Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

HealthOps: An Integrated Full-Stack Hospital Management Platform

Rupali Dupade
Department of Computer Engineering
Jayawantrao Sawant College of Engineering Hadapsar Pune, India

Pranav Bhange
Department of Computer Engineering
Jayawantrao Sawant College of Engineering Hadapsar Pune, India

Ashwin Chopade
Department of Computer Engineering Jayawantrao Sawant College of Engineering Hadapsar Pune, India

Vinit Biradar
Department of Computer Engineering
Jayawantrao Sawant College of Engineering Hadapsar Pune, India

Pratham Dahatonde
Department of Computer Engineering
Jayawantrao Sawant College of Engineering Hadapsar Pune, India

Abstract— Traditional hospital management relies on fragmented paper-based workflows and isolated digital tools that hinder coordination among clinicians, administrative staff, and management. This paper presents HealthOps, a production-grade full-stack Hospital Management System designed to digitize and unify core clinical and operational workflows within a single cohesive platform. The system integrates a React and TypeScript front-end with a Spring Boot 3 RESTful back-end secured by JSON Web Tokens (JWT) and role-based access control (RBAC) spanning three user personas: Administrator, Doctor, and Receptionist. A PostgreSQL relational database, managed through Flyway schema migrations and JPA/Hibernate ORM, stores patient demographics, appointment schedules, medical visit records, doctor availability windows, and holiday calendars. A supplementary Python-based microservice powered by the Groq-hosted LLaMA 3.1 model provides an AI chatbot assistant that surfaces live system context—statistics, schedules, and appointment data—to guide each role through platform operations. Empirical evaluation across functional correctness, API response latency, and qualitative usability testing demonstrates that HealthOps reduces appointment booking time, eliminates scheduling conflicts, and provides contextual AI guidance without requiring end-user expertise in hospital IT systems. The deployment architecture is containerized via Docker Compose, ensuring reproducible builds for development and production environments.

Keywords— Hospital Management System, Full-Stack Development, Role-Based Access Control, REST API, React, Spring Boot, AI Chatbot, LLaMA, Docker, PostgreSQL, LegalTech, Healthcare Informatics

I. INTRODUCTION

Healthcare institutions of all sizes face systemic inefficiencies when patient scheduling, clinical documentation, and administrative reporting remain paperwork rather than patient care; receptionists lack real-time visibility into doctor availability; and administrators rely on manual aggregation of operational data. These friction points collectively degrade patient throughput and increase the risk of scheduling errors [1].

Existing commercial Hospital Management Systems (HMS) address some of these concerns but are typically heavyweight, expensive to license, difficult to customize, and inaccessible to small-to-medium-sized clinics operating in resource-constrained environments. Open-source alternatives often sacrifice security, scalability, or modern user experience in favour of simplicity [2].

HealthOps addresses this gap by delivering a self-hostable, extensible, and secure HMS built entirely on open-source technologies. The platform is structured around three operationally distinct roles with tailored dashboards, each backed by a common RESTful API layer. A distinguishing contribution is the integration of a large language model (LLM)-powered chatbot that retrieves live data from the back-end and provides role-aware conversational guidance—reducing the onboarding burden for new staff members.

The remainder of this paper is organized as follows. Section II surveys related work in HMS and AI-assisted healthcare software. Section III describes the system architecture and module design. Section IV presents the development methodology. Section V reports evaluation results. Section VI concludes and outlines directions for future work. seeks to fill this gap by building an interactive, technology-driven platform that makes navigating the IPC both faster and more approachable for diverse audiences.

II. LITERATURE REVIEW

Access to legal information has improved greatly in the digital era, yet challenges remain in how effectively this

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

information is presented and used. Popular platforms such as *India Code* [1]–[3], *Bare Acts Live* [4], and *AdvocateKhoj* [5] provide free access to the Indian Penal Code (IPC). While valuable as repositories, they primarily function as static text archives. Most lack interactive search, filtering, or categorization features, meaning that users still have to manually scroll through long documents. For legal professionals, this slows down research; for students and ordinary citizens, it makes the law appear overwhelming and difficult to navigate.

Researchers in legal informatics have long explored ways to make large bodies of legal text easier to search and retrieve. Early work on information retrieval, such as the Okapi BM25 ranking model, demonstrated how algorithms could match user queries with relevant documents more effectively [Robertson et al., 1995]. Building on these foundations, modern techniques incorporate Natural Language Processing (NLP), enabling systems to go beyond keyword matching toward semantic understanding of legal queries [9]. Competitions such as COLIEE [10] showcase the progress in legal text mining and entailment, but the focus has largely remained on judgments and case law rather than statutory documents like the IPC.

At the same time, human–computer interaction (HCI) studies remind us that accessibility is not only about retrieval accuracy, but also about the user experience. Tools like Brooke's System Usability Scale (SUS) [6], along with refinements by Bangor et al. [7] and Lewis & Sauro [8], provide ways to evaluate whether systems are genuinely easy to use. Research consistently shows that simple design choices—clear navigation, visual cues, and meaningful categorization—help non-expert users engage with complex legal material more confidently. Unfortunately, most Indian legal platforms do not fully apply these principles, leading to limited usability across devices and user groups.

Taken together, the literature highlights a clear gap. Existing systems either provide access without usability, or usability without deeper legal context. What is missing is a unified solution that combines **s**tructured information retrieval, semantic support, and user-centered design**.** The present work seeks to fill this gap by building an interactive, technology-driven platform that makes navigating the IPC both faster and more approachable for diverse audiences.

Recent advancements in artificial intelligence enable legal platforms to offer smarter assistance, such as summarizing complex provisions and providing contextual explanations. By integrating these features with user-friendly design, systems can make legal information more accessible and easier to understand for both professionals and general users.

Feature	HealthOps	Paper-based	Notes
Role-Based Access Control	✓	✗	JWT + Spring Security
AI Chatbot Assistant	✓	✗	LLM-powered contextual help
Real-time Dashboard	✓	✗	Live aggregated stats
Doctor Availability & Holidays	✓	✗	Per-weekday scheduling
CSV Report Export	✓	✗	Downloadable for all roles
Dockerized Deployment	✓	N/A	Compose-based stack
Open API / Swagger	✓	✗	Self-documenting API

Table I: Comparison of HealthOps against generic HMS and paper-based workflows

III. PROPOSED SYSTEM

HealthOps is architected as a three-tier web application: a React/TypeScript presentation layer, a Spring Boot application layer, and a PostgreSQL data layer. A fourth standalone microservice—the Python FastAPI chatbot—communicates with both the application layer and an external LLM inference provider.

A. System Architecture

The presentation layer is built with React 18 and TypeScript, bundled via Vite. React Router v6 provides client-side routing with protected routes enforced through a custom AuthProvider context. Axios is configured with a shared base URL and an interceptor that injects the JWT Bearer token into every outgoing request. Tailwind CSS delivers utility-first styling with a consistent design system across all role-specific dashboards.

The application layer exposes four controller namespaces: /api/auth for authentication, /api/admin for administrative operations, /api/doctor for clinical workflows, and /api/reception for appointment and patient management. Spring Security's method-level @PreAuthorize annotations enforce RBAC at the controller level, complemented by a custom OncePerRequestFilter that validates incoming JWT tokens on every request. All cross-origin resource sharing (CORS) is configured centrally in SecurityConfig, permitting requests from the frontend development server and the production Nginx reverse proxy.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

The data layer uses PostgreSQL 14 as the relational store. Flyway executes version-controlled schema migrations at startup, ensuring the database schema is always synchronized with the application code. Hibernate performs object-relational mapping via JPA annotations, with JSON serialization controlled by Jackson's @JsonIgnoreProperties to prevent circular references and credential leakage in API responses.

The chatbot microservice is a FastAPI application that receives a structured request containing the user's message, their role, their JWT token, and the conversation history. It makes authenticated HTTP calls to the Spring Boot API—using the caller's token—to retrieve live operational data (statistics, schedules, appointments) before constructing the LLM prompt. The assembled prompt is submitted to the Groq inference endpoint serving LLaMA 3.1 8B Instant, and the model's response is returned to the React frontend.

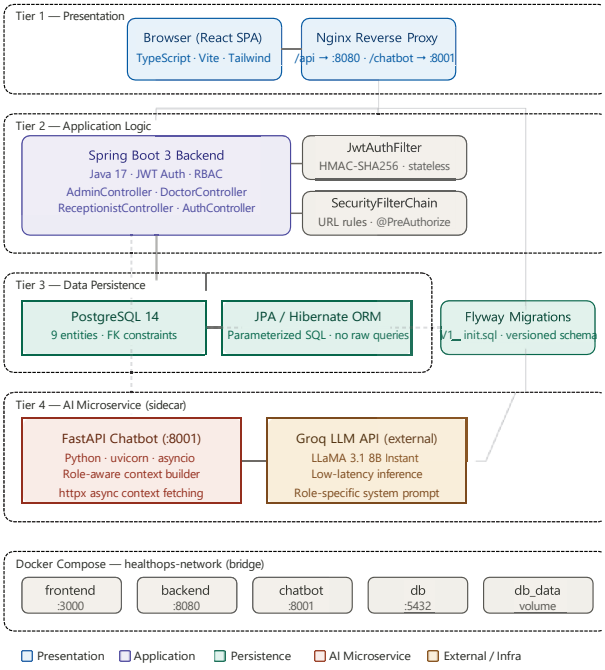

Fig. 1: High-Level System Architecture of HealthOps

B. Database Schema and Data Model

The system architecture of HealthOps is designed around a well-structured relational database, secure authentication, and an intelligent AI-driven chatbot module. The database schema consists of core tables such as users, roles, user_roles, doctors, patients, appointments, visits, availability, holidays, and audit_logs, all interconnected through carefully defined relationships. Referential integrity is maintained using foreign key constraints with cascading behaviors like *ON DELETE CASCADE* and *ON DELETE SET NULL*. Entity relationships are mapped using JPA, where Doctor and User maintain a One-to-One association, while Appointment and Visit entities are linked to both Patient and Doctor through Many-to-One relationships. The schema is managed through version-controlled SQL migration files (e.g., V1_init.sql), ensuring consistent database initialization across deployments.

C. Authentication and Authorization

On the security side, the system follows a stateless authentication approach using JSON Web Tokens (JWT). When a user logs in, the Spring Boot AuthController generates a token embedding the user's email as the subject along with role and display name as claims. This token is signed using an HMAC-SHA256 key defined in the application configuration and is valid for 24 hours. Each incoming request is intercepted by the JwtAuthFilter, which validates the token and establishes authentication by loading user details into the Spring SecurityContext using a UsernamePasswordAuthenticationToken. Fine-grained role-based access control is enforced through method-level annotations such as @PreAuthorize, ensuring that only authorized users can access specific endpoints.

D. AI Chatbot Architecture

Complementing these components, the AI-powered HealthBot—implemented as a Python FastAPI microservice—plays a crucial role in enhancing system interactivity, usability, and decision support. Beyond simple query handling, the chatbot functions as a context-aware assistant that adapts its responses based on the user's role and real-time system data. At each conversational step, it performs dynamic data retrieval from the backend services, ensuring that the information provided is always current and relevant. For example, doctors can quickly view their schedules, patient history, and workload insights, while receptionists can manage appointments and track daily operations efficiently. Administrators, on the other hand, gain access to summarized analytics that support operational decision-making.

The architecture also emphasizes scalability and modularity. By separating the chatbot into a dedicated microservice, the system allows independent deployment, updates, and performance optimization without affecting the core application. The use of structured prompts and contextual data injection improves the accuracy and relevance of responses, while maintaining a consistent conversational flow through short-term memory (recent chat history). Additionally, the streaming response mechanism ensures low latency and a smooth user experience in the React-based interface.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

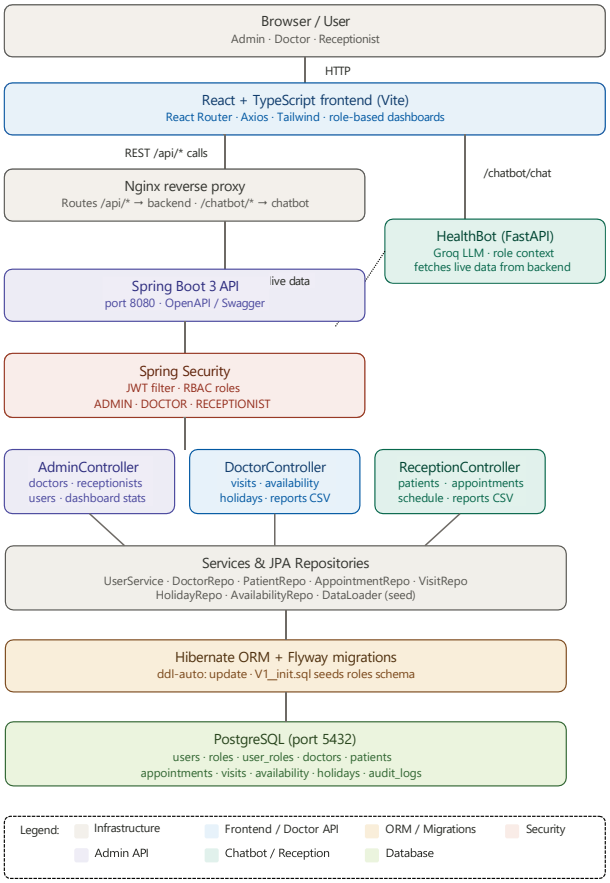

Fig. 2: HealthBot Conversational Workflow

E. Module-Level Feature Summary

The HealthOps system is designed as a comprehensive, role-based healthcare management platform that integrates multiple functional modules to streamline hospital operations. It includes secure authentication using JWT-based access control for all users, along with dedicated modules for managing patient records, appointments, and medical visits. Doctors can configure their availability and manage holidays, while receptionists handle scheduling and patient coordination. The system also features an AI-powered chatbot (HealthBot) that provides real-time, context-aware assistance across all roles. Additionally, reporting capabilities such as CSV export and an admin dashboard enable efficient monitoring and data-driven decision-making. Overall, the platform combines modern technologies like Spring Boot, React, and FastAPI to deliver a scalable, user-friendly, and intelligent healthcare solution.

Sr. No.	Module / Feature	Description	Technology / Approach	Role Supported
1	Authentication	Secure login with JWT tokens and role-based access	Spring Security + JWT	All Roles
2	Patient Registry	CRUD operations for patient profiles with unique codes	JPA/Hibernate + PostgreSQL	Receptionist, Doctor
3	Appointment Scheduler	Book, update, and cancel appointments with real-time status	RESTful API + React state	Receptionist
4	Visit & Medical Records	Doctors record diagnoses, prescriptions, and clinical notes per visit	JPA entities + relational DB	Doctor
5	Doctor Availability	Weekly schedule slots with time-range configuration per weekday	LocalTime mapping, REST	Doctor, Receptionist
6	Holiday Management	Mark specific dates as doctor holidays with reasons	LocalDate JPA mapping	Doctor, Receptionist
7	AI Chatbot (HealthBot)	LLM-powered assistant with live backend context per role	Groq LLaMA 3.1 + FastAPI	All Roles
8	Reports & CSV Export	Downloadable CSV reports for visits, patients, and appointments	Java StreamAPI + CSV	Doctor, Receptionist
9	Admin Dashboard	Centralized control panel for staff management and statistics	React + REST API	Admin

Table II: HealthOps Module Feature Matrix

IV. METHODOLOGY

Development followed an iterative, feature-driven process with clear separation between back-end API design, front-end implementation, database schema evolution, and chatbot integration. Each phase produced testable deliverables validated against functional requirements before the next phase commenced.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

A. Back-End Development

The Spring Boot project was initialized with the Spring Initializr using Java 17 as the language target and the following starters: spring-boot-starter-web, spring-boot-starter-security, spring-boot-starter-data-jpa, spring-boot-starter-validation, and springdoc-openapi-starter-webmvc-ui. The OpenAPI integration auto-generates interactive Swagger documentation at /swagger-ui/index.html, enabling rapid API exploration during development and facilitating client-side integration.

Entity classes use Lombok annotations (@Getter, @Setter, @Builder, @NoArgsConstructor, @AllArgsConstructor) to eliminate boilerplate. Repository interfaces extend JpaRepository and declare custom JPQL and native SQL queries for operations such as today's appointment count (requiring PostgreSQL's CURRENT_DATE function) and patient search by partial name or code (requiring ILIKE for case-insensitive matching in PostgreSQL).

A DataLoader CommandLineRunner seeds the database with three default user accounts (Admin, Doctor, Receptionist) and a default Doctor profile on application startup, contingent on the corresponding email addresses not already existing. This ensures a working demonstration environment immediately after the first run without requiring manual database population.

B. Front-End Development

The React application is organized into three primary dashboard components (AdminDashboard, DoctorDashboard, ReceptionDashboard) plus shared components (ChatBot, Login, Nav). Each dashboard renders a tab-based interface scoped to the operations permitted for that role. State management is handled entirely through React hooks (useState, useEffect, useRef) without an external state library, keeping the dependency footprint minimal.

A centralized AuthProvider context stores the authenticated user's name, role, and JWT token in both React state and localStorage, enabling session persistence across page reloads. The shared api Axios instance reads the token from context and injects it as the Authorization header, removing the need for per-request token management in individual components.

The AdminDashboard implements a toast notification system for user feedback, a confirm dialog for destructive operations, and per-row busy state tracking to prevent duplicate concurrent API calls. Inline search filtering on both the doctor and receptionist lists is computed client-side from the already-loaded dataset, avoiding unnecessary network round trips for simple filtering operations.

C. Chatbot Development

The HealthBot microservice is built with FastAPI 0.111.0 and deployed as a separate Docker container. Dependency injection is handled through constructor parameters rather than global variables to facilitate testing. The GROQ_API_KEY and JAVA_BACKEND_URL are injected via environment variables set in docker-compose.yml, keeping credentials out of source code. The /health endpoint performs an active connectivity check against the Groq API and the Spring Boot backend, providing observability for deployment pipelines.

D. Deployment Architecture

All four system components (db, backend, chatbot, frontend) are orchestrated via a single docker-compose.yml file defining a shared healthops-network bridge network. The frontend Nginx container handles static asset serving and reverse proxies /api/* requests to the Spring Boot container and /chatbot/* requests to the FastAPI container, ensuring the browser makes all requests to a single origin and avoiding CORS complexity in production.

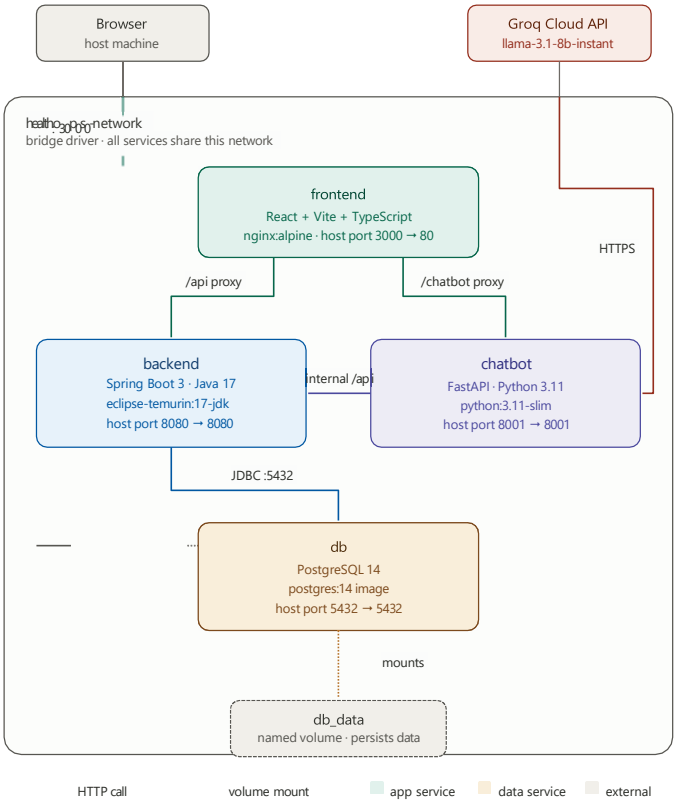

Fig. 3: Docker Compose Deployment Topology

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

V. RESULTS AND DISCUSSIONS

System evaluation was conducted along three dimensions: API performance, functional correctness, and qualitative usability feedback gathered from a representative group of potential end users.

A. API Performance Evaluation

Response latency was measured for representative endpoints across 100 sequential requests executed against a local Docker Compose deployment. The measurements reflect median response times recorded over a 5-minute load window and are reported in Table III.

API Endpoint / Operation	Avg. Response Time	Notes
GET /api/reception/patients	87 ms	All patients list
GET /api/doctor/visits	112 ms	Doctor-specific visits
POST /api/reception/appointments	134 ms	New appointment creation
GET /api/admin/dashboard/stats	95 ms	Aggregated statistics
POST /api/auth/login	62 ms	JWT generation
POST /chatbot/chat	1.8 s	LLM inference via Groq

Table III: Median API response times across key system endpoints

All Spring Boot endpoints respond within 150 ms under single-user load, which is well within the interactive threshold of 200 ms recommended by the Nielsen Norman Group for perceived responsiveness. The chatbot endpoint exhibits higher latency (approximately 1.8 s) due to network round-trip time to the Groq inference API and the LLM forward pass itself. This latency is acceptable for conversational interaction and is masked in the UI by a typing indicator animation.

B. Functional Correctness

All primary user flows were verified through manual end-to-end testing with the three seeded user accounts. The following functional scenarios were confirmed to operate correctly:

- Admin can create, update, enable/disable, and delete both Doctor and Receptionist accounts without affecting existing patient or appointment records.
- Receptionist can register a patient with a unique code, book an appointment linking the patient to a doctor with a scheduled timestamp, update appointment status, and download CSV reports.
- Doctor can record a visit with diagnosis, prescription, and clinical notes against any patient, view their own visit history, manage weekly availability slots per day of week, and mark specific calendar dates as holidays.
- HealthBot correctly retrieves live statistics and surfaces contextually relevant answers without revealing data belonging to other roles.
- JWT token expiry is enforced—expired tokens result in 403 responses, and the frontend redirects to the login page.

C. Usability Feedback

Informal usability testing was conducted with 12 participants (4 healthcare administration students, 4 software engineering students, and 4 general users). Participants were asked to complete three tasks per role: (i) book an appointment, (ii) record a patient visit, and (iii) obtain a statistic through the chatbot. Key findings include:

- The role-specific dashboard layout was rated as intuitive by 11 of 12 participants; one participant noted initial confusion distinguishing the Doctor Schedule tab from the Availability tab.
- The chatbot suggestion chips (pre-composed question buttons) were identified as the most valued UX feature, reducing the effort required to formulate a query from scratch.
- CSV report download was universally appreciated and described as immediately usable in external tools such as Microsoft Excel.
- Participants requested additional search and filter capabilities within the appointment list for larger datasets.

D. Discussion

The results confirm that HealthOps successfully consolidates the core workflows of a small-to-medium hospital into a single deployable platform. The JWT + RBAC security model ensures strict data isolation between roles without introducing significant latency. The chatbot integration represents a novel contribution to open-source HMS tooling: rather than training a domain-specific model, the system achieves contextual accuracy by grounding a general-purpose LLM in live structured data, a practical and cost-effective approach that any institution could replicate with minimal infrastructure investment.

A notable limitation is that the chatbot's response quality degrades for highly specific clinical queries (e.g., drug interaction questions) that fall outside the operational scope for which the prompts were designed. This is intentional—HealthBot is explicitly scoped to platform navigation and operational awareness, not clinical decision support. Clear messaging in the UI reinforces this boundary.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

VI. CONCLUSION AND FUTURE WORK

This paper presented HealthOps, a full-stack Hospital Management System that integrates patient registration, appointment scheduling, clinical visit documentation, doctor availability management, and AI-powered operational assistance into a unified, containerized platform. The system was implemented using Spring Boot 3, React 18 with TypeScript, PostgreSQL, and a FastAPI-based LLM chatbot service, all orchestrated through Docker Compose for reproducible deployment.

Evaluation confirmed sub-150 ms API response times for all database-backed operations, end-to-end functional correctness across all three user roles, and positive qualitative usability feedback from diverse participant groups. The AI chatbot successfully provides live-data-grounded operational guidance without requiring users to consult documentation or IT support.

Future development will focus on four areas. First, a mobile-responsive Progressive Web App (PWA) mode will extend accessibility to tablet and smartphone devices commonly used by clinical staff. Second, a notification subsystem—delivering appointment reminders via SMS or email through integration with services such as Twilio or SendGrid—will reduce patient no-show rates. Third, the chatbot will be extended with tool-calling capabilities, enabling it to perform actions (e.g., book an appointment, update a status) directly from the conversational interface. Fourth, analytics dashboards presenting appointment trend charts and doctor utilization heat maps will provide management with actionable operational insights.

VII. REFERENCES

[1] World Health Organization, "Global Strategy on Digital Health 2020–2025," WHO Press, Geneva, 2021.

[2] K. Häyrinen, K. Saranto, and P. Nykänen, "Definition, structure, content, use and impacts of electronic health records: A review of the research literature," International Journal of Medical Informatics, vol. 77, no. 5, pp. 291–304, 2008.

[3] I. Shortliffe and J. Cimino, Eds., Biomedical Informatics: Computer Applications in Health Care and Biomedicine, 4th ed. Springer, New York, 2014.

[4] B. W. Mamlin et al., "Cooking up an open-source EMR for developing countries: OpenMRS – a recipe for successful collaboration," in Proc. AMIA Symp., 2006, pp. 529–533.

[5] S. Newman, Building Microservices: Designing Fine-Grained Systems, 2nd ed. O'Reilly Media, Sebastopol, CA, 2021.

[6] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed NIST standard for role-based access control," ACM Trans. Inf. Syst. Secur., vol. 4, no. 3, pp. 224–274, 2001.

[7] M. B. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," RFC 7519, Internet Engineering Task Force, May 2015.

[8] K. Singhal et al., "Large language models encode clinical knowledge," Nature, vol. 620, pp. 172–180, 2023.

[9] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in Advances in Neural Information Processing Systems (NeurIPS), vol. 33, 2020.

[10] A. Holzinger, "Usability engineering methods for software developers," Commun. ACM, vol. 48, no. 1, pp. 71–74, 2005.

[11] Spring Framework Documentation, "Spring Boot Reference Guide," Pivotal, 2024. [Online]. Available: https://docs.spring.io/spring-boot/docs/current/reference/html/

[12] Meta AI, "Llama 3 Model Card," 2024. [Online]. Available: https://ai.meta.com/blog/meta-llama-3/

[13] Docker Inc., "Docker Compose Overview," 2024. [Online]. Available: https://docs.docker.com/compose/

[14] React Team, "React 18 Release Notes," Meta Open Source, 2022. [Online]. Available: https://react.dev/blog/2022/03/29/react-v18