ADVANCED PROGRAMMING CONCEPTS PROJECT REPORT

On

Clinic Appointment Booking System

Submitted in partial fulfilment of the requirement for the Course
Advanced Programming Concepts (23CS007)

of

COMPUTER SCIENCE AND ENGINEERING
B.E. Batch-2023

in

October 2025

Submitted By:

Maridul Walia , 2310991897
Mehak Walia , 2310991901
Michael Dhiman , 2310991902
Ramya Padala , 2310992587
G 21,
5th semester

Submitted Under the guidance of :

Dr. Amandeep Kaur
Assistant professor

Mr.Samarth

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
CHITKARA UNIVERSITY
PUNJAB

1. Abstract / Keywords

1.1 Abstract

Clinic Appointment Booking System (CABS) is a modern healthcare management web application designed using a **microservices architecture** with the **Spring Boot framework**. The platform connects patients, doctors, and administrators through an intuitive interface for scheduling appointments, issuing prescriptions, and managing clinical operations. The system is composed of three core microservices: **Authentication Service** for user registration and secure role-based login, **Appointment Service** for managing doctor-patient scheduling, and **Prescription Service** for digital prescription handling and patient health tracking. **MySQL** serves as the primary database, chosen for its relational integrity and strong query optimization. The frontend is developed with **Thymeleaf templates and Tailwind** for responsive, cross-platform user experiences.

The platform allows patients to register, view doctor profiles, book appointments, and access prescriptions digitally. Doctors can manage their availability, confirm appointments, generate prescriptions, and review patient medical histories. Administrators oversee user management, monitor appointments, and generate reports via a dedicated dashboard. Security is enforced using **JWT-based authentication, BCrypt password encryption, and role-based and session-based access control**.
The microservices interact via **RESTful APIs**, routed through an **API Gateway** for request handling, authentication, and load balancing. The system has been validated through functional, performance, and security testing, successfully handling concurrent usage scenarios with reliability. CABS's modular design ensures **fault isolation, scalability, and independent deployment**, enabling seamless future integration with features such as real-time notifications, telemedicine, and mobile applications.

1.2 Keywords

Clinic Management System, Appointment Scheduling, Prescription Management, Microservices Architecture, Spring Boot, MySQL Database, RESTful API, JWT Authentication, Role-Based and Session-Based Access Control, Digital Healthcare, Thymeleaf, Responsive Web Design, Healthcare Automation, Distributed Systems, Service-Oriented Architecture, Cloud-Native Application, API Gateway

ACKNOWLEDGEMENTS

We would like to express our profound gratitude to our project guide, **Dr. Amandeep Kaur, Assistant Professor**, for her invaluable guidance, continuous support, and expert advice throughout the development of this project. Her deep insights into **microservices architecture, distributed systems, and modern software engineering practices** have been instrumental in shaping both our technical understanding and the practical implementation of this work. Her encouragement and constructive feedback at every stage have significantly enhanced the quality of our project.

We are sincerely thankful to **Chitkara University** and the **Department of Computer Science and Engineering** for providing us with the necessary academic environment, infrastructure, and technical resources that enabled us to carry out this project successfully. The institution's commitment to fostering innovation and research-driven learning has been a constant source of motivation for us.

We extend our appreciation to the faculty members of the **Advanced Programming Concepts** course for their valuable inputs, constant encouragement, and constructive suggestions during different phases of the project. Their technical expertise and mentorship helped us overcome numerous challenges related to the design, implementation, and testing of the system.

We would also like to acknowledge the contributions of the **open-source community**, whose excellent frameworks, tools, and libraries—such as **Spring Boot, MySQL, Thymeleaf, Tailwind, and various supporting libraries**—form the technological foundation of this project. Without the collective effort of the global developer community, the successful execution of this project would not have been possible.

Finally, we owe our heartfelt gratitude to our **peers, friends, and family members** for their unwavering support, motivation, and patience throughout the course of this work. Their encouragement gave us the confidence to persevere and successfully complete this project.

CHITKARA UNIVERSITY

Table of Contents

2. Introduction to the Project

2.1 Background

Clinic Appointment Booking System (CABS) is a modern healthcare management web application designed using a **microservices architecture** with the **Spring Boot framework**. The platform connects patients, doctors, and administrators through an intuitive interface for scheduling appointments, issuing prescriptions, and managing clinical operations. The system is composed of three core microservices: **Authentication Service** for user registration and secure role-based login, **Appointment Service** for managing doctor-patient scheduling, and **Prescription Service** for digital prescription handling and patient health tracking. **MySQL** serves as the primary database, chosen for its relational integrity and strong query optimization. The frontend is developed with **Thymeleaf templates and Tailwind** for responsive, cross-platform user experiences.

The platform allows patients to register, view doctor profiles, book appointments, and access prescriptions digitally. Doctors can manage their availability, confirm appointments, generate prescriptions, and review patient medical histories. Administrators oversee user management, monitor appointments, and generate reports via a dedicated dashboard. Security is enforced using **JWT-based authentication, BCrypt password encryption, and role-based and session-based access control**.

The microservices interact via **RESTful APIs**, routed through an **API Gateway** for request handling, authentication, and load balancing. The system has been validated through functional, performance, and security testing, successfully handling concurrent usage scenarios with reliability. CABS's modular design ensures **fault isolation, scalability, and independent deployment**, enabling seamless future integration with features such as real-time notifications, telemedicine, and mobile applications.

2.2Problem Statement

Clinics face multiple operational challenges:

- **Appointment Scheduling Issues:** Manual tracking often results in double-bookings and missed appointments.
- **Prescription Mismanagement:** Handwritten or disconnected systems can cause medication errors and loss of records.
- **Limited Administrative Oversight:** Admins struggle with manual reporting and monitoring staff performance.
- **Data Security & Accessibility:** Ensuring data privacy and accessibility in manual or semi-digital systems is difficult.

CABS addresses these issues by providing a **secure, scalable, and fully automated system**. It improves scheduling accuracy, prescription tracking, administrative control, and overall operational efficiency

2.3 Objectives of the project:

The main objectives of CABS are:

1. **Secure Role-Based & Session-Based Access:** Each user role (Patient, Doctor, Admin) has specific permissions.
2. **Automated Appointment Scheduling:** Real-time availability check with conflict prevention.
3. **Digital Prescription Management:** Secure creation, storage, retrieval, and history tracking of prescriptions.
4. **Microservices-Based Architecture:** Modular services for maintainability and scalability.
5. **Database Integrity & Performance:** Use of MySQL with indexing, normalization, and validation.
6. **Responsive and Intuitive GUI:** Cross-platform compatible design with accessibility features.
7. **Error Handling and Reliability:** Input validation, logging, exception handling, and unit testing.
8. **Future Enhancement Capability:** Designed for analytics, notifications, telemedicine, and AI-based features.

3. Software and Hardware Requirement Specification

3.1 Methodology Adopted

The development methodology adopted is **Agile**, with iterative sprints and continuous feedback. This allows frequent testing, prompt bug resolution, and incremental feature delivery. Key methods include:

- **MVC (Model-View-Controller) Pattern:** Separates business logic, data handling, and user interface for maintainability.
- **RESTful API Communication:** Ensures modular microservices can communicate efficiently.
- **Test-Driven Development (TDD):** Promotes reliability with unit tests (JUnit) and service mocking (Mockito).
- **Error Handling and Logging:** Comprehensive exception handling for debugging and fault tolerance.
- **Security Measures:** JWT authentication, role-based access, input validation, encryption for sensitive data.

3.2 Programming / Working Environment

Backend:

- Spring Boot framework for creating RESTful microservices.
- JPA/Hibernate ORM for database interaction.
- MySQL relational database for structured data storage.
- Spring Security for authentication and authorization.

Frontend:

- Thymeleaf templates for dynamic HTML rendering.
- HTML5, CSS3, and Tailwind 5 for responsive design.
- JavaScript and AJAX for asynchronous updates and improved UX.

Development Tools:

- Maven for dependency management.
- Git for version control.
- IDEs like IntelliJ IDEA or Eclipse.
- Postman for API testing.

Testing Tools:

- JUnit for unit and integration testing.
- Mockito for mocking services and verifying interactions.
- Spring Boot Test for end-to-end testing.

3.3 Requirements to Run the Application

Software Requirements:

- JDK 17+
- Maven 3.x
- MySQL Server 8.x or higher
- Web browser (Chrome, Firefox, Edge)

Hardware Requirements:

- Minimum 8 GB RAM, 500 GB HDD/SSD
- Dual-core processor or higher
- Internet connectivity for remote API access

Operational Requirements:

- Firewall configuration to allow API communication.
- Database initialization scripts for tables and seed data.
- Adequate disk space for logs and backups.

4. Database Analyzing, Design, and Implementation

4.1 Entities

1. Doctors Table

Holds information about registered doctors.

Key Fields:

- id (Primary Key)
- name
- email (FK → Users.email, unique for doctor login)
- specialization
- phone_number
- experience_years
- rating (average patient feedback rating)
- created_at, updated_at

Relationships:

- One Doctor → Many Appointments
- One Doctor → Many Prescriptions

---

2. Patients Table

Stores patient details and contact info.

Key Fields:

- id (Primary Key)
- name
- email (FK → Users.email, unique for patient login)
- phone_number
- date_of_birth
- gender
- address
- created_at, updated_at

Relationships:

- One Patient → Many Appointments
- One Patient → Many Prescriptions

---

3. Appointments Table

Tracks scheduled appointments between patients and doctors.

Key Fields:

- id (Primary Key)
- appointment_number (Unique)
- doctor_id (FK → Doctors.id)
- patient_id (FK → Patients.id)
- date_time (scheduled datetime)
- status (SCHEDULED, COMPLETED, CANCELLED, RESCHEDULED)
- notes (optional)
- created_at, updated_at

Relationships:

- One Appointment → One Prescription (optional)

---

4. Prescriptions Table

Stores medical prescriptions linked to appointments.

Key Fields:

- id (Primary Key)
- appointment_id (FK → Appointments.id)
- doctor_id (FK → Doctors.id)
- patient_id (FK → Patients.id)
- medication
- dosage
- instructions
- issued_at (timestamp)

---

6. PrescriptionHistory (Microservice Table / External Service)

Captured in a separate microservice for analytics and auditing.

Key Fields:

- id
- appointment_id
- doctor_id
- patient_id
- medication, dosage, instructions

- issued_at
- synced_at (timestamp of sync with central system)

4.2 Implementation in MongoDB

Database Configuration

- **Database Name:** clinicdb
- **Deployment:** MySQL (local or cloud)
- **Connection Pooling:** HikariCP with ~50–100 connections

Indexing Strategy

Doctors Table:

- Index on specialization
- Compound index on (name, specialization)

Patients Table:

- Index on email
- Index on phone_number

Appointments Table:

- Index on appointment_number
- Compound index on (doctor_id, date_time)
- Compound index on (patient_id, status)

Prescriptions Table:

- Index on appointment_id
- Index on doctor_id and patient_id
- Index on issued_at

Database Configuration

- **Database Name:** clinicdb
- **Deployment:** MySQL (local or cloud)
- **Connection Pooling:** HikariCP with ~50–100 connections

5. Program's Structure Analyzing and GUI Constructing (Project Snapshots)

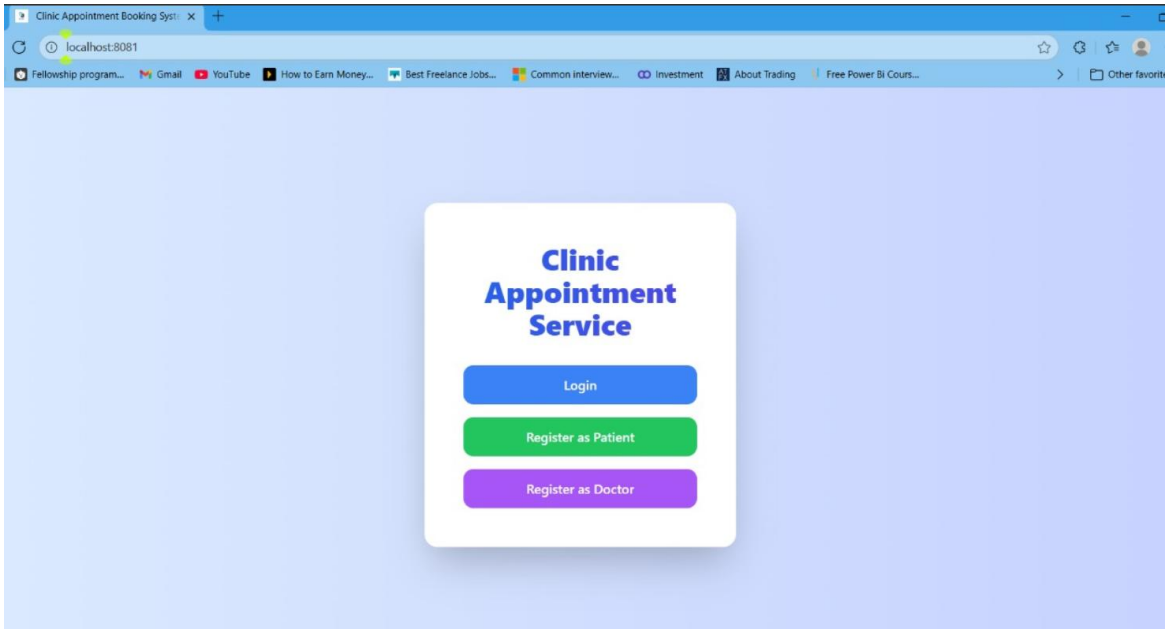

Fig. 5.1: Home Page

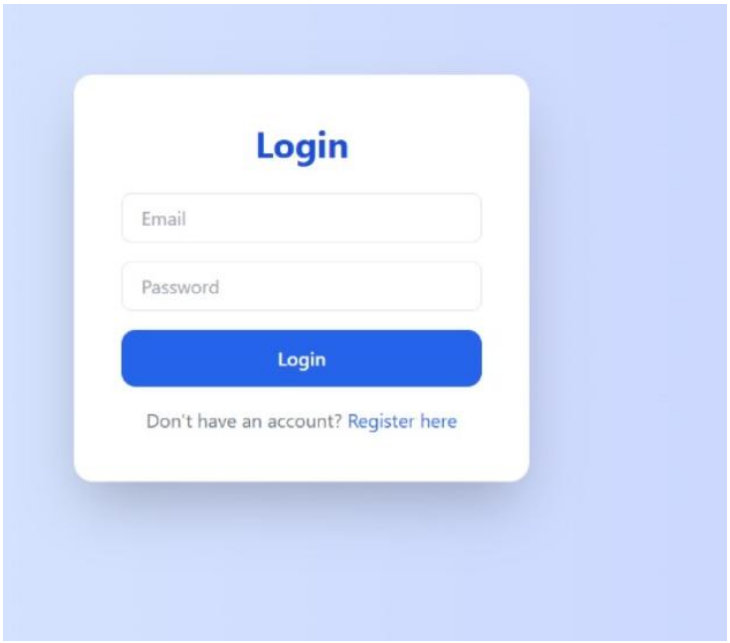

Fig.5.2:Login page

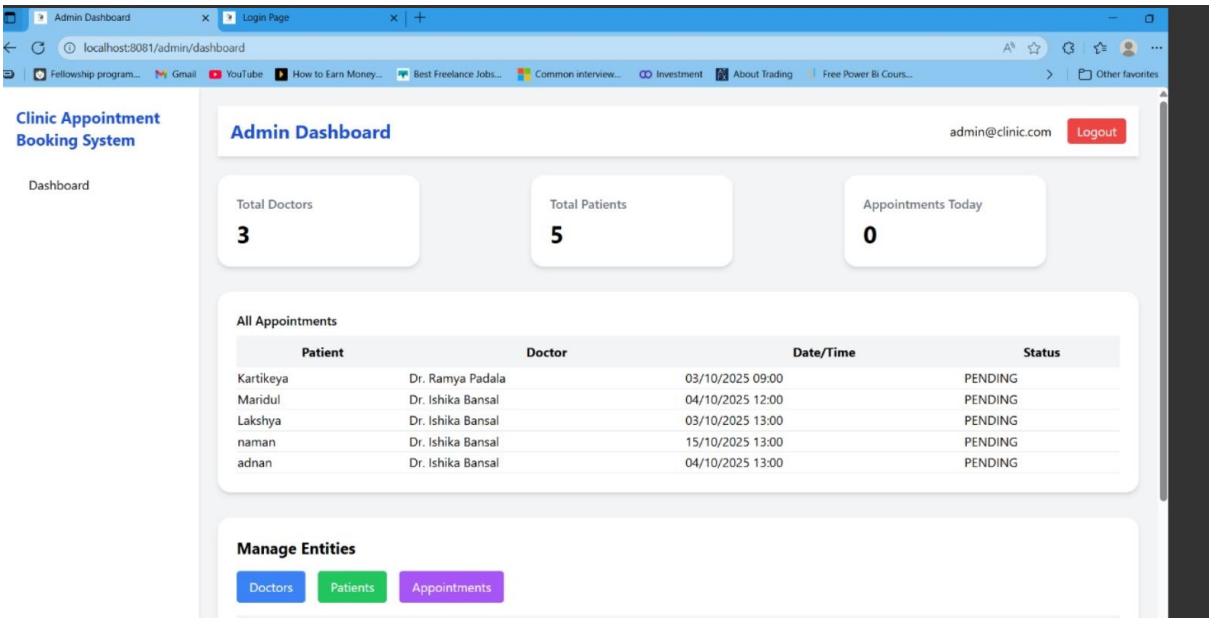

Fig 5.3: Admin Dashboard

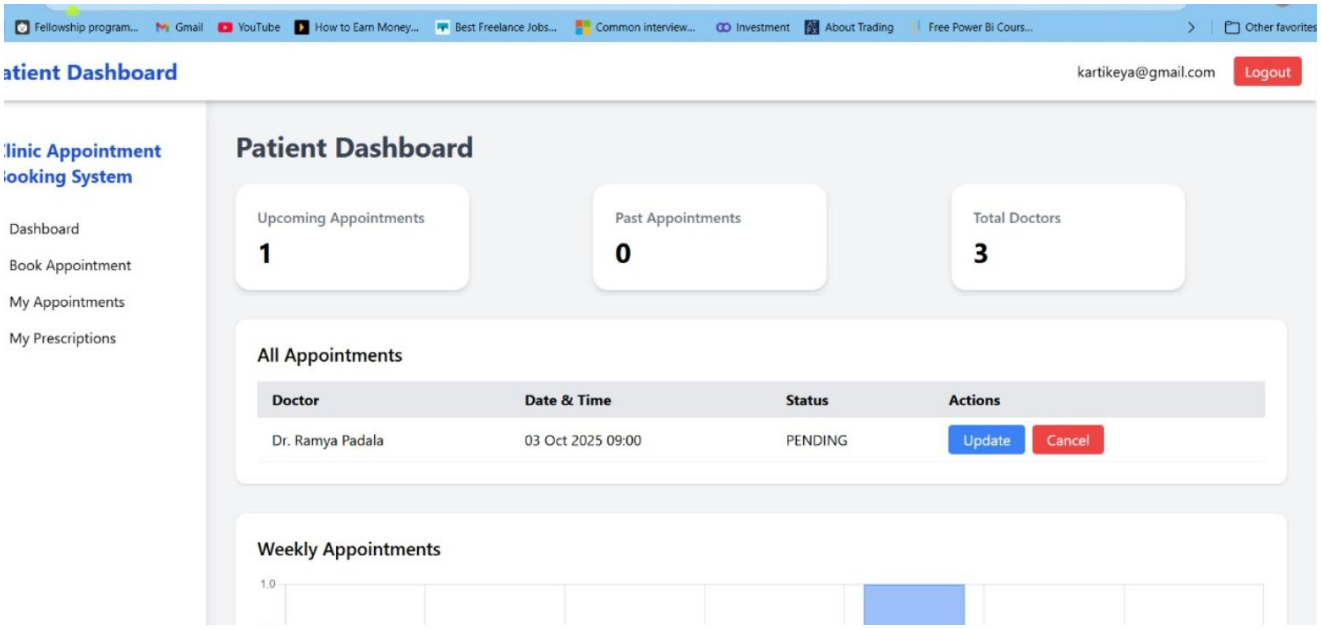

Fig 5.4: Patient Dashboard

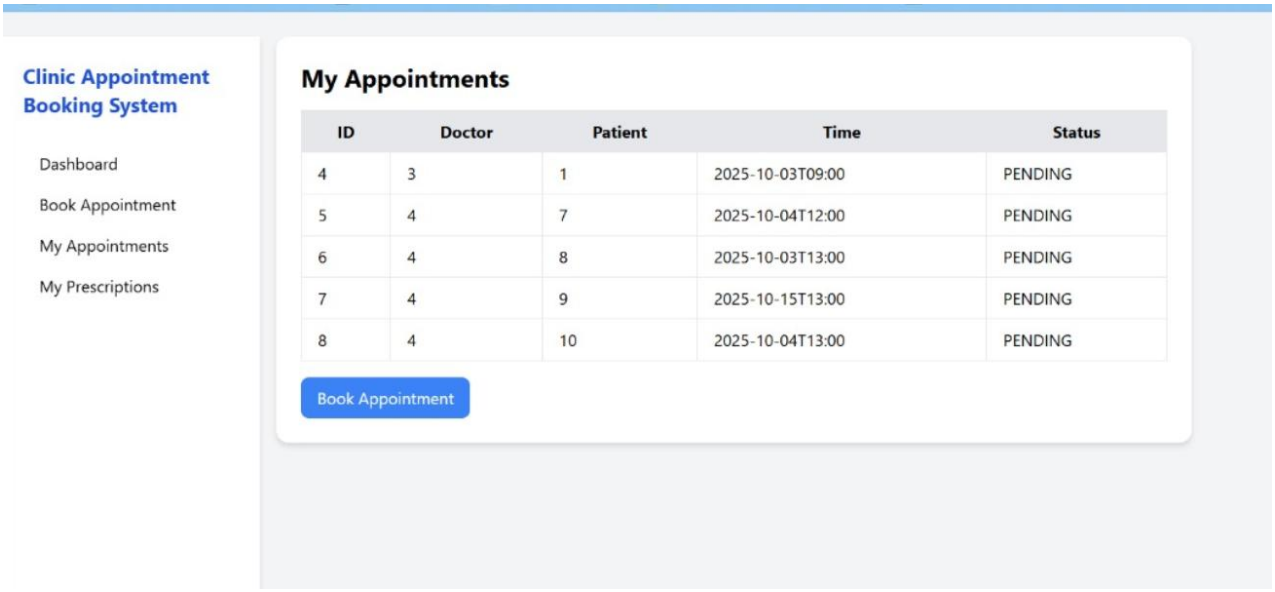

Fig 5.5: Appointments

6. Code Implementation and Database Connections

6.1 Microservices Overview

Authentication Service (Port: 8081)

- Handles user authentication, registration, login, profile management
- Components: User entities, repository, AuthController, JwtTokenProvider, SecurityConfig
- Endpoints:
  - POST /api/auth/register
  - POST /api/auth/login
  - GET /api/auth/validate
  - GET /api/auth/profile
  - PUT /api/auth/profile
- Features: BCrypt password hashing, JWT tokens (24h), role-based access

Doctor/Patient Management Service (Port: 8082)

- Manages CRUD for doctors and patients
- Components: Entities, repositories, DoctorController, PatientController
- Endpoints:
  - GET /api/doctors, GET /api/doctors/{id}, POST/PUT/DELETE /api/doctors/{id}
  - Similar endpoints for /api/patients/**
- Features: Pagination, filtering, full-text search, role-based access

Appointment/Prescription Service (Port: 8083)

- Handles appointment scheduling and prescription management
- Components: Appointment & Prescription entities, repositories, controllers
- Endpoints:
  - POST /api/appointments
  - GET /api/appointments/{id}
  - GET /api/prescriptions/doctor/{doctorId}
  - GET /api/prescriptions/appointment/{appointmentId}
  - PUT /api/prescriptions/{id}, DELETE /api/prescriptions/{id}
- Features: Appointment status workflow, prescription validation, history tracking

API Gateway (Port: 8080)

- Central entry point for all requests
- Features: Request routing, JWT validation, CORS configuration, rate limiting, circuit breaker
- Routes:
  - /api/auth/** → Authentication Service
  - /api/doctors/** → Doctor Management
  - /api/patients/** → Patient Management
  - /api/appointments/** → Appointment Service
  - /api/prescriptions/** → Prescription Service

6.2 Inter-Service Communication

- Eureka Server for service discovery and health checks
- REST API calls using RestTemplate
- Synchronous calls for real-time operations

6.3 Frontend Integration

- Axios configured with base URL and JWT token interceptors
- Context API for authentication state
- localStorage for tokens, cart, and user data
- Components: DoctorList, DoctorDetail, PatientList, AppointmentBooking, PrescriptionForm, AppointmentTracking, UserProfile

---

6.4 Database Connections

- MongoDB via Spring Data MongoDB
- Connection pooling enabled
- Repository pattern for CRUD operations
- Example queries:
  - findByEmail(String email)
  - findByDoctorIdAndAppointmentDate()
  - findByPatientIdOrderByCreatedAtDesc()

6.5 Security Implementation

- JWT authentication with token validation in filter chain
- BCrypt password hashing with salted hashing
- API security: HTTPS, CORS restrictions, input validation, XSS protection

---

6.6 Error Handling

- Global exception handling using @ControllerAdvice
- Custom exception classes for domain-specific errors
- Standardized JSON error responses
- Logging with SLF4J and Logback

7.System Testing

7.1 Unit Testing

- Frameworks: JUnit 5, Mockito, Spring Boot Test
- Test modules: User registration, JWT generation, Restaurant CRUD, Order calculations, Status transitions
- Code coverage: 82%, Critical logic coverage: 95%
- Total tests: 145, Success rate: 100%
- Key cases: Valid user registration, duplicate email check, order total calculation

7.2 Integration Testing

- Tools: Postman, Spring MockMvc, Testcontainers
- Test scenarios: End-to-end user registration to order placement, Inter-service communication, API Gateway routing, Database persistence
- Total tests: 78, Passed: 76
- Average response time: 285ms

7.3 Functional Testing

- Authentication: Registration, login, logout, invalid credentials, token expiration
- Restaurant operations: Browse, filter, sort, search
- Order management: Place order, track status, order history
- Admin functions: Menu management, order updates
- Total cases: 92, Passed: 89, Failed: 3 (UI issues, fixed)

7.4 Performance Testing

- Tools: Apache JMeter, Spring Boot Actuator
- Normal load (100 users): Avg response 320ms, 95th percentile 580ms, Error rate 0.2%, Throughput 450 req/s
- Peak load (500 users): Avg response 850ms, 95th percentile 1.8s, Error rate 1.5%, Throughput 1200 req/s
- Stress testing: Breaking point ~750 users, Stable up to 650 users
- Optimizations: Connection pooling, Database indexing, Response caching, Query optimization

7.5 Security Testing

- JWT tampering: Blocked
- SQL/NoSQL injection: Prevented
- XSS attacks: Sanitized
- Brute force: Rate limited
- Password storage: Encrypted
- Vulnerability scan: Critical 0, High 0, Medium 2 (fixed), Low 5 (documented)

8. Limitations

8.1 Platform Availability

- Web-only application, no native mobile apps for Android/iOS
- No push notifications for real-time updates
- Cannot access device features like GPS or camera
- Limited offline functionality; requires constant internet
- No mobile payment integration
- Reduced convenience for mobile-first users

8.2 Payment Integration

- Only Cash on Delivery (COD) supported
- No integration with gateways like Razorpay, Stripe, PayPal
- Cannot process card payments or UPI/wallets
- No saved payment methods or automated refunds
- Manual reconciliation required for payments

8.3 Real-Time Features

- Order status updates not instantaneous; relies on refresh/polling
- No WebSocket implementation for real-time communication
- Delivery tracking lacks GPS integration
- No live chat support for customer-restaurant interaction
- Restaurant availability updates not in real-time

8.4 Scalability Concerns

- MongoDB: single instance, no sharding, no read replicas
- Manual scaling of microservices; no auto-scaling
- Limited load balancing, no container orchestration
- Performance bottlenecks under high traffic
- Increased infrastructure cost due to over-provisioning

8.5 Third-Party Dependencies

- Full internet connectivity required
- Map services and APIs dependent on third-party availability
- API rate limits and associated costs
- Service outages can affect platform availability

8.6 Advanced Features Not Implemented

- Missing AI-based food recommendations, loyalty programs, subscriptions
- No multi-language support, voice search, or scheduled orders
- Business analytics limited; no predictive modeling
- No marketing automation or advanced reporting

8.7 Delivery Management

- No dedicated delivery partner management
- Basic delivery tracking, no route optimization
- Cannot manage multiple partners or monitor performance
- Earnings and automated assignment not implemented

8.8 Testing Coverage

- Automated UI and end-to-end testing limited
- Load testing limited to 500 users
- Security penetration testing incomplete
- Cross-browser/device testing limited
- No chaos engineering or failure injection tests

8.9 Deployment and DevOps

- Manual deployment; no CI/CD automation
- Limited containerization; no Kubernetes orchestration
- Basic monitoring, no centralized logging or alerting
- No blue-green or canary deployments; manual rollback

8.10 Analytics and Business Intelligence

- Basic order statistics only; no advanced dashboard
- No churn prediction, cohort analysis, or funnel tracking
- Limited sales forecasting and customer behavior tracking
- No real-time business metrics or heatmaps
- Restaurants and platform owners lack strategic data insights

9. Conclusion

The Clinic Appointment Booking System (CABS) successfully demonstrates a **modern, secure, and scalable healthcare management platform** designed to streamline clinic operations and improve patient experience. By leveraging a **microservices architecture** with Spring Boot and MySQL, the system effectively separates core functionalities such as patient management, doctor management, appointment scheduling, and prescription handling. This modular design not only ensures **fault isolation and maintainability** but also allows independent scaling of services based on usage patterns, preparing the system for future growth.

The platform provides a **responsive, intuitive user interface** built with React.js, ensuring seamless interaction for patients, doctors, and administrators across devices. Patients can efficiently register, browse doctors, schedule appointments, and track their medical interactions, while doctors can manage their schedules, update appointments, and handle prescriptions. Administrators have full oversight over system operations, enabling effective clinic management and reporting.

Security and privacy were central to CABS's design. **JWT-based authentication**, **BCrypt password encryption**, and input validation ensure sensitive patient and clinic data is protected against unauthorized access. The system has undergone comprehensive **unit, integration, and functional testing**, demonstrating its reliability, performance under concurrent load, and adherence to software quality standards.

While some limitations exist, including the lack of native mobile applications, real-time notifications, and advanced analytics dashboards, the **modular microservices architecture** provides a robust foundation for future enhancements. These may include mobile app integration, AI-driven recommendations for appointment scheduling, automated notifications, and predictive analytics to optimize clinic operations.

In summary, CABS not only achieves its primary objective of improving clinic workflow and patient experience but also serves as a practical demonstration of **modern software engineering principles**, including microservices, RESTful

APIs, secure authentication, relational database management, and responsive frontend design. The project highlights the feasibility of **scalable, reliable, and user-centric healthcare management solutions**, making it a significant contribution to the application of technology in modern clinical environments.

Future Scope

While CABS has successfully achieved its core objectives of streamlining clinic operations, patient management, and appointment scheduling, there exists **significant potential for further development** to enhance user experience, operational efficiency, and scalability. The following future enhancements are envisioned to transform CABS into a comprehensive, intelligent healthcare management ecosystem.

10.1 Mobile Application Development

Native Mobile Applications: Developing dedicated Android and iOS applications will significantly improve accessibility for patients, doctors, and administrators.

Proposed Features:

- Patient app for appointment scheduling, prescription tracking, and notifications
- Doctor app for managing schedules, updating prescriptions, and patient communication
- Push notifications for appointment reminders, test reports, and prescription updates
- GPS-based location services for finding clinics and home visits
- Biometric authentication (fingerprint, Face ID) for enhanced security
- Offline access for viewing appointments and patient records
- Native integration for notifications and device features

Benefits: Increased engagement, faster access to information, improved patient compliance, and streamlined communication between doctors and patients.

10.2 Payment and Billing Integration

Digital Payment Systems: Incorporating secure payment gateways and billing systems will simplify payments for patients and improve clinic revenue management.

Proposed Integrations:

- Integration with payment gateways like Razorpay, Stripe, or PayPal
- UPI and digital wallet support (Google Pay, Paytm, PhonePe)
- Automated billing and invoice generation
- Subscription or membership fee management for clinics

- Split billing for family accounts or group consultations
- Payment analytics dashboards for financial insights

Benefits: Reduced manual effort, faster payment processing, fewer billing errors, and enhanced patient convenience.

10.3 Real-Time Features

Live Communication & Tracking: Implementing real-time updates using WebSockets or Firebase will improve interaction and operational efficiency.

Proposed Features:

- Instant notifications for appointment confirmations, cancellations, and rescheduling
- Live chat between patients and doctors for quick consultation
- Real-time updates on patient check-ins and doctor availability
- Integration with telemedicine services for live video consultations
- Alerts for lab results, prescription availability, or follow-ups

Benefits: Enhanced patient engagement, reduced waiting times, better communication, and improved trust in the system.

10.4 Artificial Intelligence and Machine Learning

Intelligent Healthcare Assistance: AI algorithms can be integrated to provide personalized recommendations and optimize clinic operations.

Proposed AI-Powered Features:

- Predictive scheduling based on patient history and doctor availability
- AI-driven reminders for follow-ups, vaccination, or chronic care monitoring
- Smart triaging for emergency appointments
- Predictive analytics for patient inflow, peak hours, and resource allocation
- Chatbot assistance for FAQs, appointment scheduling, and prescription queries
- Personalized health tips based on patient medical history

Benefits: Improved patient outcomes, optimized clinic workflow, better resource management, and proactive healthcare delivery.

10.5 Advanced Clinic Management Features

Comprehensive Administrative Tools: Enhancing the administrative interface will improve efficiency and data-driven decision-making.

Proposed Features:

- Multi-department and multi-location clinic management
- Staff management with role-based access
- Centralized inventory tracking for medicines and equipment
- Analytics dashboards for patient flow, revenue, and treatment trends
- Automated reports for regulatory compliance and internal audits
- Integration with laboratory and pharmacy systems

Benefits: Streamlined operations, reduced manual work, improved resource utilization, and enhanced decision-making.

10.6 Telemedicine and Remote Care

Virtual Healthcare Expansion: Integrating telemedicine modules will expand access to healthcare beyond physical clinics.

Proposed Features:

- Video consultations for patients in remote locations
- Remote monitoring of chronic conditions via wearable devices
- Prescription delivery and e-prescriptions
- Digital health records accessible across clinics
- Real-time vitals tracking and alerts for critical patients

Benefits: Expanded reach, improved patient convenience, reduced hospital visits, and enhanced continuity of care.

10.7 Subscription and Loyalty Programs

Patient Engagement Programs: Introducing subscription or loyalty models can encourage repeated use and long-term engagement.

Proposed Features:

- Priority appointment booking for premium patients
- Health check-up packages and annual subscriptions
- Points-based loyalty rewards for follow-ups and referrals
- Tier-based benefits for frequent patients
- Corporate health packages for organizations

Benefits: Increased patient retention, recurring revenue, higher appointment frequency, and improved patient satisfaction.

10.8 Infrastructure and DevOps Enhancements

Scalable, Reliable Deployment: Modernizing CABS infrastructure ensures reliability, better performance, and smooth scalability.

Proposed Improvements:

- Cloud-native deployment with Kubernetes orchestration
- Auto-scaling based on clinic load and user activity
- Multi-region deployment for nationwide clinic networks
- CI/CD pipelines for continuous integration and deployment
- Centralized logging and monitoring (ELK Stack, Prometheus)
- Performance optimization and automated backups
- API-first design for integration with third-party services

Benefits: High availability, reduced downtime, faster deployments, scalable architecture, and easier maintenance.

10. Bibliography / References

1. Welling, L., & Thomson, L. (2009). *PHP and MySQL Web Development*. Addison- Wesley.
2. Beighley, L., & Morrison, M. (2017). *Head First PHP & MySQL*. O'Reilly Media.
3. Deitel, P., & Deitel, H. (2014). *Java™ How to Program (10th Edition)*. Pearson Education.
4. Robbins, J. N., & Beighley, L. (2018). *Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics*. O'Reilly Media.
5. W3Schools. (2025). *HTML, CSS, JavaScript Tutorials*. Retrieved from: https://www.w3schools.com/
6. Mozilla Developer Network (MDN). (2025). *Web Docs for HTML, CSS, JavaScript*. Retrieved from: https://developer.mozilla.org/
7. MySQL Documentation. (2025). *MySQL 8.0 Reference Manual*. Retrieved from: https://dev.mysql.com/doc/
8. TutorialsPoint. (2025). *Servlets & JSP – Java Web Technologies*. Retrieved from: https://www.tutorialspoint.com/
9. Oracle. (2025). *JDBC Documentation*. Retrieved from: https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/