International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 10 | Oct - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

Care Flow AI: AI-Powered Smart Hospital Appointment Scheduling and Management System

Sanjay Krishna B[1],Saravanan P[2], Sruthi V[3], Subhasri KT[4], Vignesh Kumar S[5]

1 Assistant Professor, Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology,
Email: sanjaykrishnab@siet.ac.in

2 Student, Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, India.
Email:saravananp23cse@srishakthi.ac.in

3 Student, Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, India.
Email: sruthiv23cse@srishakthi.ac.in

4 Student, Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, India.
Email: subhasrikt23cse@srishakthi.ac.in

5 Student, Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, India.
Email:vigneshkumars23cse@srishakthi.ac.in

Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, India

ABSTRACT

CareFlow AI is a microservices-based smart hospital appointment scheduling and management platform designed to improve the efficiency, accessibility, and reliability of healthcare operations. The system combines intelligent scheduling, doctor matching, appointment management, emergency handling, laboratory workflows, billing, payments, notifications, and role-based access within a unified web application. The frontend is implemented with Next.js, React, TypeScript, and responsive web technologies, while the core backend uses Java 21 and Spring Boot with Spring Security, JPA/Hibernate, PostgreSQL, and Flyway. A dedicated Python FastAPI service provides the artificial-intelligence layer for scheduling and matching decisions. Redis is used for caching and concurrency control, while Prometheus and Grafana provide application observability. JWT-based authentication and role-based authorization separate patient, doctor, staff, and administrator capabilities. The architecture is designed for containerized deployment and future Kubernetes orchestration. Functional and regression testing across 375 test cases achieved a 98.40% pass rate, with no critical or high-severity findings. The system demonstrates that combining AI-assisted decision support with conventional hospital management workflows can reduce manual coordination and provide a more structured digital experience for patients and healthcare staff.

Keywords: *hospital management, intelligent scheduling, doctor matching, artificial intelligence, microservices, Spring Boot, FastAPI, Next.js, Redis, PostgreSQL, healthcare automation*

I. INTRODUCTION

Healthcare institutions manage a large number of time-sensitive activities every day, including patient registration, appointment scheduling, doctor availability, emergency cases, laboratory investigations, billing, payments, and communication. When these activities are coordinated manually or through disconnected systems, patients may experience long waiting times, scheduling conflicts, and limited visibility into appointment status. Doctors and administrative staff also need reliable access to current information before making operational decisions.

CareFlow AI was developed to address these challenges through a unified digital platform that combines hospital management functions with AI-assisted decision support. The system provides separate capabilities for patients, doctors, staff, and administrators while maintaining a common backend and consistent authorization model. Patients can manage profiles, search for doctors, request appointments, view appointment information, access relevant medical and billing information, and receive notifications. Doctors can manage schedules and appointments, while staff and administrators can operate hospital-wide workflows.

The central innovation of the system is the use of an AI microservice to assist with smart scheduling and doctor

matching. Instead of treating appointment booking as a simple first-available-slot operation, the platform can consider structured information such as doctor specialization, availability, appointment requirements, and scheduling constraints. The AI service is isolated from the main transactional backend so that AI workloads can evolve independently without weakening the consistency of core hospital operations.

CareFlow AI also addresses concurrency and reliability. Appointment booking and related resources can be accessed by multiple users simultaneously, so Redis-based caching and locking mechanisms are incorporated to reduce race conditions. PostgreSQL provides durable transactional storage, while Spring Boot exposes secured REST APIs. Monitoring through Prometheus and Grafana supports operational visibility, and the complete architecture is suitable for containerized deployment and Kubernetes-based scaling.

II. LITERATURE SURVEY

Digital hospital information systems have traditionally focused on storing patient records, scheduling appointments, and supporting administrative processes. Modern healthcare platforms increasingly combine these functions with cloud-native services, event-driven communication, analytics, and decision-support systems. A major challenge is balancing intelligent automation with the consistency and security required for clinical and financial transactions.

Intelligent scheduling research shows that appointment allocation can be treated as an optimization problem involving resource availability, patient preferences, service duration, and urgency. Rule-based schedulers are predictable but may become difficult to maintain when many constraints are introduced. AI-assisted approaches can provide recommendations while allowing the transactional application to remain responsible for validation and final persistence. CareFlow AI follows this separation by keeping AI inference in a dedicated FastAPI service while Spring Boot controls authenticated business operations.

Microservice architecture is widely used when a system contains independently evolving capabilities. A hospital platform may benefit from separating AI processing, core APIs, persistence, monitoring, and other infrastructure concerns. Spring Boot provides the central service layer in CareFlow AI, FastAPI provides the AI interface, PostgreSQL provides relational persistence, and Redis supports low-latency access and concurrency control. This arrangement also supports independent scaling and deployment.

Security is particularly important in healthcare applications because patient, appointment, billing, and authentication data must not be exposed to unauthorized users. CareFlow AI uses stateless JWT authentication and role-based authorization. Authentication establishes the identity of the caller, while authorization rules restrict access according to PATIENT, DOCTOR, STAFF, or ADMIN roles. Passwords are protected using BCrypt, and invalid or missing credentials are separated from authenticated-but-forbidden requests.

Web application research also emphasizes responsive interfaces because hospital users may access systems from desktop computers, tablets, and smaller screens. The CareFlow AI frontend uses Next.js and React with responsive layouts so that core workflows remain usable across screen sizes. A common API and role-aware navigation model reduces duplicated business logic while allowing each user group to see relevant functionality.

III. PROPOSED METHODOLOGY

The proposed CareFlow AI methodology is based on four principles: intelligent assistance, transactional reliability, role-based security, and modular scalability. The system first authenticates the user and establishes the permitted role. The request is then processed by the Spring Boot API, which validates business rules and interacts with PostgreSQL or Redis as required. AI-dependent requests are forwarded to the Python FastAPI service. The result is returned to the application and, where appropriate, persisted as part of the hospital workflow.

The system is organized into the following major modules: authentication and authorization, patient management, doctor management, staff management, appointment scheduling, AI doctor matching, emergency management, laboratory management, billing and payments, notifications, and administrative monitoring. Each module follows the same security and validation conventions while exposing only the operations appropriate for the requesting role.

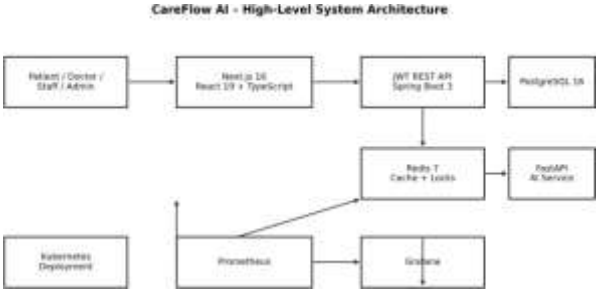


Fig. 1. High-level architecture of the CareFlow AI platform.

The frontend communicates with the Spring Boot service through REST APIs secured with JWT. The backend validates the token and applies role-based access control before executing the requested operation. PostgreSQL stores transactional entities such as users, doctors, appointments, emergency cases, laboratory orders, invoices, and payment-related records. Flyway manages database migrations so that schema changes can be versioned and reproduced consistently.

The AI service is implemented with Python and FastAPI. It is designed as a separate service so that AI models and scheduling logic can be updated without changing the core Java service. The service receives structured requests, evaluates available information, and returns a recommendation or matching result. The backend remains responsible for authorization, transactional validation, and persistence, preventing an AI response from bypassing application security.

Redis is used for caching and distributed coordination. Frequently accessed information can be cached to reduce repeated database work, while locking mechanisms help protect operations where simultaneous requests could otherwise create duplicate bookings or inconsistent state. This is particularly important for appointment slots because two users may attempt to reserve the same availability at nearly the same time.

The platform also incorporates observability. Spring Boot Actuator exposes operational metrics, Prometheus collects monitoring data, and Grafana provides dashboards for visualization. This allows developers and administrators to identify service-health problems, response-time changes, and resource-related issues without relying only on user reports.

IV. SYSTEM IMPLEMENTATION

The CareFlow AI frontend is developed using Next.js 16, React 19, and TypeScript. It provides role-aware pages and workflows for patients, doctors, staff, and administrators. The interface is organized around a shared layout system so that navigation, authentication state, notifications, and responsive behavior remain consistent. The frontend communicates with backend REST endpoints and maintains the authenticated JWT session for subsequent requests.

The backend is implemented using Java 21 and Spring Boot 3. Spring Security handles authentication and authorization, while JPA/Hibernate provides object-relational mapping. Flyway manages schema migrations, and PostgreSQL 16 stores persistent data. REST controllers expose versioned API endpoints. Service classes contain business rules, while repository components provide controlled database access. This separation improves maintainability and makes the individual workflows easier to test.

Authentication begins with registration or login. Successful authentication returns a signed JWT containing the necessary identity and role information. The JWT filter validates the token for protected requests. Requests without valid credentials are rejected as unauthorized, while authenticated users attempting to access another role's protected operation receive a forbidden response. This distinction was explicitly verified during regression testing.

The appointment module is a core part of the system. Patients can search for suitable doctors and available slots, submit appointment requests, and track appointment status. The scheduling workflow validates doctor availability and prevents conflicting bookings. Redis locking is used for concurrency-sensitive operations so that simultaneous requests cannot easily create duplicate reservations for the same slot.

The AI doctor-matching component uses structured request information to recommend appropriate doctors based on available scheduling and clinical service attributes represented by the system. The objective is not to replace clinical judgment but to reduce the administrative effort required to locate an appropriate provider. The final appointment remains subject to the application's validation, availability, and authorization rules.

The emergency module supports a structured emergency queue. Cases can be created with patient information, complaint details, acuity, assigned doctor or department, bed-related information, waiting time, and workflow status. Typical states include WAITING, IN_ASSESSMENT, IN_TREATMENT, ADMITTED, DISCHARGED, TRANSFERRED, and CANCELLED. This provides staff with a more visible workflow for urgent cases.

The laboratory module supports laboratory order creation, test categories, sample collection, processing, result entry, completion, and critical-result handling. Laboratory information can be associated with the relevant patient and clinical workflow so that authorized staff and doctors can retrieve results. The billing module provides patient billing search, invoice generation, line items, totals, payment tracking, and receipt-related operations. These modules reuse the common authentication, validation, persistence, and error-handling architecture.

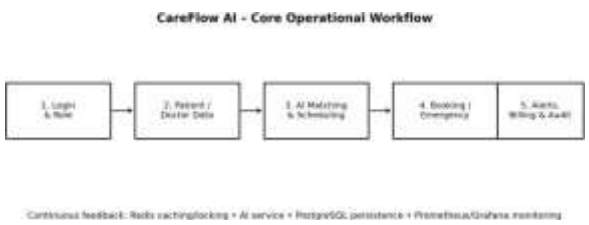


Fig. 2. Core operational workflow from authentication to hospital services.

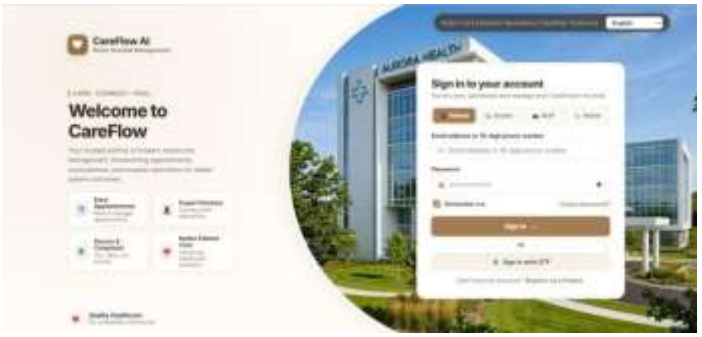
Notifications provide status information to users when important workflow events occur. The administrative portal brings together doctor, staff, user, emergency, laboratory, and billing management. Administrative operations are protected by role-based access rules, and sensitive actions are not exposed to patient accounts.

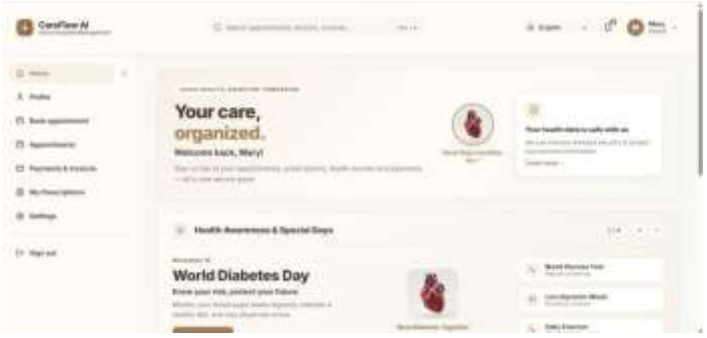
The deployment model is designed around containers and can be orchestrated using Kubernetes. The development architecture includes the Next.js frontend, Java Spring Boot API, Python FastAPI AI service, PostgreSQL, Redis, Prometheus, and Grafana. Containerization makes the environment reproducible and supports future horizontal scaling. The services can be independently restarted or scaled according to workload.

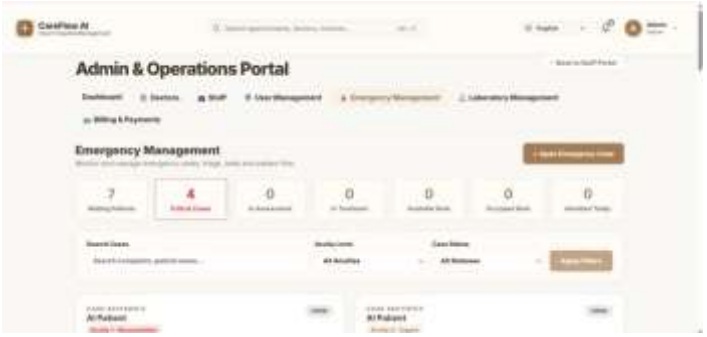
Testing is performed at several levels. Authentication and authorization are tested with valid and invalid credentials. API validation tests malformed and incomplete payloads. Appointment tests include concurrent booking attempts. Payment and receipt tests verify transactional behavior. Emergency, laboratory, and billing modules are tested as complete workflows. Responsive and localization checks validate frontend behavior, while database and Redis tests verify persistence and concurrency behavior.

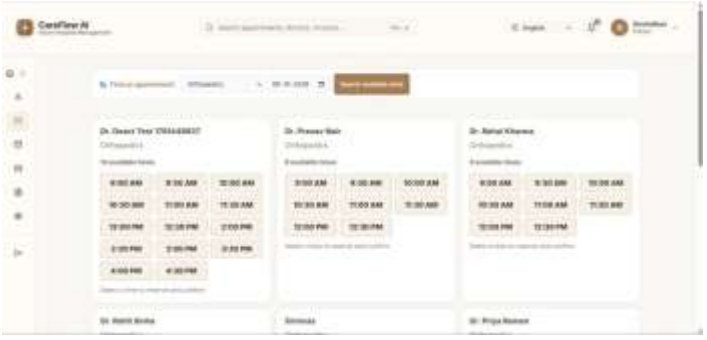
V. ADVANTAGES

- Intelligent scheduling support: reduces manual effort by using AI-assisted doctor matching and appointment recommendations.
- Unified hospital workflow: combines appointments, emergency operations, laboratory services, billing, payments, notifications, and administration in one platform.
- Strong role separation: patient, doctor, staff, and administrator operations are protected by JWT authentication and role-based authorization.
- Concurrency protection: Redis-based locking and transactional database operations reduce the risk of duplicate appointment reservations.
- Modular architecture: the AI service is separated from the core Java backend, allowing AI models and logic to evolve independently.
- Operational visibility: Prometheus and Grafana provide monitoring and dashboard capabilities for service health and performance.
- Responsive user experience: the Next.js interface is designed to adapt to desktop, tablet, and smaller screens.
- Scalable deployment: containerized services and Kubernetes compatibility provide a foundation for future horizontal scaling.
- Maintainable data management: PostgreSQL, JPA/Hibernate, and Flyway provide structured relational persistence and versioned schema management.


CareFlow AI
Welcome to CareFlow
Sign in to your account


Your care, organized.
World Diabetes Day


Admin & Operations Portal
Emergency Management



VI. RESULTS AND ANALYSIS

The implemented CareFlow AI prototype was evaluated through a broad regression audit covering authentication, role-based access control, patient and doctor workflows, staff and administration, appointments, payments, receipts, emergency, laboratory, billing, notifications, localization, responsive behavior, API validation, database operations, Redis locking, AI services, and security. The audit used valid authenticated sessions for all four application roles, avoiding the false failures caused by invalid test accounts in an earlier test cycle.

The latest regression audit contained 375 test cases. A total of 369 tests passed and 6 tests failed, giving an overall pass rate of 98.40%. There were no P0 or P1 findings. One P2 finding concerned a patient medical-record request returning HTTP 500 when no clinical records existed; the expected behavior was an empty list. Three P3 findings concerned authorization-versus-validation behavior and duplicate registration conflict handling, while two P4 findings involved optional strict validation expectations for malformed login inputs.

All major hospital workflows were reported as passing in the audit: appointments 30/30, payments 20/20, receipts 10/10, emergency 20/20, laboratory 20/20, billing 20/20, notifications 10/10, localization 15/15, responsive testing 15/15, database testing 15/15, Redis/locking 15/15, AI testing 10/10, and security testing 20/20. These results indicate that the primary system functions are stable while a small number of edge cases remain suitable for targeted refinement.

The audit also confirmed the expected separation between authentication and authorization. Requests without authentication or with invalid tokens return unauthorized responses, while authenticated users attempting operations outside their role are rejected as forbidden. This is important for a hospital application because merely possessing a valid account must not grant access to every category of information or operation.

From a performance and architecture perspective, the use of Redis for caching and locking reduces repeated database work and helps coordinate concurrent operations. PostgreSQL provides the transactional foundation, while the AI service isolates computational logic. Prometheus and Grafana improve operational visibility. Together, these components provide a practical foundation for evolving the prototype into a production-oriented hospital platform.

VII. CONCLUSION

CareFlow AI demonstrates how artificial intelligence and modern software architecture can be combined to improve hospital appointment scheduling and operational management. The system unifies patient, doctor, staff, and administrative workflows while adding AI-assisted scheduling and doctor matching. Its microservices-oriented design separates transactional hospital operations from AI processing, making the system easier to maintain and evolve.

The implementation combines Next.js, React, TypeScript, Java 21, Spring Boot, Spring Security, PostgreSQL, Redis, Python FastAPI, Prometheus, Grafana, and container-oriented deployment. The architecture emphasizes security, concurrency control, observability, and modularity. Testing results show strong coverage of the implemented functionality, with a 98.40% overall regression pass rate and no critical or high-severity findings in the latest audit.

Although the current system provides a strong prototype foundation, healthcare production deployment would require additional clinical validation, operational policies, compliance assessment, integration with established hospital information systems, and more extensive real-world testing. The present implementation therefore serves as a scalable technical foundation rather than a replacement for clinical decision-making.

VIII. FUTURE WORK

Future versions of CareFlow AI can extend the intelligent scheduling engine with more advanced optimization techniques that jointly consider appointment duration, urgency, doctor workload, patient preferences, department capacity, and historical scheduling patterns. Explainable recommendations can also be added so that staff can understand why a doctor or slot was recommended.

The platform can be integrated with external healthcare interoperability standards and existing hospital information systems. Additional notification channels such as SMS, email, and secure messaging can improve communication. Mobile applications for patients and clinicians could provide appointment reminders, live status updates, and emergency coordination.

Further improvements can include predictive analytics for patient flow, laboratory workload, bed occupancy, and appointment demand. Kubernetes-based autoscaling, centralized logging, distributed tracing, and advanced observability can strengthen the deployment model. Security can be expanded with

stronger secret management, audit trails, rate limiting, penetration testing, and compliance-oriented controls.

Finally, the identified edge cases from regression testing can be resolved systematically. Empty medical-record collections should return a valid empty response, duplicate registration should produce a clear conflict response, and authorization behavior should remain consistent with the security model even when invalid request payloads are supplied. These refinements will improve robustness before a production deployment.

REFERENCES

[1] Spring Boot Documentation, "Spring Boot Reference Documentation," VMware/VMware Tanzu. Available: https://docs.spring.io/spring-boot/

[2] Spring Security Documentation, "Spring Security Reference," Spring. Available: https://docs.spring.io/spring-security/reference/

[3] Next.js Documentation, "Next.js Documentation," Vercel. Available: https://nextjs.org/docs

[4] FastAPI Documentation, "FastAPI – Modern Web Framework for Building APIs with Python," Available: https://fastapi.tiangolo.com/

[5] PostgreSQL Global Development Group, "PostgreSQL Documentation," Available: https://www.postgresql.org/docs/

[6] Redis Documentation, "Redis Documentation," Available: https://redis.io/docs/

[7] Kubernetes Documentation, "Kubernetes Documentation," Cloud Native Computing Foundation. Available: https://kubernetes.io/docs/

[8] Prometheus Authors, "Prometheus Documentation," Available: https://prometheus.io/docs/

[9] Grafana Labs, "Grafana Documentation," Available: https://grafana.com/docs/grafana/latest/

[10] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," RFC 7519, Internet Engineering Task Force, 2015. Available: https://www.rfc-editor.org/rfc/rfc7519

[11] OWASP Foundation, "OWASP API Security Top 10," Available: https://owasp.org/www-project-api-security/

[12] M. Fowler, "Microservices," Martin Fowler. Available: https://martinfowler.com/articles/microservices.html