAt. These equations define the analytical evaluation baseline and must be verified against the finalized queue-engine source code once completed.

### F. Evaluation Metrics

Across $m$ evaluated tokens with observed waits $A_i$ and predicted waits $P_i$, continuous estimation accuracy is measured using Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and tolerance-based accuracy within a pre-chosen window $\tau$ (such as $\pm 5$ minutes):

$$\text{MAE} = \frac{1}{m}\sum_{i=1}^{m} |A_i - P_i|, \qquad \text{RMSE} = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(A_i - P_i)^2}$$

$$\text{Accuracy}_{\tau} = \left(\frac{\text{count}(i: |A_i - P_i| \le \tau)}{m}\right) \times 100\%$$

Binary classification metrics—Precision, Recall, and F1-score—do not apply to continuous waiting-time estimation and should only be reported if a labeled classifier (such as automated no-show prediction) is implemented and tested; manually flagging a no-show on the staff dashboard is not a predictive classifier. Operational reliability is additionally evaluated through API response time, queue-update latency, token-creation success rate, throughput, and RBAC test pass rate.

### G. Build and Deployment Tooling

The repository structure includes Docker configuration and GitHub Actions workflows, supporting containerized local execution and automated build checks [11], [12].

## VI. SYSTEM ARCHITECTURE

The architecture of QueueLess is organized into six logical layers, with each layer handling a clear responsibility across user interaction, security enforcement, queue execution, and data persistence [11], [12].

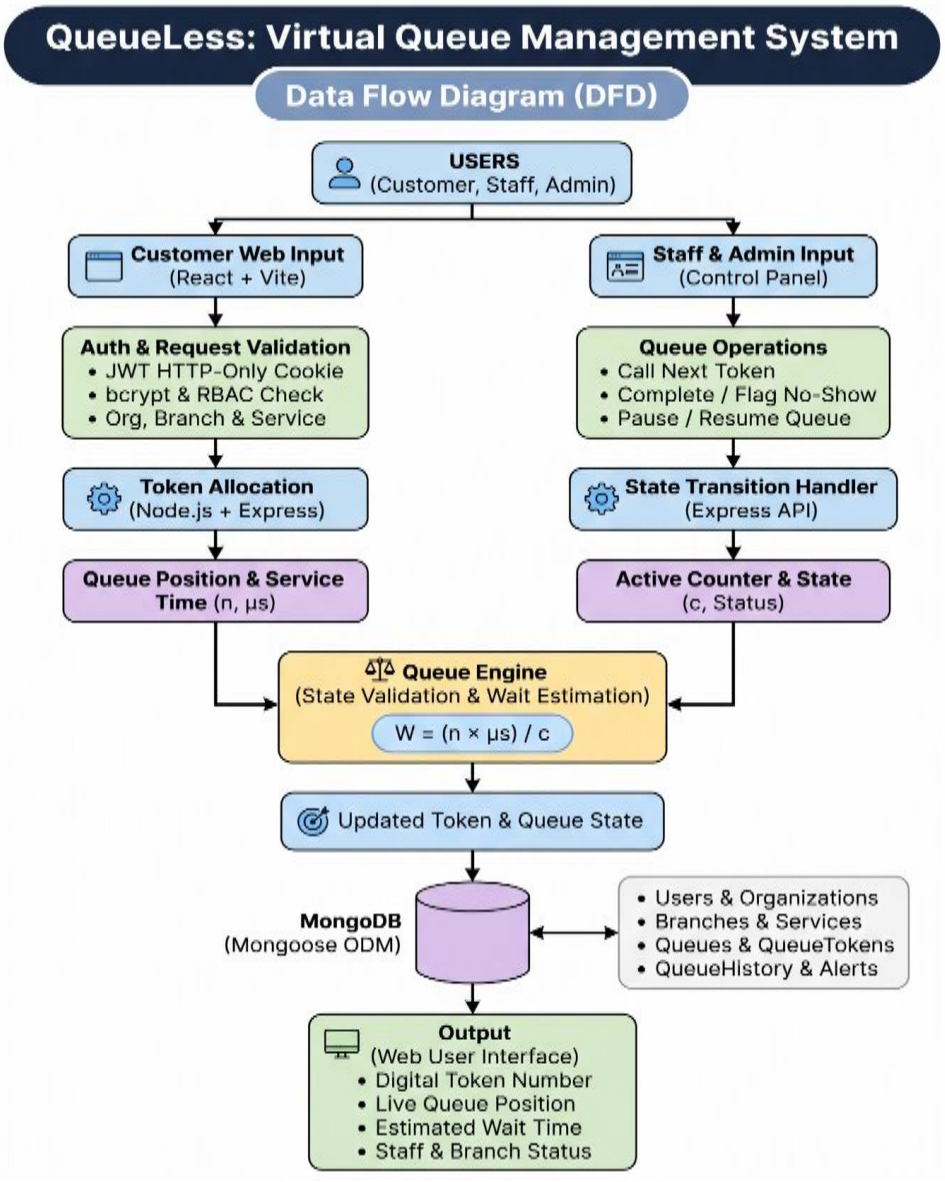



*Fig. 1. System Architecture and Data Flow Diagram (DFD) of QueueLess.*

### A. User Interface Layer

Built with React, Vite, and Redux Toolkit, this layer provides the browser interface through which customers browse services and track tokens, counter staff process live queues, and administrators configure organizations and branches.

### B. API Routing Layer

Implemented using Node.js and Express.js, this layer receives incoming HTTP requests from the web client, validates request parameters, routes calls to dedicated controllers, and formats structured API responses.

### C. Authentication and RBAC Layer

Operating as middleware before core business logic, this layer validates JWT HTTP-only cookie sessions, verifies bcryptjs password hashes during login, and enforces strict role permissions across customers, staff, and administrators.

### D. Business Logic and Queue Engine Layer

This core layer handles sequential token issuance, enforces valid token state transitions (waiting, called, completed, skipped/no-show, paused), recalculates live queue positions, and computes baseline wait-time estimates ($\widehat{W} = (n \times \mu_s)/c$).

### E. Data Access and Database Layer

Mongoose ODM validates schemas and document relationships before persisting records inside MongoDB collections (Users, Organizations, Branches, Services, Queues, QueueTokens, QueueHistory, and Notifications).

### F. Output and Notification Layer

The final layer reflects updated queue states back to the browser interface—displaying digital token numbers, live queue positions, estimated wait times, and status alerts for customers alongside real-time operational controls for staff and administrators.

## VII. EXPERIMENTAL RESULTS

No measured performance dataset or live test log was included with the reviewed repository material. Although the public README outlines the target user-facing capabilities, the accompanying PROJECT_DOCUMENTATION.md labels the build as Phase 1 and marks the core queue engine as pending [11], [12]. Accordingly, no unverified waiting-time reduction percentages or latency scores are claimed here; instead, Table I defines the controlled evaluation suite to execute once the live queue engine is audited, followed by reserved spaces for capturing the running application screens.

| Test Case | Controlled Procedure | Target Metric / Evidence |
|---|---|---|
| **Token Creation** | Submit repeated valid queue-join requests | Success rate (%); unique token IDs |
| **Position Updates** | Enqueue multiple tokens; clear front tokens | Position accuracy after each transition |
| **Wait Estimation** | Log predicted wait vs. joinedAt–calledAt | MAE, RMSE, and $\text{Accuracy}_{\tau}$ |
| **Concurrency** | Dispatch simultaneous joins and staff calls | Zero duplicate tokens; API latency |
| **Role Authorization** | Call permitted & forbidden routes per role | Complete RBAC pass/fail matrix |
| **Staff Controls** | Run call, complete, no-show, pause actions | Valid state transitions; update latency |

*Table I. Controlled Evaluation Test Plan for QueueLess*

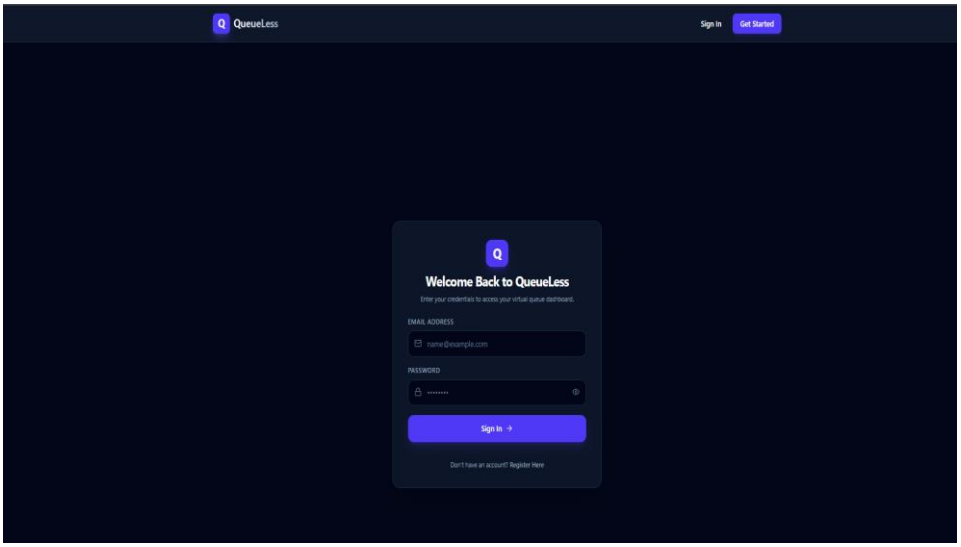



*Fig1: Landing Page*

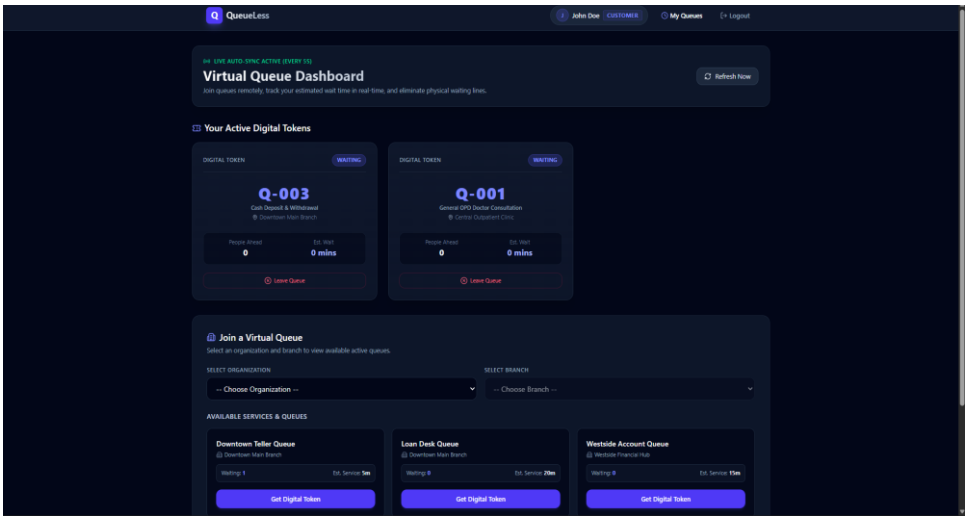



*Fig 2: User Dashboard*

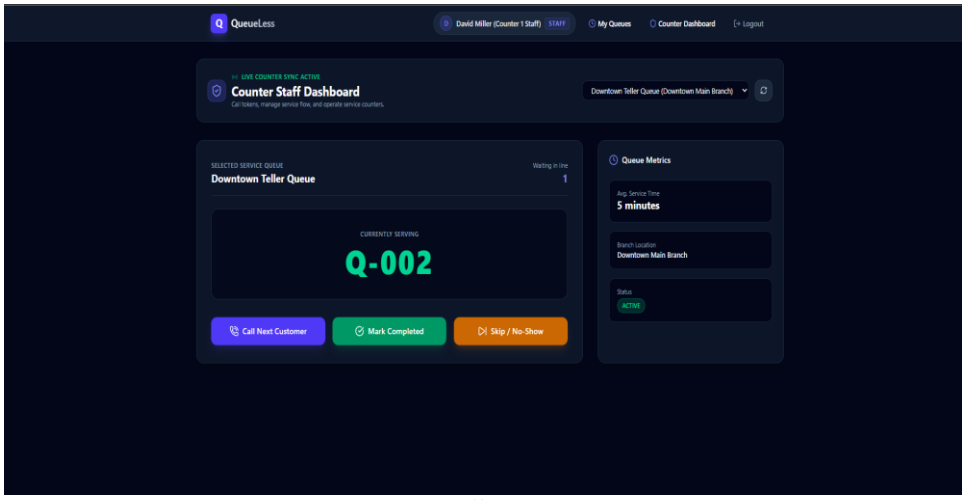



*Fig 3: Staff Dashboard*

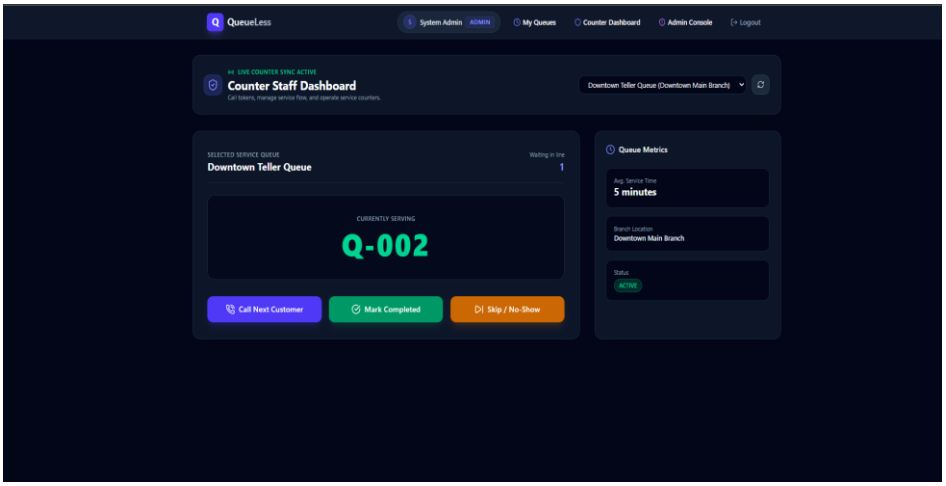



*Fig 4: Admin Dashboard*

## VIII. CONCLUSION

The project **"QueueLess: A Full-Stack Virtual Queue Management and Waiting-Time Optimization System"** presents a structured web-based approach to replacing crowded physical waiting lines with remote queue joining and digital token tracking. By combining React, Vite, and Redux Toolkit on the frontend with Node.js, Express.js, Mongoose, and MongoDB on the backend—secured through JWT HTTP-only cookies, bcryptjs, and role-based access control—the platform unifies customer self-service, counter staff workflows, and multi-branch administration in one coherent system.

Through its hierarchical Organization → Branch → Service → Queue → QueueToken data model, the system is designed to give customers live visibility into their queue position and estimated waiting time while providing staff with clear

controls to call, complete, skip, or pause active queues. Compared to conventional paper-token workflows, this integrated structure reduces on-site uncertainty and preserves persistent service histories for operational review.

Because the reviewed documentation still lists the queue engine as pending, validating the running implementation and executing the six-part experimental test matrix remains the essential next milestone. Quantitative metrics such as MAE, RMSE, throughput, and concurrency latency should be populated strictly after controlled trials are completed and verified.

While the baseline average-time formula provides a transparent starting point, real-world waiting durations remain sensitive to demand spikes, irregular transaction lengths, no-shows, and counter breaks. Future refinements outlined in the project roadmap include event-driven real-time push updates, SMS and WhatsApp notifications, QR-based walk-in check-in, native mobile clients, calendar integration, and machine-learning-based waiting-time prediction trained on accumulated QueueHistory records [11].

## REFERENCES


1. N. M. Ngugi, J. K. Mwai, and M. Njomo, "Virtual Queue Management System," International Journal of Social Sciences and Information Technology, 2018. Available: https://www.ijssit.com/main/wp-content/uploads/2018/08/Virtual-Queue-Management-System.pdf
2. "Collaborative Recognition of Queuing Behavior on Mobile Phones," IEEE Transactions on Mobile Computing, vol. 15, no. 1, pp. 60–73, 2016, doi: 10.1109/TMC.2015.2407400.
3. Q. Yu, Y. Zhang, and Y.-P. Zhou, "Delay Information in Virtual Queues: A Large-Scale Field Experiment on a Major Ride-Sharing Platform," Management Science, vol. 68, no. 8, pp. 5745–5757, 2022, doi: 10.1287/mnsc.2022.4448.
4. K. Gulhane, "Enhancing Queuing Efficiency Using Discrete Event Simulation," International Journal of Logistics Systems and Management, vol. 36, no. 4, pp. 531–546, 2020, doi: 10.1504/IJLSM.2020.108930.
5. E. H. Jo, O. H. Chi, and J. Kim, "How Waiting Decisions Are Made: A Comparison of Restaurant Waiting Equity and Behavioral Intentions in Physical and Virtual Queues," Journal of Hospitality and Tourism Management, pp. 140–152, 2025, doi: 10.1016/j.jhtm.2025.03.009.
6. D. F. Ferraiolo, J. Barkley, and D. R. Kuhn, "A Role-Based Access Control Model and Reference Implementation Within a Corporate Intranet," ACM Transactions on Information and System Security, vol. 2, no. 1, pp. 34–64, 1999, doi: 10.1145/300830.300834.
7. R. Sandhu, D. Ferraiolo, and D. R. Kuhn, "The NIST Model for Role-Based Access Control: Towards a Unified Standard," Proc. Fifth ACM Workshop on Role-Based Access Control, pp. 47–63, 2000, doi: 10.1145/344287.344301.
8. N. Provos and D. Mazières, "A Future-Adaptable Password Scheme," Proc. USENIX Annual Technical Conference, pp. 81–92, 1999.
9. D. F. Ferraiolo and D. R. Kuhn, "Role-Based Access Controls," Proc. 15th National Computer Security Conference, 1992.
10. S. Awale, R. C. Dewarde, A. V. Pawal, P. S. Lohar, and P. K. Bujare, "Digital Queue Management Systems: A Literature Review of Waiting-Time Estimation, Virtual Queues, and Service Operations," Iconic Research and Engineering Journals, vol. 10, no. 4, 2026, doi: 10.64388/IREV10I4-1724331.
11. Sujal16V, "QueueLess: A Full-Stack Virtual Queue Management and Waiting-Time Optimization System," GitHub repository README. Available: https://github.com/Sujal16V/QueueLess_By_Sujal
12. Sujal16V, "QueueLess Project Documentation and Database Documentation," GitHub repository documentation. Available: https://github.com/Sujal16V/QueueLess_By_Sujal/tree/master/docs