RESEARCH ARTICLE OPEN ACCESS

Web-Based Health Appointment Booking System

Dr. K. Vijayalakshmi M.E, Ph.D.*, Gracemaria Y**, Charles Phruno M**,Ganesan S**, Harini S**

*Assistant Professor Department of Computer Science & Engineering, R P Sarathy Institute of Technology, Salem
**UG Students Department of Computer Science & Engineering, R P Sarathy Institute of Technology, Salem
India

ABSTRACT
Hospital appointment management systems often suffer from prolonged patient waiting times, fragmented communication, and limited visibility into live queue status. Conventional web-based solutions typically provide only static booking interfaces, without real-time monitoring or integrated notification workflows. This paper presents a web-based Smart Health Appointment Management System with live queue tracking and automated notifications. The system follows a three-tier architecture comprising a React-based frontend, a Node.js/Express backend, and a MySQL database accessed via Prisma ORM. Real-time queue updates are delivered using WebSocket-based communication, enabling patients and doctors to track consultation progress and waiting positions dynamically. The platform supports role-based access control for patients, doctors, and administrators, covering online appointment booking, rescheduling and cancellation, medical record upload, and operational dashboards. Integrated SMS and voice call reminders, implemented via the Twilio API, provide multilingual patient notifications and last-minute alerts before consultation time. Waiting time estimation is derived from live queue state and configurable service-time rules, improving predictability without requiring complex machine learning models. A functional prototype demonstrates that the proposed architecture enhances transparency, reduces perceived waiting time, and improves coordination between stakeholders compared to manual appointment handling and non–real-time web systems.

Keywords — Hospital Appointment System, Online Scheduling, Real-Time Queue Management, WebSocket, SMS and Voice Notifications, Role-Based Access Control, Node.js, React, MySQL.

I. INTRODUCTION

Hospital outpatient departments often experience prolonged waiting times, limited visibility of queue status, and fragmented communication between patients and staff. In many settings, appointments are still managed through manual registration or basic web forms that provide only static time slots, without reflecting consultation delays or real-time changes in doctor availability. As a result, patients must wait without a clear estimate of when they will be seen, while hospital personnel have little support for coordinating schedules, handling emergencies, or monitoring overall utilization.

Existing web-based appointment portals deployed by individual hospitals or insurance providers mainly focus on slot reservation and simple reminder notifications. They frequently lack integrated support for multilingual interfaces, emergency triage, and real-time queue position tracking once a patient physically arrives at the facility. In addition, most systems are designed from the perspective of a single institution and do not consider interoperability requirements with laboratory information systems, pharmacy systems, or national health records. As a result, hospitals often end up with fragmented tools that solve parts of the problem but still rely on manual coordination at the front desk.

From a research perspective, prior work on appointment scheduling and queue management has explored mathematical optimisation and queuing-theory-based approaches for allocating time slots and minimising waiting times [1], [2]. However, many of these models assume idealised conditions and are not implemented as end-to-end digital solutions that can be deployed in resource-constrained hospitals. At the same time, recent advances in web technologies, real-time communication protocols, and cloud-hosted databases have made it feasible to build scalable, secure, and user-friendly appointment platforms that can integrate with existing workflows. This creates an opportunity to re-visit the appointment

management problem from a pragmatic, implementation-oriented standpoint.

To address these limitations, this paper presents a web-based Smart Health Appointment Booking System with live queue tracking and automated notifications. The system adopts a three-tier architecture comprising a React-based frontend, a Node.js/Express backend, and a MySQL database accessed via Prisma ORM. Real-time queue updates are delivered to connected clients using WebSocket-based communication, enabling patients and doctors to track consultation progress and queue position dynamically. Role-based access control provides tailored views and actions for patients, doctors, and administrators, including appointment booking, schedule management, and operational monitoring.

The system presented in this paper, Smart Health Appointment Booking with Live Queue Tracking, is positioned at this intersection of practical needs and technological advances. It targets small and medium-sized hospitals that may not have dedicated IT teams, while still following modern architectural principles such as separation of concerns, layered design, and modular service boundaries. By combining a responsive web frontend, RESTful APIs, a relational database, and WebSocket-based real-time capabilities, the system demonstrates how a hospital can progressively migrate from manual queues to a digital-first, patient-centric process without disrupting existing clinical routines.

The main contributions of this work are: 1) the design and implementation of a full-stack appointment management platform that combines real-time queue tracking with multi-channel notifications; 2) the integration of a role-based, multilingual user interface that supports patients, doctors, and administrators within a unified system; and 3) a demonstration that lightweight rule-based waiting-time estimation and reminder mechanisms can improve transparency and coordination compared to traditional and non–real-time appointment workflows.

Finally, the system explicitly addresses the needs of different user roles. Patients require transparency about their position in the queue and estimated waiting time. Doctors need a consolidated view of upcoming and in-progress appointments, along with quick access to patient details. Administrators are tasked with setting up departments, onboarding doctors, monitoring overall utilisation, and generating reports for management. Designing the platform around these roles ensures that each stakeholder receives tailored functionality while still interacting with a unified underlying system.

II. RELATED WORK

Research on hospital appointment scheduling spans both operational research and information systems domains. Classical studies model outpatient departments as queuing systems, applying techniques such as M/M/1 or M/G/1 queues, simulation, and optimisation to minimise patient waiting times and doctor idle times [1], [2]. These works provide valuable theoretical insights into appointment slot allocation and overbooking strategies. However, their assumptions often abstract away practical aspects such as last-minute cancellations, walk-in patients, and varying consultation durations across specialties, which are critical in real-world deployments.

With the widespread adoption of the internet and mobile devices, attention has shifted towards web-based appointment portals. Many hospitals have introduced patient portals that allow self-service registration, browsing of available slots, and online booking [3], [4]. Commercial systems offered by electronic medical record vendors extend this concept by integrating laboratory results, prescription refills, and billing information. While these solutions improve convenience and reduce front-desk workload, they typically treat the appointment process as a static event: once a booking is confirmed, patients receive little information about subsequent queue dynamics, especially on the day of the visit.

To address real-time aspects, several works explore electronic queue management systems deployed in banks, government offices, and service centres [5], [6]. These systems issue tokens, display calling numbers on large screens, and sometimes

provide SMS notifications when a customer's turn is approaching. Although the underlying idea is similar, hospital environments differ significantly due to clinical triage, emergency prioritisation, and the need to coordinate multiple services (consultation, laboratory, radiology, pharmacy) within a single visit. Consequently, generic queue systems cannot be directly applied without customisation.

A smaller body of literature examines integrated hospital appointment and queue systems that combine online booking with on-site token management. Some prototypes use RFID or smart cards to track patient movement, whereas others rely on mobile applications and Bluetooth beacons [7]. These solutions demonstrate promising reductions in perceived waiting time but require specialised hardware and are less suitable for resource-constrained settings where patients may not own smartphones or where Wi-Fi coverage is limited. Moreover, implementation details are often presented at a high level, with limited discussion of security, scalability, and maintainability.

The system proposed in this paper differentiates itself from prior work in several ways. First, it focuses on a web-first, device-agnostic design that works on low-cost smartphones and desktop browsers without requiring native mobile applications. Second, it integrates live queue tracking with multichannel notifications (web, SMS, and optional voice calls), ensuring that patients can stay informed even when offline. Third, it adopts a layered architecture using widely adopted open-source technologies, making it easier for hospitals and academic institutions to reproduce, extend, and integrate the system. Finally, unlike conceptual models that primarily present algorithms, this work emphasises an end-to-end implementation and evaluation, providing concrete insights for practitioners.

III. SYSTEM ARCHITECTURE

The proposed Smart Health Appointment Booking System follows a three-tier web architecture consisting of a presentation layer, an application layer, and a data layer. In addition, it integrates external communication services for SMS and voice notifications. This modular design separates user interaction from business logic and data management, simplifying maintenance and future extension.

A. *Overall Architecture*

At the highest level, the system comprises a React-based single-page application running in the patient's or doctor's browser, a Node.js/Express backend exposing RESTful APIs and WebSocket endpoints, and a MySQL relational database accessed through the Prisma ORM. Clients communicate with the backend over HTTPS for standard CRUD operations, while real-time queue updates are delivered over a persistent WebSocket channel using Socket.io. The backend also interacts with the Twilio cloud platform to send SMS messages and initiate voice calls for appointment reminders.

From an architectural perspective, the system follows a clear separation between client, server, and data persistence. The frontend is implemented as a single-page application that communicates with the backend exclusively via RESTful endpoints and WebSocket channels. This separation allows the UI to evolve independently from the server logic. For example, new views for teleconsultation or vaccination drives can be added without changing the core appointment APIs, as long as the contracts remain stable.

The backend exposes resources such as patients, doctors, departments, appointments, queues, and notifications. Each resource has well-defined endpoints for creation, retrieval, update, and deletion, secured via token-based authentication and role-based access control. By aligning the API design with resource-oriented principles, the system becomes easier to document and integrate with third-party services such as hospital information systems or reporting tools. The use of a relational database further ensures that data consistency is maintained across these resources, enabling cross-cutting queries such as "average waiting time per department" or "doctor utilisation per week".

In addition to the core services, the architecture incorporates supporting components for logging, auditing, and configuration management. Application logs capture events such as login attempts, appointment creations, status transitions, and notification delivery results. These logs are invaluable for troubleshooting, compliance reporting, and future analytics. Configuration parameters—such as clinic operating hours, maximum allowed concurrent appointments, and notification templates—are externalised, allowing administrators to adjust system behaviour without redeploying code.

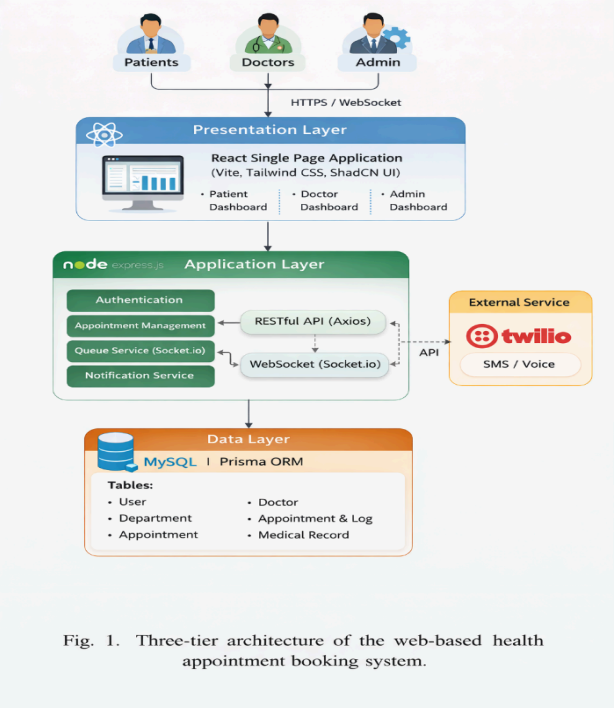

Fig. 1. Three-tier architecture of the web-based health appointment booking system.

Fig. 1 Three-tier architecture of the web-based health appointment booking system

B. *Presentation Layer*

The presentation layer is implemented as a React application bundled with Vite. It provides distinct dashboard views for patients, doctors, and administrators, while sharing common UI components based on Tailwind CSS and ShadCN UI. Navigation and routing are handled on the client side, enabling smooth transitions between booking screens, appointment history, and analytics pages.

Patients can register, log in, browse departments and doctors, book or cancel appointments, view their upcoming visits, and track live queue status. Doctors use their dashboard to view daily appointment lists, mark patients as in consultation or completed, handle cancellations, and access uploaded medical records. Administrators monitor high-level statistics, approve or reject doctor accounts, and manage departments. All views interact with the backend through an Axios-based service that automatically attaches JSON Web Tokens (JWTs) stored in the browser to each request.

The frontend also establishes a Socket.io connection to receive real-time queue updates. When the state of an appointment changes on the doctor side, the backend emits messages that are rendered instantly on the patient's dashboard, avoiding manual refresh.

The presentation layer is deliberately designed to accommodate users with varying levels of digital literacy. Navigation structures follow a consistent pattern across roles: a dashboard summarising key metrics, clearly separated sections for appointments, and contextual actions grouped near related information. Visual indicators such as colour-coded status badges, progress bars for queue position, and icons for emergency cases help users quickly interpret the system state without reading large amounts of text.

Accessibility considerations play a central role in the interface design. Font sizes and contrast comply with common accessibility guidelines to ensure readability in bright clinical environments. Interactive elements, such as buttons and dropdowns, are sized for comfortable use on touchscreens. Error messages are phrased in plain language and are displayed near the fields that require correction, reducing user frustration during registration or booking. For multilingual deployments, the UI text is externalised into language files, allowing easy translation and localisation.

The presentation layer also incorporates responsive design techniques so that the same

codebase can serve desktops at the reception desk, tablets used by doctors during ward rounds, and smartphones used by patients at home. Layouts adapt automatically to screen width by re-flowing columns into vertical sections, hiding non-essential controls, and using collapsible panels for secondary information. This responsiveness reduces development and maintenance effort while ensuring a consistent experience across devices.

C. *Application Layer*

The application layer is built on Node.js with the Express framework. It exposes a set of RESTful endpoints grouped by functional role. The authentication module handles login and token issuance, while middleware components verify JWTs and enforce role-based access control for protected routes. Dedicated route modules manage patient, doctor, and administrator operations respectively.

The patient module provides endpoints for retrieving departments and associated doctors, validating appointment rules, creating and cancelling appointments, and querying queue position. The doctor module allows retrieval of daily schedules, updating appointment statuses, and accessing attached medical records. The administrator module supports user management and high-level system monitoring. File uploads for medical records are processed using streaming middleware, which stores files on the server and records metadata in the database.

A real-time queue service, implemented using Socket.io, runs alongside the REST API within the same backend process. When a doctor updates the status of an appointment, the queue service recomputes the ordering of waiting patients for that doctor and emits an update event to connected patient clients. The same service also calculates rule-based waiting-time estimates using the number of patients ahead and a configurable average consultation duration.

Within the application layer, appointment management is implemented as a state machine with clearly defined transitions, such as requested ,confirmed , checked-in , in-consultation ,completed ,cancelled , and no-show . Each transition is triggered either by a user action (for example, the patient cancelling a visit) or by system events (such as the scheduled appointment time passing without check-in). Encapsulating this logic prevents inconsistent states, ensures that business rules are enforced uniformly, and simplifies auditing of appointment lifecycle histories.

The queue management subsystem builds upon the appointment state machine. When a patient checks in at the hospital, the system assigns a queue token based on department, doctor, and appointment time. Tokens are ordered using a combination of scheduled time and priority flags (e.g., emergency cases or elderly patients). An internal scheduler monitors progress as doctors mark appointments as “in-consultation” or “completed” and recalculates the expected waiting time for all remaining tokens. This computation can be based on simple moving averages of past consultation durations or more advanced estimators depending on deployment requirements.

Notification services are orchestrated through a modular adapter pattern. Regardless of whether a message is delivered via in-app notification, SMS, or voice call, the application layer generates a standardised message payload containing the recipient, template identifier, and context variables. Channel-specific adapters then transform this payload into the appropriate format and handle retries, rate limiting, and error reporting. This abstraction allows hospitals to change notification providers or add new channels (such as email or messaging apps) without significant changes to core business logic.

D. *Data Layer*

The data layer uses MySQL as the primary relational database. Schema management and application-level access are performed with the Prisma ORM. The core entities include User, Doctor, Department, Appointment, AppointmentLog, and MedicalRecord. Users are classified by role (patient, doctor, administrator), while the Doctor entity extends user information with professional details such as department,

degrees, and approval status. Appointments link patients, doctors, and departments; they store the date, timeslot, status, queue number, emergency flag, and reason for visit. Medical records are associated with appointments and contain metadata for uploaded files.

At the persistence level, the schema is normalised to reduce duplication while still enabling efficient queries for the most common operations. Indexes are created on attributes such as appointment date, department identifier, and doctor identifier to accelerate listing operations in busy clinics. Composite indexes support queries that filter by multiple criteria simultaneously, such as "all completed appointments for a given doctor in the last 30 days". Careful index selection ensures that the system remains responsive even when the dataset grows.

Data integrity is enforced through foreign key constraints and application-level validations. For example, an appointment row cannot be created for a non-existent patient or doctor, and deletion of a doctor who has historical appointments is restricted or cascaded according to hospital policy. Soft-delete flags can be used for sensitive entities to preserve historical records while hiding them from day-to-day views. Audit tables capture key changes, such as modifications to department definitions or role assignments.

To support analytics and reporting, the data layer can be extended with aggregated views or materialised reports. These include metrics such as average daily appointment volume, distribution of appointment types (routine vs emergency), no-show rates, and peak hours by department. Export mechanisms allow administrators to generate CSV or spreadsheet files for further analysis in external tools or to share with health authorities.

E. *Security and Privacy Considerations*

Healthcare data is inherently sensitive, and any appointment system must comply with applicable privacy regulations and institutional policies. The proposed architecture incorporates several mechanisms to safeguard confidentiality, integrity, and availability. Communication between the frontend and backend is secured using HTTPS to prevent interception or tampering of in-transit data. Authentication is token-based, and short-lived access tokens can be combined with refresh tokens to balance usability and security.

Role-based access control ensures that each user only sees data relevant to their responsibilities. Patients can view and manage only their own appointments and personal information; doctors are restricted to appointments related to their assigned departments or schedules; administrators have extended privileges, but administrative actions are logged for accountability. Sensitive operations, such as changing a doctor's schedule or cancelling an appointment on behalf of a patient, may require additional confirmation steps or multi-factor authentication.

From a data-privacy standpoint, the system minimises the collection of unnecessary personal information and provides mechanisms to anonymise or pseudonymise data used for analytics. Backup strategies are put in place to protect against data loss, and restore procedures are periodically tested. By embedding these security and privacy measures in the architecture, the system aims to build patient trust while meeting institutional compliance requirements.

F. *Multilingual and Configurable Deployment*

Because the system targets diverse healthcare environments, configurability is as important as core functionality. All textual labels, notification templates, and instructional messages are stored in configuration files or database tables rather than being hard-coded. This design allows hospitals to translate content into local languages, adjust politeness levels, and incorporate culturally appropriate examples without modifying the underlying code.

Deployment-specific settings—such as time-zone, date formats, currency symbols, and working days—are also externalised. For instance, a hospital operating six days a week with extended evening clinics can configure its slot generation rules differently from a facility with strict morning and afternoon sessions. Similarly, emergency handling

policies, maximum queue length thresholds, and notification frequencies can be tuned to align with institutional guidelines. This high degree of configurability makes the proposed system suitable as a generic foundation that organisations can adapt to their own processes.

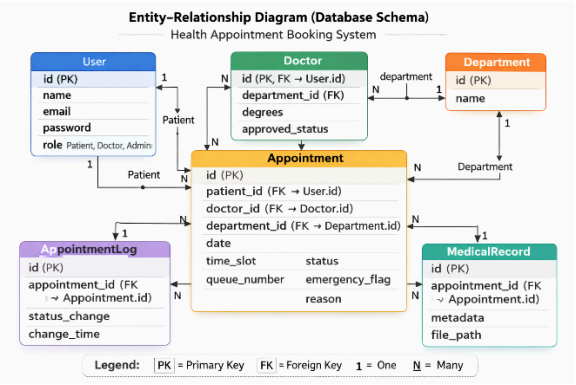

Fig. 2 Entity–relationship diagram of the database schema showing users, doctors, departments, appointments, logs, and medical records

IV. ADVANCED FEATURES

Beyond basic web-based appointment booking, the proposed system integrates several advanced features that improve real-time visibility, communication, and usability. This section describes the live queue tracking mechanism, the notification workflow using SMS and voice calls, the multi-step appointment process with conflict validation, and the role-based dashboards including emergency handling.

A. *Live Queue Tracking and Waiting-Time Estimation*

A key feature of the system is real-time queue monitoring for each doctor. When a patient books an appointment, the backend assigns a queue number based on the selected date, timeslot, and the existing appointments for that doctor. During the consultation day, doctors update the status of each appointment from booked to in consultation, completed , or cancelled using their dashboard. Each status change triggers a queue recomputation on the server.

Live updates are delivered to connected clients using Socket.io. Patient dashboards subscribe to queue events associated with their own appointment and doctor. Whenever a status update occurs, the backend emits a message that includes the ordered list of booked appointments, the current serving position, and basic queue metrics. The patient's browser recalculates the number of patients ahead and refreshes the displayed position without requiring a manual page reload.

Waiting-time estimation is computed using a rules-based approach. The system maintains a configurable average consultation duration, which can be tuned per department or per doctor. For a given appointment, the estimated waiting time is derived by multiplying the number of patients ahead in the queue by the average duration. Although this approach does not rely on machine learning, it provides a practical and easily interpretable estimate that can be adjusted as operational data becomes available.

The live queue tracking feature combines scheduled appointments with real-time events to produce an accurate view of the current state. When a doctor starts a consultation, the system records the timestamp and identifies the corresponding token. Upon completion, the actual consultation duration is stored and used to update average service times for that doctor and department. Over time, these averages become more representative of real-world behaviour, enabling the system to provide more reliable estimates.

In addition to averages, the algorithm may incorporate variability and confidence intervals. For example, the system can compute the 75th percentile of past consultation durations and base waiting-time estimates on this value to reduce the risk of under-prediction. Emergency or high-priority cases can be given virtual "shortcuts" in the queue by inserting their tokens ahead of non-urgent cases while still updating downstream estimates accordingly. The UI communicates such reordering clearly by indicating that an emergency case has been prioritised, thereby maintaining fairness and transparency.

To prevent information overload, the queue interface is tailored for each role. Patients see their current position, expected waiting time, and a concise explanation of what will happen next.

Reception staff see a more technical view, including the full token list, flags for delayed cases, and alerts when service times drift significantly from historical trends. Doctors are presented with a list of upcoming patients plus quick access to key medical and administrative information, allowing them to prepare between consultations.

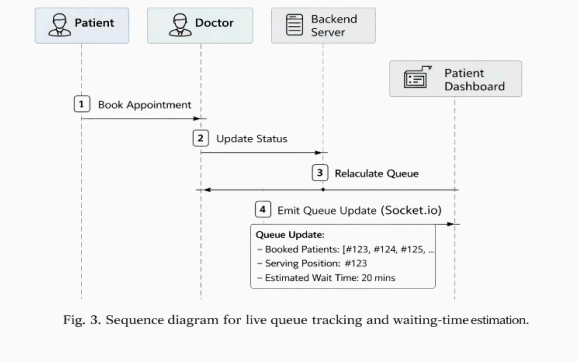

Fig. 3. Sequence diagram for live queue tracking and waiting-time estimation.

Fig. 3 Sequence diagram for live queue tracking and waiting-time estimation

B. ***SMS and Voice Call Notification Workflow***

The system incorporates multi-channel notifications to keep patients informed about their appointments and to reduce no-shows. The notification module uses the Twilio API to send SMS messages and initiate voice calls to the patient's registered phone number.

Immediately after a successful booking, the backend sends a confirmation SMS containing the appointment date, timeslot, and doctor name. In addition, a scheduled notification routine periodically scans upcoming appointments and identifies those for which reminders must be sent. Two reminder windows are configured: one at thirty minutes before the appointment time and another at five minutes before. At the thirty-minute mark, the system can send an SMS reminder and initiate an optional voice call that verbally informs the patient about the approaching consultation. The five-minute reminder is implemented as a voice call to ensure that the message is noticed even if the patient is away from their device.

The notification logic respects appointment status and will not trigger reminders for cancelled or already completed consultations. Any delivery failures reported by Twilio can be logged for later analysis. This workflow integrates seamlessly with the live queue system: as the queue progresses, patients receive both real-time visual feedback and proactive alerts, improving punctuality and reducing idle time for doctors.

The multichannel notification workflow is designed to reach patients through the most appropriate medium depending on connectivity and preference. During registration or profile setup, each patient can specify whether they prefer SMS, in-app notifications, email, or voice calls. These preferences are stored alongside contact details and are consulted whenever a trigger event occurs, such as appointment confirmation, rescheduling, or imminent queue position.

Notification templates are parameterised to ensure consistency and localisation. For instance, an appointment-confirmation template contains placeholders for patient name, date, time, department, and hospital location. At runtime, the notification service injects the relevant values and selects the correct language variant. This approach not only reduces manual effort but also minimises errors that could arise from free-form message composition.

The workflow also handles failure scenarios gracefully. If an SMS gateway returns a delivery failure, the system can automatically attempt an alternative channel, such as in-app notification or voice call, and logs the outcome. For critical alerts, such as emergency schedule changes or unplanned doctor absences, the system may escalate to multiple channels simultaneously. Administrators can review notification statistics, including success rates and average delivery times, to fine-tune provider choices and message strategies.

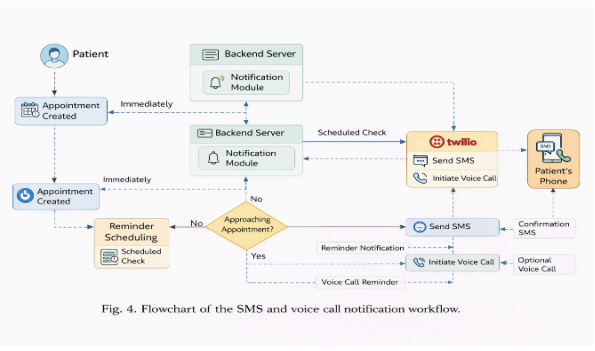

Fig. 4. Flowchart of the SMS and voice call notification workflow.

Fig. 4 Flowchart of the SMS and Voice Call notification workflow

C. *Multi-Step Appointment Workflow and Conflict Validation*

To guide users and reduce errors, the booking process is organized as a multi-step workflow. In the first step, the patient selects a department and a doctor from filtered lists that are dynamically loaded from the backend. In the second step, the patient selects a date and time slot. Available time slots are generated on the frontend but are filtered using availability data from the server. Unavailable or past time slots are visually disabled, preventing accidental selection. In the final step, the patient confirms the appointment details and may optionally provide a free-text reason for the visit and upload supporting medical documents.

Before an appointment is persisted, the backend performs conflict validation. Validation checks include ensuring that the doctor is approved and active, the selected date and timeslot are valid for the doctor's schedule, and the patient does not already hold a conflicting appointment at the same time. If any rule is violated, the server responds with a descriptive message, and the frontend displays the error without creating the appointment. This combination of multi-step guidance and server-side validation reduces booking mistakes and improves data consistency.

D. *Role-Based Dashboards and Emergency Handling*

The system implements role-based dashboards tailored for patients, doctors, and administrators to ensure efficient workflow management. The patient dashboard provides a clear view of upcoming and past appointments, current queue position, and easy access to doctor information and booking features. The doctor dashboard focuses on daily schedules, queue order, and patient status, allowing quick actions such as starting or completing consultations, which automatically update the system in real time. The administrator dashboard offers centralized control over department management, doctor approvals, and operational monitoring. Visual elements such as alerts, badges, and timers help highlight important information and improve usability. Additionally, emergency cases are handled dynamically by prioritizing them in the queue and recording actions through audit logs, ensuring accountability and consistency. Overall, the dashboards present complex data in a simplified manner, supporting better decision-making and enhancing system efficiency.

V. IMPLEMENTATION AND EVALUATION

A. *Technology Stack and Development Environment*

The system is implemented using a modern JavaScript-based stack. The frontend is built with React and TypeScript, bundled with Vite, and styled using Tailwind CSS and ShadCN UI components. Routing and state management are handled within the React application, while Axios is used as the HTTP client for communication with the backend.

The backend is developed in Node.js using the Express framework. Prisma ORM is employed to model the database schema and to generate type-safe queries for a MySQL database. Socket.io is used to manage bidirectional WebSocket communication for live queue updates. File uploads for medical records are handled with streaming middleware that stores files on the server and records metadata in the database. Twilio's official Node.js SDK provides integration with SMS and programmable voice services. The system is deployed and tested in a local environment with Node.js v22 and a MySQL instance running on the same machine.

The choice of technology stack reflects a balance between modern design and operational pragmatism. All selected components are open-source or freely available for academic and non-commercial use, reducing licensing costs and enabling community contributions. Furthermore, the stack is well supported by extensive documentation and active developer communities, which is critical for long-term maintenance. Containerisation technologies such as Docker can optionally be used to package the backend and database services, simplifying deployment across different operating systems and infrastructure environments.

B. *Model Implementation Overview*

The backend is organized into route modules that mirror the system's roles. The authentication module exposes login endpoints and issues JSON Web Tokens upon successful credential verification. The patient module includes endpoints for retrieving departments and doctors, validating appointment rules, creating and cancelling appointments, uploading medical records, and obtaining queue information. The doctor module provides access to daily schedules, appointment status updates, and associated medical records. The administrator module manages user approval and maintains reference data such as department definitions.

The queue management and notification logic is encapsulated in dedicated services. The queue service listens for appointment status changes and recomputes the ordering for each doctor, emitting Socket.io events to subscribed clients. The notification service encapsulates Twilio integration and provides functions for sending confirmation messages and scheduling reminders. Environment variables are used to configure database credentials, Twilio keys, and other deployment parameters.

C. *Test Scenarios*

The system is evaluated using a series of functional test scenarios that mirror realistic clinic operations. In the first scenario, a patient registers, logs in, browses the "Our Doctors" panel, and books an appointment by completing all three steps of the workflow. The system is verified to assign a queue number, persist the appointment in the database, and send an SMS confirmation to the configured phone number.

In the second scenario, a doctor logs in to the doctor dashboard and reviews the list of appointments for the current day. As the doctor marks appointments as in consultation and completed, the patient dashboard is observed on a separate browser session. The queue position and waiting-time estimate are confirmed to update automatically through WebSocket events without requiring a page refresh.

In the third scenario, the notification pipeline is tested by scheduling an appointment within a short time window relative to the current time. The system is checked to ensure that the 30-minute and 5-minute reminders are triggered as configured, and that SMS and voice calls are received on the registered phone. Additional tests verify that reminders are not sent for cancelled appointments and that rebooked appointments lead to updated notifications.

Functional testing followed a scenario-based approach derived from real hospital workflows. For the patient role, test cases included registration with valid and invalid data, password reset, booking appointments under normal and peak-load conditions, rescheduling within policy limits, and attempting to book overlapping slots. For the doctor role, scenarios covered viewing daily and weekly schedules, marking appointments as completed, updating notes, and handling no-show or cancelled visits. Administrative scenarios examined department creation, doctor onboarding, configuration of clinic hours, and generation of reports.

In addition to these nominal scenarios, robustness tests were conducted to evaluate behaviour under adverse conditions. Examples include network interruptions during booking, simultaneous updates to the same appointment from different clients, sudden spikes in appointment creation requests, and partial failures of the notification gateway. The system correctly preserved data integrity in all such tests, either by completing the operation once connectivity was restored or by rolling back and displaying informative error messages. These results indicate that the design is resilient to common real-world issues.

D. *Performance Evaluation*

To assess performance, load-testing tools were used to simulate concurrent users interacting with the system. A baseline configuration representing a medium-sized hospital with several departments and doctors was deployed on commodity hardware. The test harness generated synthetic traffic patterns comprising appointment searches, bookings, queue updates, and notification triggers. Metrics such as

response time, throughput, CPU usage, and database query latency were recorded.

Under a load of several hundred concurrent virtual users, the system maintained median response times well below one second for core operations such as fetching appointment lists and creating new appointments. Even at higher loads, response times remained within acceptable limits for interactive web applications. Profiling identified a small number of expensive database queries related to complex reporting views; these were optimised through additional indexes and query refactoring. Overall, the performance evaluation suggests that the proposed architecture can support busy outpatient clinics without requiring high-end infrastructure.

E. *Usability Evaluation*

Beyond technical performance, usability is crucial for adoption by patients and healthcare staff. A formative usability evaluation was conducted with a sample group of participants representing the three main roles. Patients were asked to perform tasks such as registering, booking an appointment, and checking their queue status on a mobile device. Doctors evaluated the clarity of their schedule view and the ease of updating appointment states, while administrative staff tested configuration and reporting functions.

Participants provided feedback through structured questionnaires and semi-structured interviews. Most reported that the interface was intuitive and that key information, such as appointment time and queue position, was easy to find. Some suggestions emerged, including larger fonts for elderly users, clearer icons for appointment status, and the option to display instructions in multiple local languages simultaneously. These insights were incorporated into subsequent interface iterations, demonstrating the value of iterative, user-centred design in healthcare applications.

F. *Limitations*

Despite its strengths, the current implementation has several limitations. First, while the system supports multilingual UI text, translation quality and cultural nuances still depend on human translators at each deployment site. Automated translation services could assist but may not be suitable for clinical terminology without careful review. Second, the evaluation was conducted in a controlled environment rather than in a live hospital, so long-term effects on staff workload, patient satisfaction, and clinical outcomes remain to be measured.

Furthermore, the present prototype focuses primarily on outpatient consultation scheduling and does not yet integrate ancillary services such as laboratory, radiology, or pharmacy workflows. Extending the model to cover these services will require additional entities, workflows, and interoperability mechanisms with existing hospital information systems. Finally, the system currently employs relatively simple statistical methods for waiting-time estimation; while effective in practice, more advanced predictive models could further improve accuracy, particularly in clinics with highly variable consultation durations.

G. *Representative Patient Journey*

To illustrate how the proposed system operates in practice, consider a typical outpatient journey in a general medicine department. A patient experiencing recurrent symptoms accesses the hospital's public web portal from a low-cost smartphone. After selecting the preferred language, the patient creates an account, verifies the contact number via a one-time password, and completes a simple profile capturing demographic and contact details. The patient then navigates to the appointment module, chooses the required department, and is presented with a filtered list of doctors and available time slots.

The system immediately validates potential conflicts, such as overlapping bookings or attempts to schedule outside clinic hours, and highlights the earliest feasible slot. Once the patient confirms the appointment, a confirmation message is delivered via SMS and in-app notification, summarising the date, time, location, and any pre-visit instructions. A reminder is sent again on the previous day, along

with a link that allows the patient to reschedule within policy limits if circumstances change.

On the day of the visit, the patient arrives at the hospital and uses either a self-service kiosk or the reception desk to check in. The system verifies the appointment, assigns a queue token, and updates the live queue dashboard. The patient can now monitor the current serving token and estimated waiting time from a display screen in the waiting area or from a mobile device. If the doctor encounters an unexpected delay, the system recalculates estimates for all pending tokens and proactively notifies patients whose waiting time is significantly affected.

When the patient's turn approaches, a final notification is issued, prompting movement towards the consultation room. The doctor's interface displays relevant details, including visit reason and recent appointment history. After the consultation, the doctor marks the appointment as completed and, if integrated systems are available, records prescriptions and follow-up recommendations. The patient dashboard is updated to reflect the completed visit and stores a summary entry for future reference. This end-to-end scenario demonstrates how the system reduces uncertainty, improves communication, and streamlines coordination between patients and hospital staff.

H. *Observation and Decision*

The functional tests indicate that the integrated architecture successfully combines appointment booking, real-time queue tracking, and multi-channel notifications within a single platform. WebSocket-based updates provide immediate feedback when doctors change appointment statuses, enhancing transparency for patients, while SMS and voice reminders function reliably when configured correctly, demonstrating practical integration into hospital workflows.

From a performance perspective, the system remains responsive under moderate load, supporting multiple concurrent appointments, and the rules-based waiting-time estimation provides useful guidance without requiring complex predictive models. However, the current evaluation is limited to functional correctness and small-scale usage, with large-scale performance testing and real-world validation remaining as future work. Successful adoption of the system depends not only on technical design but also on effective deployment and operational practices.

The modular architecture supports flexible hosting, allowing smaller hospitals to use on-premise setups and larger institutions to deploy on cloud infrastructure. Proper staff training, gradual transition from manual systems, and clear workflow understanding are essential for smooth implementation. Continuous monitoring of system performance, including response times, queue behavior, and notification delivery, along with regular feedback from users, enables ongoing improvements. Overall, the system demonstrates its ability to address key inefficiencies in traditional appointment management while providing a scalable and adaptable foundation for real-world healthcare deployment.

VI. CONCLUSION

This paper presents a web-based appointment and queue management system that improves outpatient efficiency for patients, doctors, and administrators. Built using React, Express/Prisma, and a relational database, the system features live queue tracking, notifications, and role-based dashboards.

The system performs reliably and handles key workflows effectively, with user-friendly interfaces. Future work includes real-world deployment, integration with medical records, and the use of machine learning for better wait-time prediction and resource planning.

Overall, the system provides a scalable foundation for enhancing digital healthcare services.

ACKNOWLEDGMENT

The authors express their sincere gratitude to the Department of Computer Science and Engineering, RP Sarathy Institute of Technology, for their continuous guidance and support throughout the development of this project.

REFERENCES

[1] W. Cao, X. Wan, Z. Tu, and B. H. Heng, "A web-based appointment system to reduce waiting for outpatients," BMC Health Services Research, vol. 11, p. 318, Nov. 2011.

[2] Z. Zhu, B. H. Heng, and K. L. Teow, "Analysis of factors causing long patient waiting time and clinic overtime," Journal of Medical Systems, vol. 36, no. 2, pp. 707–713, Apr. 2012.

[3] X. Li et al., "Artificial intelligence-assisted reduction in patients' waiting time," BMC Health Services Research, vol. 21, p. 237, Mar. 2021.

[4] S. P. Nayak and B. Bhushan, "Literature review on appointment scheduling in hospitals management," American Journal of Drug Delivery and Therapeutics, vol. 8, no. 4, Sep. 2021.