Software Engineering (Sessional) Report

Course: CSE-356

CUET Hospital Management System

Team Members

Rahul Saha (ID: 2104085)

Ashraf-Ul-Islam (ID: 2104096)

Avishek Biswas (ID: 2104097)

Department of Computer Science and Engineering (CSE)

Chittagong University of Engineering & Technology (CUET)

Chattogram-4349, Bangladesh

Contents

List of Figures

List of Tables

Introduction

1.1 Goals and Objectives of the Project

The primary objectives of the CUET Hospital Management System are:

- **Enhanced Health Management:** Provide students with easy access to their medical conditions and records through this website, they can maintain digital E-booklets, eliminating the need for physical booklet maintenance. We have a special feature called advanced disease prediction to avoid any medical emergency using AI.
- **Improved Hospital Operations:** Improved administrative processes including better doctors appointment scheduling, and prescription generation to reduce manual workload and errors.
- **Accessibility and Convenience:** Enable students to access medical services and information anytime, anywhere through a responsive web platform, particularly beneficial for those living away from home.
- **Data Security and Organization:** Implement secure storage and organized management of sensitive medical information with proper access controls and data integrity measures.
- **Reduce time waste of the students:** By using this website students can reduce their time waste which they often do by waiting for a doctor without any appointment.

1.2 Scope of the Work

1.2.1 Current Situation and Context

Students at CUET come from various parts of Bangladesh and reside in university halls away from their families. This geographical distance creates significant challenges in maintaining proper healthcare, especially when balancing academic demands. Currently, the medical center operates with manual, paper-based systems for record-keeping, prescription management, and scheduling which should be made faster and dynamic.

The proposed CUET Hospital Management System aims to transform the operations by digitizing key processes. The system will manage student medical issues, doctor availability rosters, appointment scheduling, prescription generation, and E-booklet storage—all currently handled manually. This digital transformation will significantly reduce administrative burden, minimize errors, and provide students with convenient 24/7 access to their medical information.

1.2.2 Competing Products (Available in the Market)

Several hospital management systems exist in developed country's universities, offering various features for healthcare administration.

As Bangladesh is a populated country it is often challenging to maintain good health situations. CUET Hospital Management System is specifically designed for Bangladeshi university medical centers with features according to student needs, cost-effective implementation.

1.3 System Overview

The CUET Hospital Management System comprises three core modules designed to work seamlessly:

- **Doctor Appointment Management:** A digitalized roster system displays doctor availability and schedules, helping students plan their visits efficiently.
- **E-Booklet System:** Students can access digital E-booklets containing their complete medical history, prescriptions, diagnoses, and treatment records. This replaces physical booklets and provides instant access from any device.
- **Search and Filter Module:** Administrators can quickly locate relevant E-booklets and medical records using keyword search and advanced filtering options based on student ID, date, diagnosis, or treatment type.

The system follows a client-server architecture with React-based frontend, Node JS backend, and MongoDB database. It supports role-based access control with distinct interfaces for administrators and students, ensuring data security while maintaining ease of use.

1.4 Structure of the Document

This report is organized into three comprehensive chapters: Introduction provides an overview of the project, including goals, objectives, scope, and system architecture. Project Management details the organizational structure, process model, constraints, resource requirements, timeline, and impact analysis. Requirements Analysis and Design identifies key stakeholders, use cases, SRS, diagrams, and safety/security requirements.

Project Management

2.1 Project Organization

The project team consists of three members working collaboratively with defined responsibilities and regular coordination meetings. The team follows an agile-inspired approach with iterative development cycles and continuous stakeholder feedback.

2.1.1 Individual Contribution to the Project

Student: Rahul Saha (ID: 2104085)

- Backend development with Django framework
- Database schema design and implementation
- Documentation preparation and technical writing
- Frontend Development using Bootstrap

Student: Ashraf-Ul-Islam (ID: 2104096)

- Development using Bootstrap
- User interface design and responsive layout implementation
- Client-side validation and state management
- Backend development with Django and Integration of frontend components with backend

Student: Avishek Biswas (ID: 2104097)

- System architecture design and component diagram design
- Documentation preparation and technical writing
- Backend development and system designing

2.2 Process Model Used

2.2.1 Rationale for Choosing the Life Cycle Model

This project adopts the Prototyping Process Model, participatory software development methodology. The prototyping paradigm is particularly effective when initial requirements are not fully defined and stakeholder feedback is crucial for refinement.

The development follows these sequential phases:

1. **Communication:** Initial stakeholder meetings to understand objectives and gather preliminary requirements
2. **Quick Planning:** Rapid assessment of scope, timeline, and resource allocation
3. **Modeling and Quick Design:** Focus on visible user interface elements and core functionality with feasibility.
4. **Construction of Prototype:** Development of working prototype demonstrating key features
5. **Deployment, Delivery, and Feedback:** Stakeholder evaluation and iterative refinement based on feedback

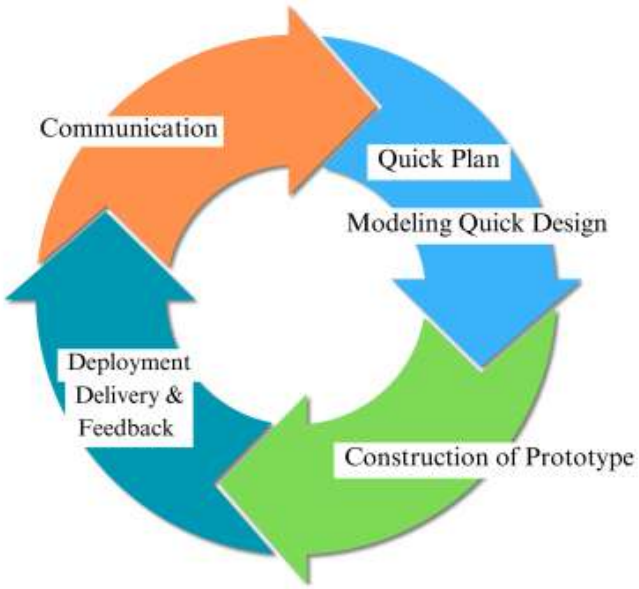


Figure 2.1: Prototyping Process Model

2.3 Constraints to Project Implementation

2.3.1 Schedule Constraints

The project timeline is constrained by academic semester deadlines. Multiple prototyping iterations may extend development time as stakeholder feedback incorporation requires additional development cycles.

2.3.2 Hardware Constraints

Efficient development requires computers with adequate processing power and graphics capabilities. Limited availability of high-specification hardware may affect development speed and the ability to test system performance under realistic load conditions.

2.3.3 Technical Constraints

Team members' varying experience levels with React, Node js, and MongoDB development create learning curves. Limited access to experienced mentors for technical guidance may slow problem resolution. Additionally, ensuring cross-browser compatibility and mobile responsiveness requires extensive testing resources.

2.4 Hardware and Software Resource (Tools/Language) Requirements

2.4.1 Hardware Requirements

A fast personal computer is essential for the development process. This includes a computer with sufficient processing power and memory to handle the demands of coding and testing.

Table 2.1: Hardware Requirements

Component	**Specification**
Processor	Intel Core i5 (8th Gen) or equivalent, minimum 2.5 GHz
RAM	Minimum 8GB DDR4 (16GB recommended)
Storage	256GB SSD for development, 512GB recommended
Graphics	Integrated graphics sufficient; dedicated GPU optional
Network	Broadband internet connection (minimum 10 Mbps)
Display	1920x1080 resolution minimum for design work

2.4.2 Software Requirements

Language: In front-end Development, we will use Bootstrap for building user interfaces, Django will serve as our back-end framework, facilitating the server-side logic and database management. the project will utilize PostgreSQL as the database management system to store and manage data efficiently.

Table 2.2: Software Requirements

Category	**Tool/Technology**	**Purpose**
Operating System	Windows 11,	Development environment
Frontend	Bootstrap	User interface development
Backend	Django	For connecting frontend and database
Database	PostgreSQL	Data storage and management
IDE	VS Code,	Code development and debugging
Version Control	Git, GitHub	Source code management
API Testing	Postman	API endpoint testing
Browser	Chrome	Testing and debugging

2.5 Project Timeline and Schedule

The project timeline and schedule are shown in Figure 2.2 through a Gantt Chart representation.

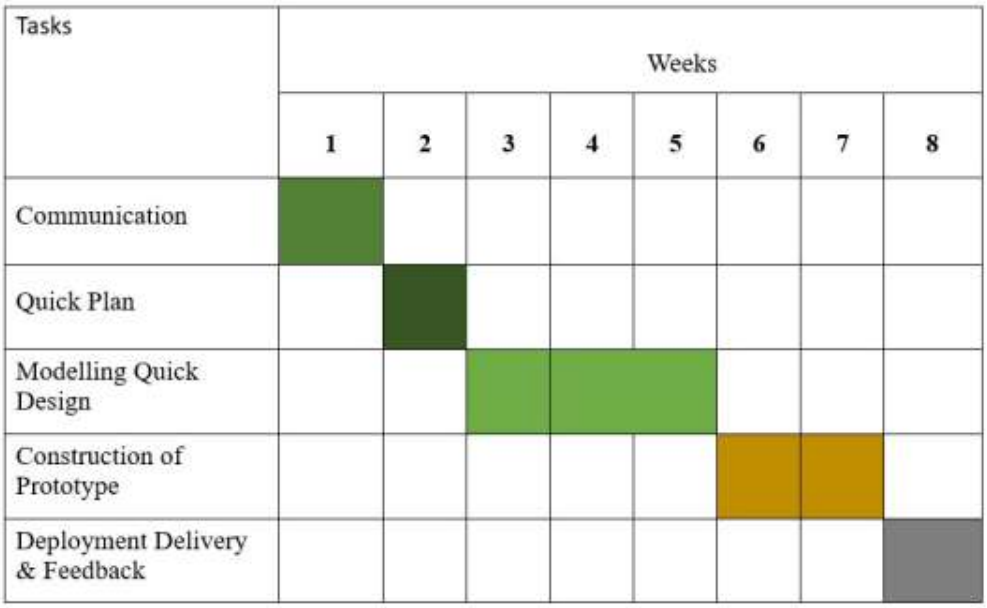


Figure 2.2: Project Timeline & Schedule Gantt Chart

2.6 Social, Cultural, and Environmental Impact of the Project

2.6.1 Social Impact

The CUET Hospital Management System significantly enhances student welfare by providing convenient, 24/7 access to medical services and health information. Students can easily schedule appointments, view their medical history, and receive timely treatment without administrative delays.

2.6.2 Cultural Impact

Implementing digital E-booklets represents a cultural shift toward modern healthcare practices aligned with contemporary digital literacy. This transition promotes a more organized, efficient approach to medical record management while familiarizing students with digital health systems.

2.6.3 Environmental Impact

By digitalizing medical records, prescriptions, and administrative processes, the system substantially reduces paper consumption, printing requirements, and physical storage needs. This transition minimizes waste generation and supports environmental sustainability.

Requirements Analysis and Design

3.1 Stakeholders of the System

3.1.1 System Administrators

System administrators are responsible for maintaining operational integrity and security. Their duties include:

- Managing user accounts and access permissions
- Creating and updating E-booklets with medical records
- Maintaining doctor roster schedules and availability
- Monitoring system performance and security
- Generating reports for medical center management
- Verifying student registration requests

3.1.2 Students

Students are the primary end-users who utilize the system to manage their healthcare needs. They can:

- Access personal medical history and E-booklets
- View current and historical prescriptions
- Check doctor availability and schedules
- Confirm receipt of prescribed medications
- Update personal contact information
- Request emergency medical services if necessary

3.1.3 Developers

Developers design, implement, and maintain the system. Their responsibilities include:

- Designing user-friendly interfaces
- Implementing backend logic and database operations
- Conducting rigorous testing and quality assurance
- Fixing bugs and implementing new features

- Ensuring system security and performance optimization
- Providing technical documentation and support

3.2 Use Case Diagram – Graphical and Textual Description

3.2.1 Use Case Diagram

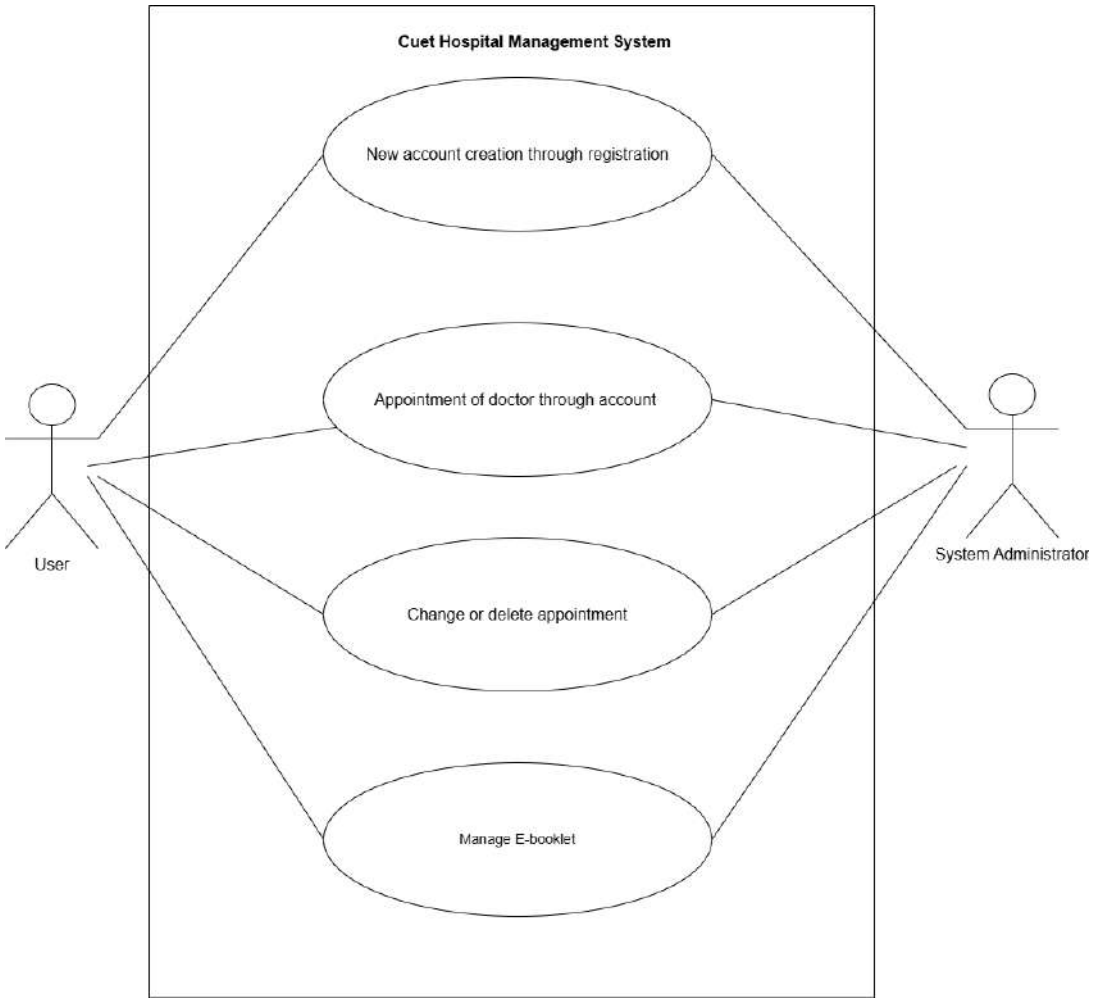


Figure 3.1: Use case diagram of CUET Hospital Management

3.2.2 Textual Description of Use Cases

Use Case 1: Student Registration

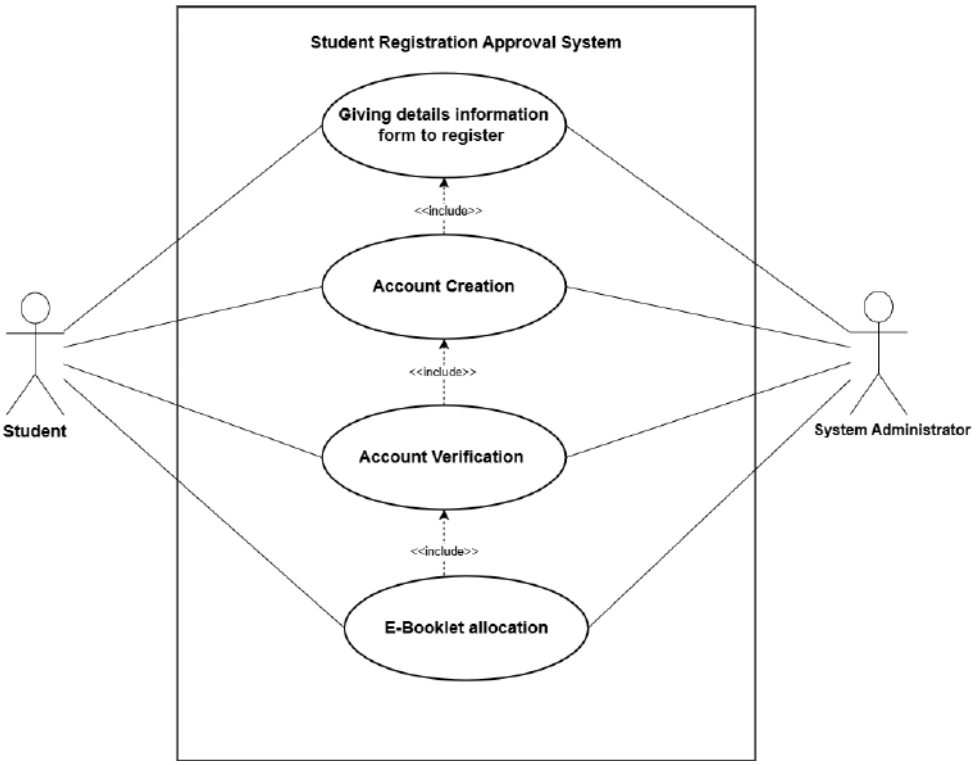


Figure 3.2: Use case diagram of Student Registration

- **Actor:** Student
- **Description:** New student registers in the system to access medical services
- **Preconditions:** Student has valid university credentials
- **Post conditions:** Registration request submitted for admin approval
- **Main Flow:**
  1. Student navigates to registration page
  2. Student enters personal details (name, ID, contact, hall)
  3. System validates input data
  4. Student submits registration form
  5. System sends confirmation and awaits admin approval
- **Alternative Flows:** Invalid data causes validation errors with specific messages
- **Exception Flows:** Network failure displays retry option

Use Case 2: View E-Booklet

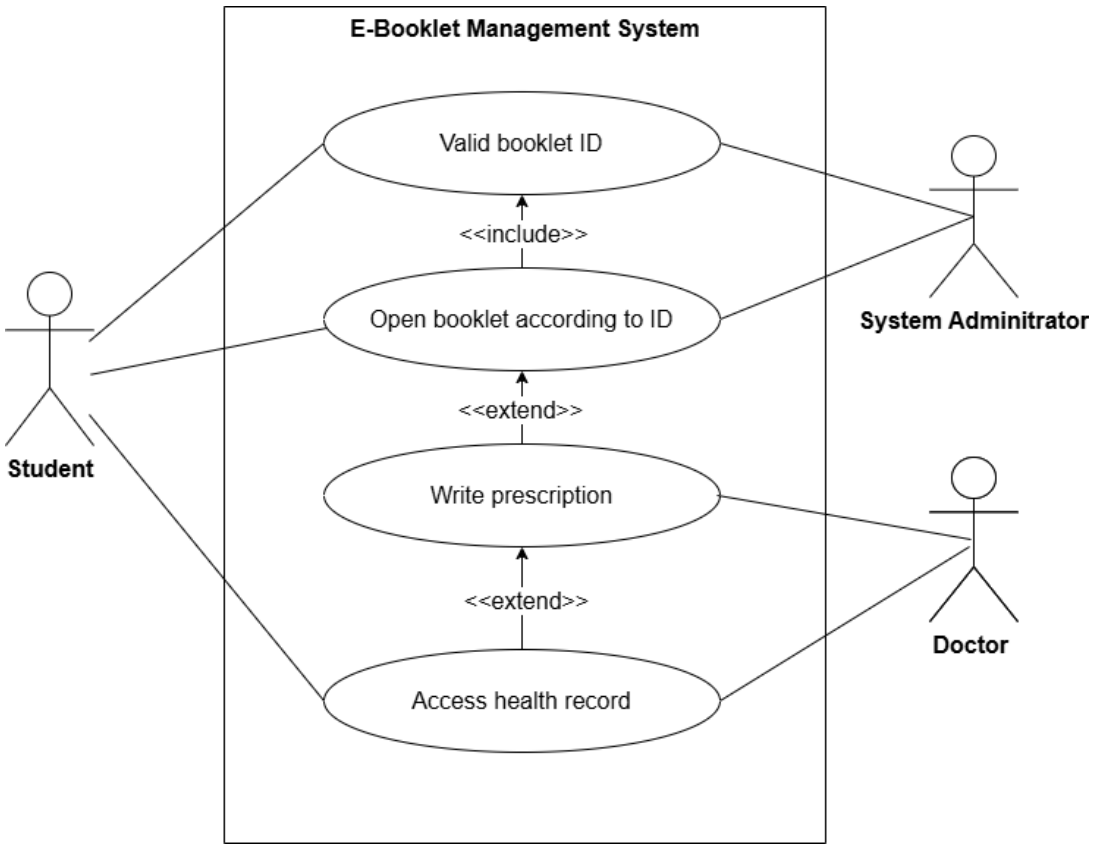


Figure 3.3: Use case diagram of E-Booklet Management

- **Actor:** Student
- **Description:** Student accesses their digital medical E-booklet
- **Preconditions:** Student is logged in and registration is approved
- **Post conditions:** E-booklet displayed with medical history
- **Main Flow:**
  1. Student logs into the system
  2. Student navigates to E-booklet section
  3. System retrieves student's medical records
  4. System displays E-booklet with prescriptions, diagnoses, and visits
  5. Student can filter by date or diagnosis type
- **Alternative Flows:** No medical history shows empty state message
- **Exception Flows:** Database error displays error message and retry option

3.3 Development of Software Requirements Specification (SRS)

3.3.1 Functional Requirements

E-Booklet Creation and Management

1. The system shall provide a form for administrators to create new E-booklet entries
2. Required fields shall include: student ID, doctor name, diagnosis, prescribed medications with dosage, investigation results, and clinical remarks
3. The system shall validate all input data for correctness and completeness
4. The system shall automatically timestamp each E-booklet entry
5. The system shall allow administrators to edit existing entries with audit trail
6. The system shall support attachment of investigation reports (PDF, images)

E-Booklet Access for Students

1. Students shall access E-booklets only after admin approves registration
2. The system shall display medical history in reverse chronological order
3. Students shall filter records by date range or diagnosis type
4. Students shall confirm medication receipt with timestamp
5. The system shall provide downloadable PDF version of E-booklet

3.3.2 Non-Functional Requirements

Performance

- System shall be accessible 24/7 with 99.5% uptime
- Page load time shall not exceed 3 seconds on standard broadband
- System shall support minimum 100 concurrent users without performance degradation
- Database queries shall execute within 500ms for typical operations

Security

- All passwords shall be hashed using industry-standard algorithms
- Access to E-booklet management shall be restricted to authorized administrators
- System shall maintain audit logs of all data access and modifications
- Session timeouts shall be implemented after 30 minutes of inactivity

3.4 Development of Use Case Template

CUET HOSPITAL MANAGEMENT SYSTEM

Student Registration (SR-REG)

- **Iteration:** 2, last modification: October 28 by Rahul Saha.
- **Primary actor:** Student.
- **Goal in context:** To register in the system and gain access to medical services and E-booklets from any location via the Internet.
- **Preconditions:** System must be fully configured; student must have valid CUET credentials.
- **Trigger:** The student decides to access medical services for the first time.

Scenario:

1. The student navigates to the CUET Hospital Management System website.
2. The student selects the "New Registration" button.
3. The student enters his or her student ID.
4. The student enters personal information (name, hall, department, contact details).
5. The student creates a password (must be at least eight characters in length).
6. The system validates student credentials with university database.
7. The system displays registration confirmation message.
8. The system sends registration request to administrator for approval.
9. The student receives temporary access pending admin approval.
10. The system stores student information in the database.

Exceptions:

1. Student ID is invalid or not recognized—see use case **Validate Student Credentials**.
2. Registration function not configured for this system—system displays appropriate error message; see use case **Configure Registration Function**.
3. Student selects "Login" instead—see use case **Student Login**.
4. Required information is missing or incomplete—display appropriate error message and see use case **Complete Registration Details**.
5. System database is unavailable—see use case **Database Connection Error**.

Priority:

- High priority, to be implemented as basic function.
- **When available:** First increment.
- **Frequency of use:** Frequent (by new students).
- **Channel to actor:** Via PC-based browser and Internet connection.

Secondary actors:

- System administrator, university database.

Channels to secondary actors:

- System administrator: Admin panel interface.
- University database: Secure API connection.

Open issues:

1. What mechanisms protect against unauthorized registration by non-CUET individuals?
2. Is the data validation sufficient against fake or incorrect information?
3. Will the system handle concurrent registration requests during peak admission periods?
4. How long should the admin approval process take, and what happens if de-

CUET HOSPITAL MANAGEMENT SYSTEM

View E-Booklet (VW-EBK)

- **Iteration:** 2, last modification: October 28 by Ashraf-Ul-Islam.
- **Primary actor:** Student.
- **Goal in context:** To access and view digital medical records, prescriptions, and treatment history from any remote location via the Internet.
- **Preconditions:** Student must be registered and approved; medical records must exist in the system.
- **Trigger:** The student needs to check medical history or show prescriptions.

Scenario:

1. The student logs onto the CUET Hospital Management System website.
2. The student enters his or her student ID.
3. The student enters the password.
4. The system displays all major function buttons.
5. The student selects the "E-Booklet" button from the major function buttons.
6. The student selects "View Medical History."
7. The system displays the complete medical record timeline.
8. The student selects a specific medical entry to view details.
9. The student selects the "View Prescriptions" button.
10. The system displays all prescribed medications with dosage and timing.
11. The system displays doctor's notes and recommendations.

Exceptions:

1. Student ID or password is incorrect or not recorded—see use case **Validate ID and Password**.
2. E-Booklet function not configured for this system—system displays appropriate error message; see use case **Configure E-Booklet Function**.
3. Student selects "Download Medical Report"—see use case **Download Medical Reports**.
4. No medical records are available or have not been entered—display appropriate error message and see use case **Enter Medical Records**.
5. An emergency medical condition is detected—see use case **Emergency Medical Alert**.

Priority:

- High priority, to be implemented as core function.
- **When available:** Second increment.
- **Frequency of use:** Moderate (when medical attention is needed).
- **Channel to actor:** Via PC-based browser and Internet connection.

Secondary actors:

- System administrator, doctors, medical staff.

Channels to secondary actors:

- System administrator: Admin panel interface.
- Doctors: Medical interface for record updates.
- Medical staff: Hospital management system.

Open issues:

1. What security measures protect sensitive medical information from unauthorized access?
2. Is the data privacy sufficient according to medical data protection standards?
3. Will system response time be acceptable when accessing large medical history files?

CUET HOSPITAL MANAGEMENT SYSTEM

CUET Hospital Management System

- **Iteration:** 1, last modification: November 15 by Avishek Biswas.
- **Primary actor:** Administrator.
- **Goal in context:** To manage and update doctor schedules, availability, and roster information in the system.
- **Preconditions:** Administrator must be logged in with appropriate privileges; doctor information must exist in the system.
- **Trigger:** Need to update doctor schedules, add new doctors, or modify existing roster information.

Scenario:

1. Administrator logs into the CUET Hospital Management System.
2. Administrator navigates to the "Doctor Management" section.
3. System displays current doctor roster and schedules.
4. Administrator selects "Manage Roster" option.
5. System shows interface for modifying doctor schedules.
6. Administrator updates availability times for specific doctors.
7. Administrator adds new doctors to the system with their schedules.
8. Administrator modifies existing doctor information if needed.
9. System validates the changes and updates the roster.
10. System notifies relevant parties of schedule changes.
11. Updated roster becomes immediately available for appointment booking.

Exceptions:

1. Schedule conflicts detected—system highlights overlapping schedules and prompts for resolution.
2. Invalid time slots entered—system validates and rejects impossible time ranges.
3. Doctor already has existing appointments—system warns administrator before making changes.
4. Database connection lost—system saves changes locally and syncs when connection restored.
5. Unauthorized access attempt—system logs the attempt and notifies security administrator.

Priority:

- High priority, essential for appointment system functionality.
- **When available:** First increment.
- **Frequency of use:** Regular (weekly updates and as needed).
- **Channel to actor:** Web-based admin interface.

Secondary actors:

- Doctors, Students, System database.

Channels to secondary actors:

- Doctors: Receive schedule updates and notifications.
- Students: Access updated availability for appointments.
- System database: Stores and retrieves roster information.

Open issues:

1. How to handle emergency schedule changes with existing appointments?
2. What is the maximum number of doctors the system can manage efficiently?
3. Should there be different roster templates for different days or seasons?
4. How to manage doctor leave and substitute arrangements?

3.5 Activity Diagram

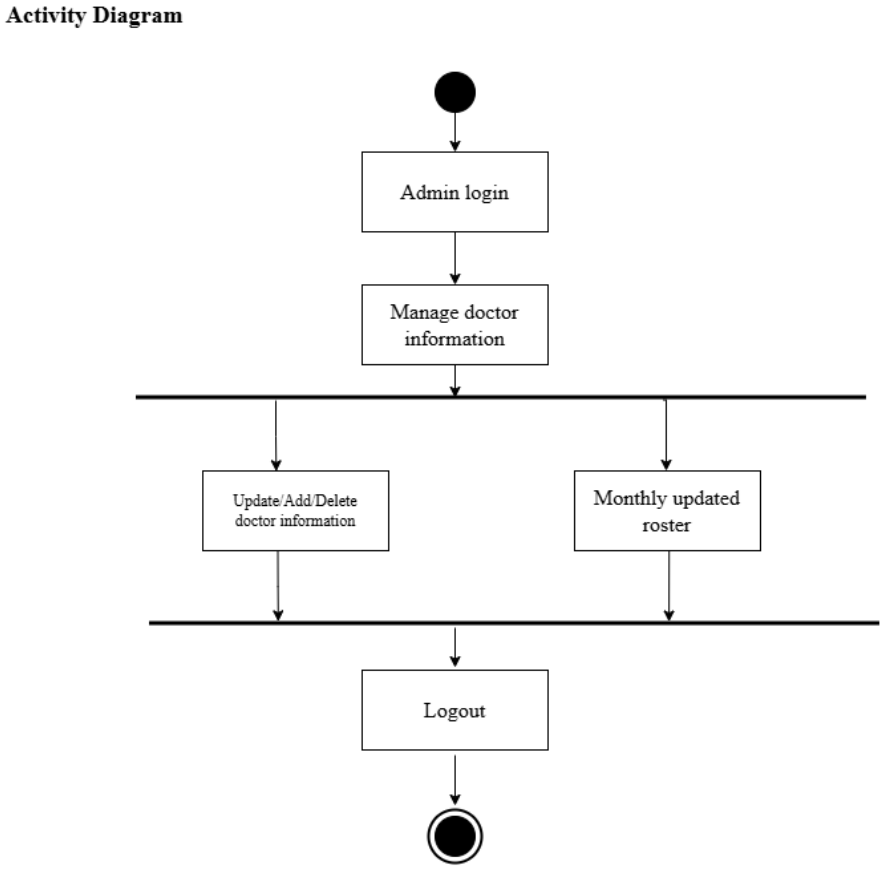


Figure 3.4: Activity Diagram Of Doctor Management

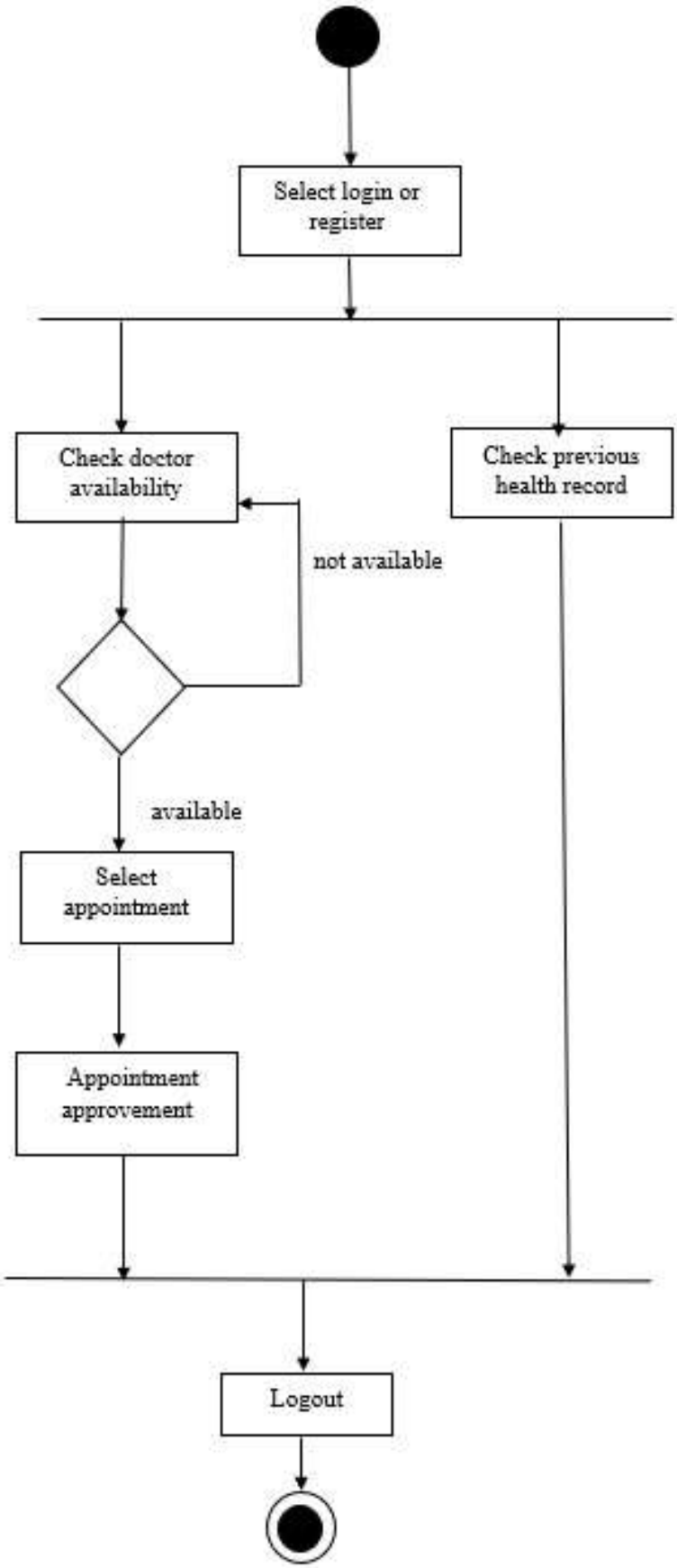


Figure 3.5: Activity Diagram Of Doctor appointment

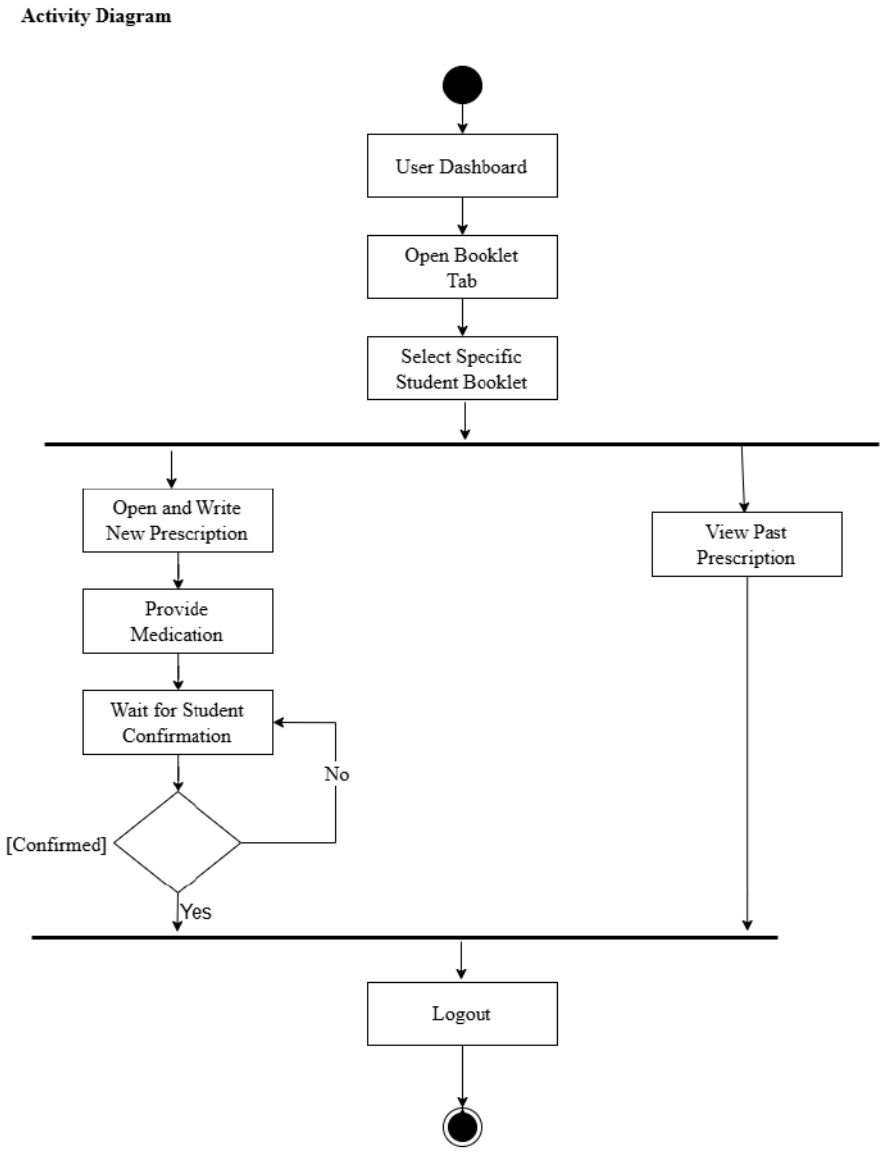


Figure 3.6: Activity Diagram Of E-booklet

3.6 Static Model – Class Diagram

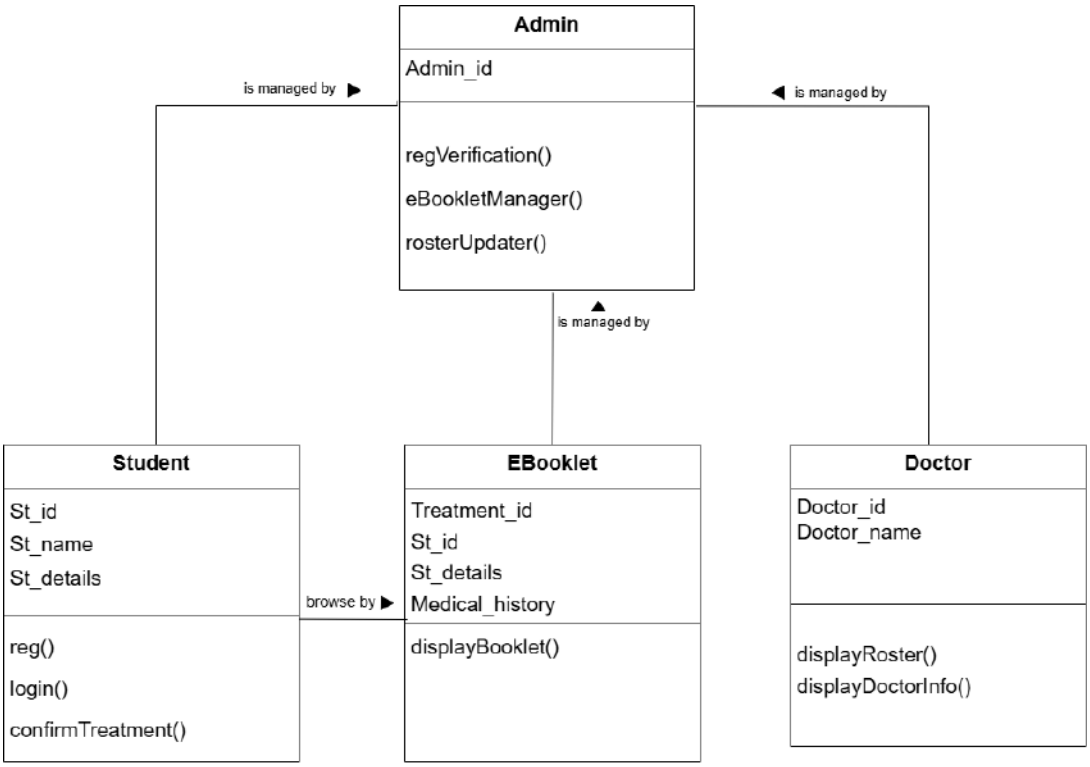


Figure 3.7: Class Diagram

3.6.1 Key Class Descriptions

Student

- **Attributes:** studentID (String), name (String), email (String), phoneNumber (String), hallName (String), registrationDate (Date), isApproved (Boolean)
- **Methods:** register(), login(), viewEBooklet(), confirmMedicationReceipt(), updateProfile()

3.7 Dynamic Model – Sequence Diagram

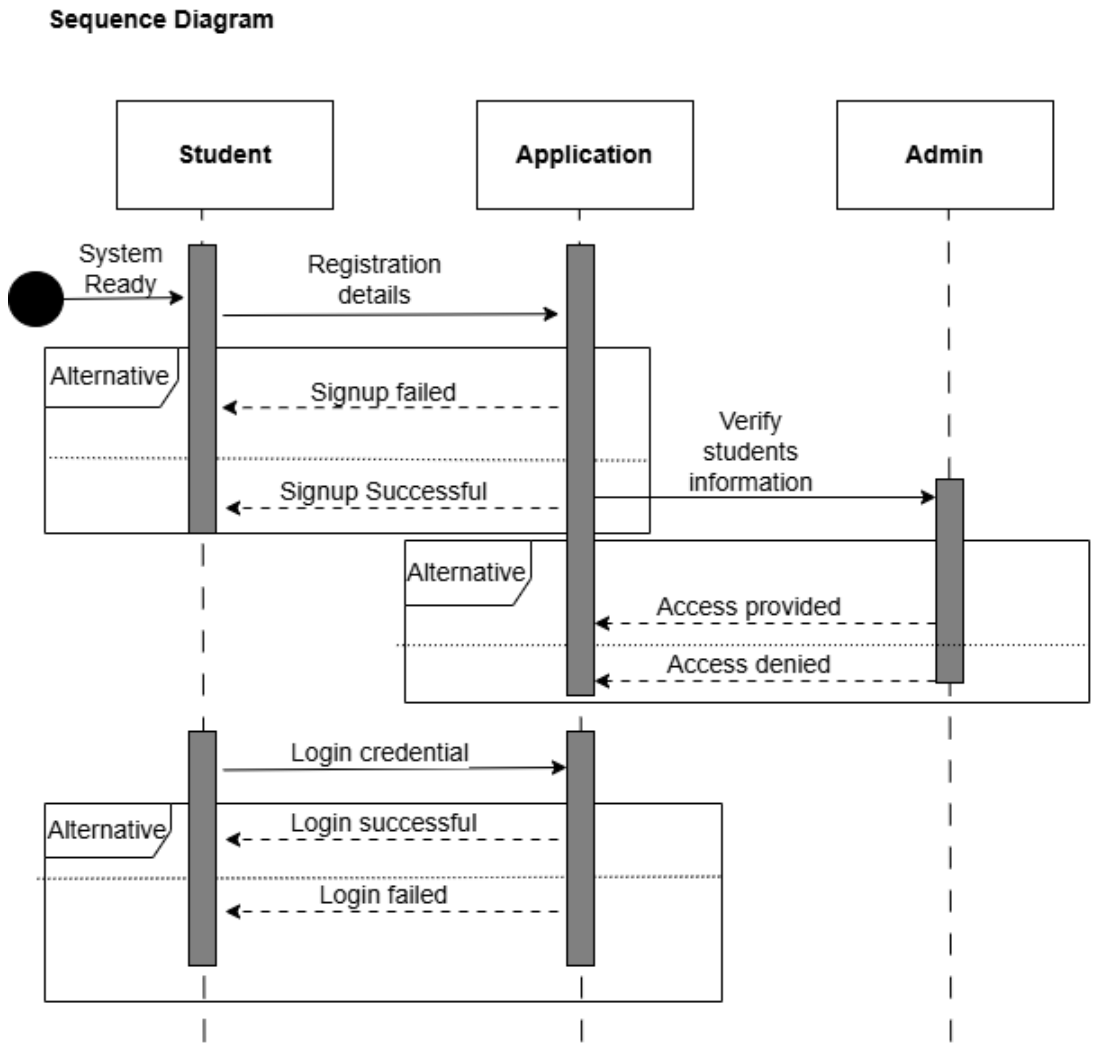


Figure 3.8: Sequence diagram for Student Registration and Approval System module

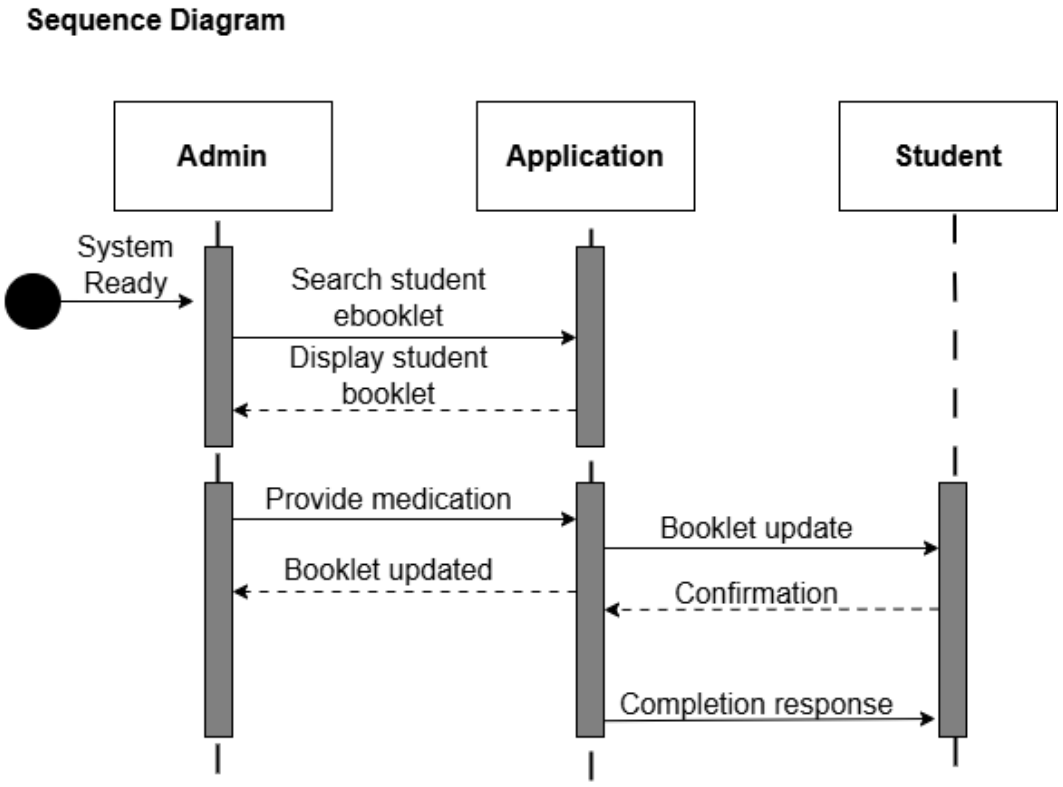


Figure 3.9: Sequence diagram for E-booklet Management System module

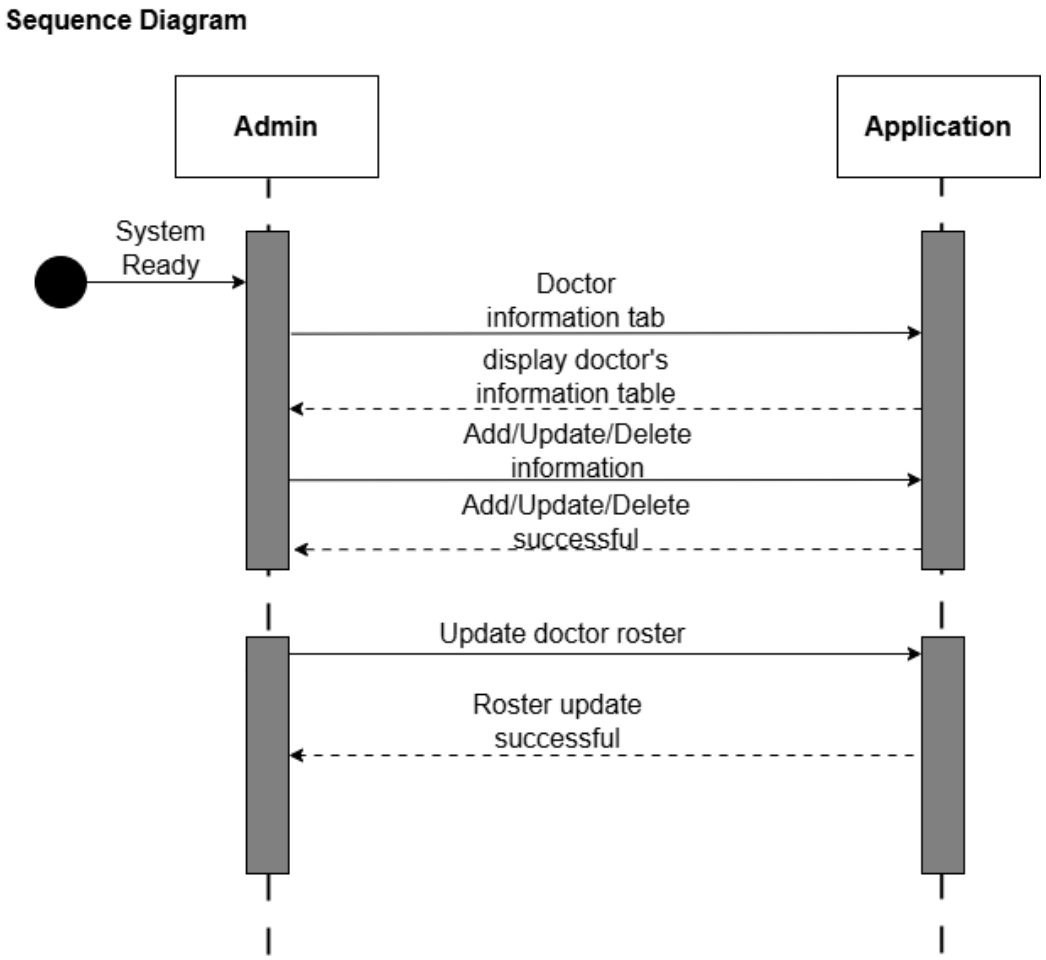


Figure 3.10: Sequence diagram for Manage Doctor Roster module

3.8 Safety and Security Requirements

3.8.1 Access Requirements

- Role-based access control for administrators and students
- Multi-factor authentication for sensitive operations

3.8.2 Integrity Requirements

- Data validation for all inputs and updates
- Regular database backups and integrity checks

3.8.3 Privacy Requirements

- Encryption of all sensitive data at rest and in transit
- Compliance with medical data privacy regulations

Architecture

4.1 Architectural model/style used

The architecture of the CUET Hospital Management System follows a client-server model with a Bootstrap-based frontend, Django backend, and PostgreSQL database for data management. This architecture ensures scalability, flexibility, and real-time data updates. An architectural style is a modification to a system's overall design, creating a structured framework for its components. It serves as both a descriptive mechanism and a blueprint for system construction, guiding its implementation. For the CUET Hospital Management System, a data-centered architectural design was utilized. In this architecture, the database is centrally positioned, frequently accessed by components performing operations like updating, adding, deleting, or modifying stored data.

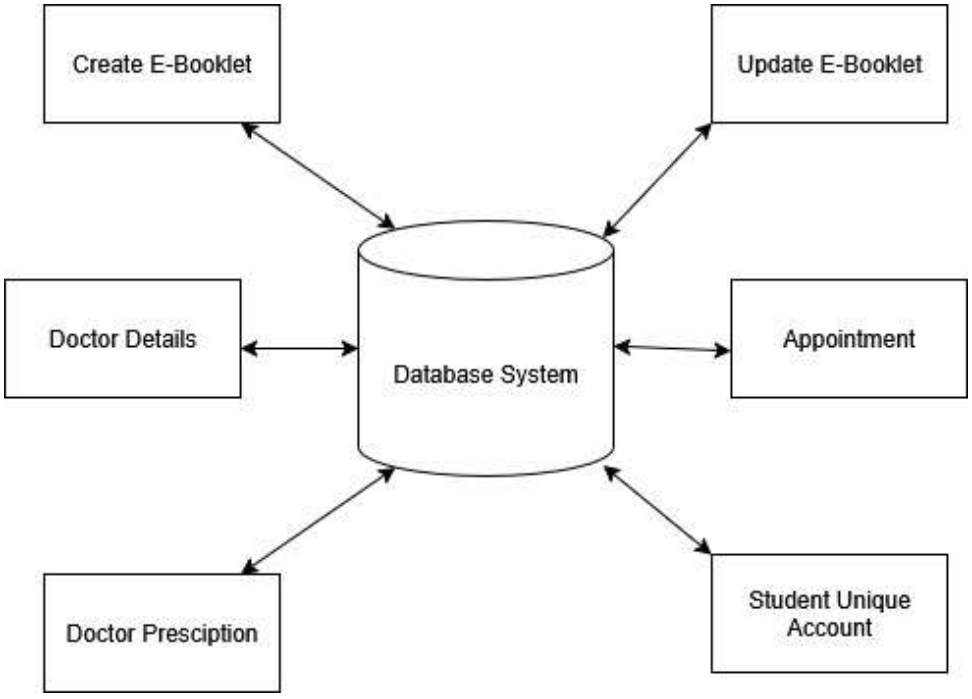


Figure 4.1: System Architecture Diagram

4.1.1 Rationale for choosing your architectural model/style

This architectural model was chosen for its modularity and ease of integration. The Bootstrap frontend provides a responsive and dynamic user interface, while Django allows asynchronous handling of requests, improving system performance. PostgreSQL database, is used to store large amounts of unstructured medical data efficiently. This combination supports efficient data retrieval and scalability, which is crucial for the

growth of the CUET Hospital Management System. Here are several important advantages of using DCA (Data Centered Architecture):

Scalability: DCA can grow or shrink according to the application's requirements. By increasing or decreasing the number of servers in the data center, the system can efficiently manage higher or lower volumes of traffic.

Blackboard Communication: Information can be exchanged between clients through a blackboard-style system, where a shared blackboard component manages and coordinates data transfer among them.

Ease of Integration: Existing components can be modified, and new client components can be introduced without affecting others, since each client operates independently within the architecture.

4.2 Technology, software, and hardware used

4.2.1 Software Technologies

- **Frontend:** Bootstrap is used to design responsive, visually appealing, and user-friendly interfaces. It provides a consistent layout framework, reusable UI components, and mobile-first styling that enhances the overall usability of the hospital management system.
- **Backend:** Django powers the server-side logic with its robust, secure, and scalable architecture. It supports rapid development, handles routing and request processing, and ensures smooth communication between the frontend, the database, and other system components.
- **Database:**SQLite is used for storing, managing, and retrieving medical data efficiently. Its reliability, ACID compliance, and strong querying capabilities make it ideal for handling patient records, appointments, and hospital workflow data securely.
- **Version Control:** Git and GitHub for version control and collaborative development.
- **API Testing:** Postman for testing and ensuring the functionality of REST APIs.

4.2.2 Hardware Requirements

- **Processor:** Intel Core i5 (8th Gen) or equivalent, minimum 2.5 GHz.
- **RAM:** Minimum 8GB DDR4, recommended 16GB.
- **Storage:** 256GB SSD for development; 512GB recommended for larger datasets.
- **Network:** Minimum 10 Mbps broadband internet for efficient collaboration and data transfer.
- **Display:** Minimum 1920x1080 resolution for optimal design and development.

Design

5.1 Component level design following pattern

A software component is an independent module that provides a specific, well-defined function and communicates through a clear interface. It is modular, portable, and designed to be easily reused or replaced when needed. In essence, it acts as a building block within a software system, offering targeted functionality through encapsulation and structured interactions with other components. The system follows a layered architecture pattern such as :

- **Presentation Layer:** This layer consists of the user interface built with Bootstrap and Material-UI, ensuring a responsive and user-friendly experience.
- **Application Layer:** This layer contains the business logic and API handling implemented in Django, connecting the frontend to the backend.
- **Data Layer:** SQLite serves as the data layer for storing medical records, student information, and other necessary data.

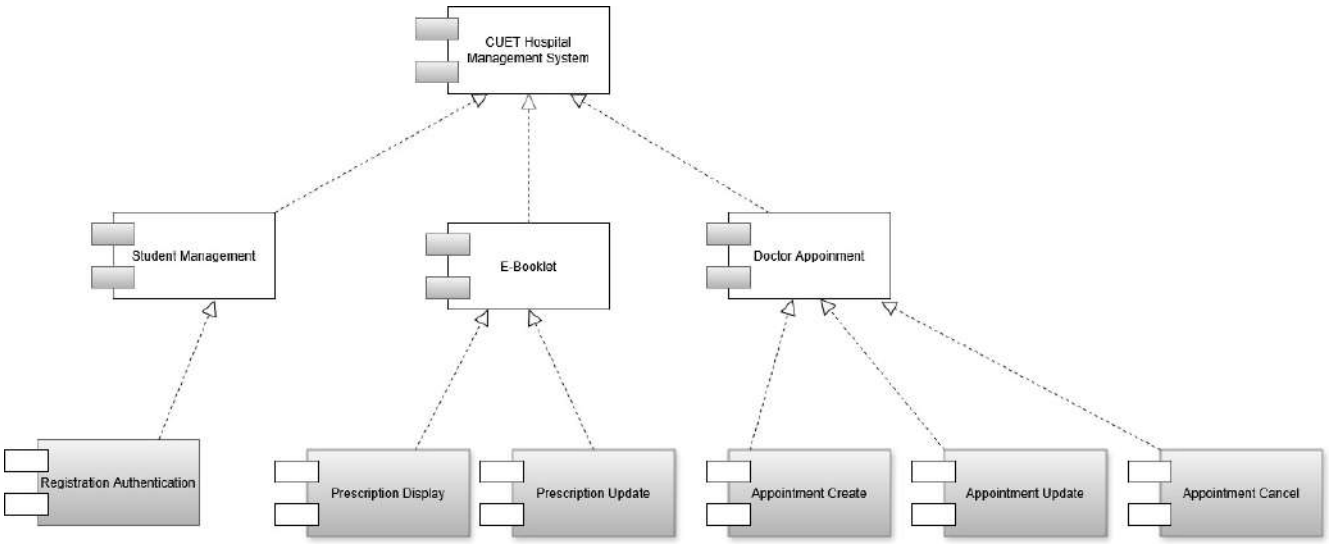


Figure 5.1: Structure chart for the CUET Hospital management System

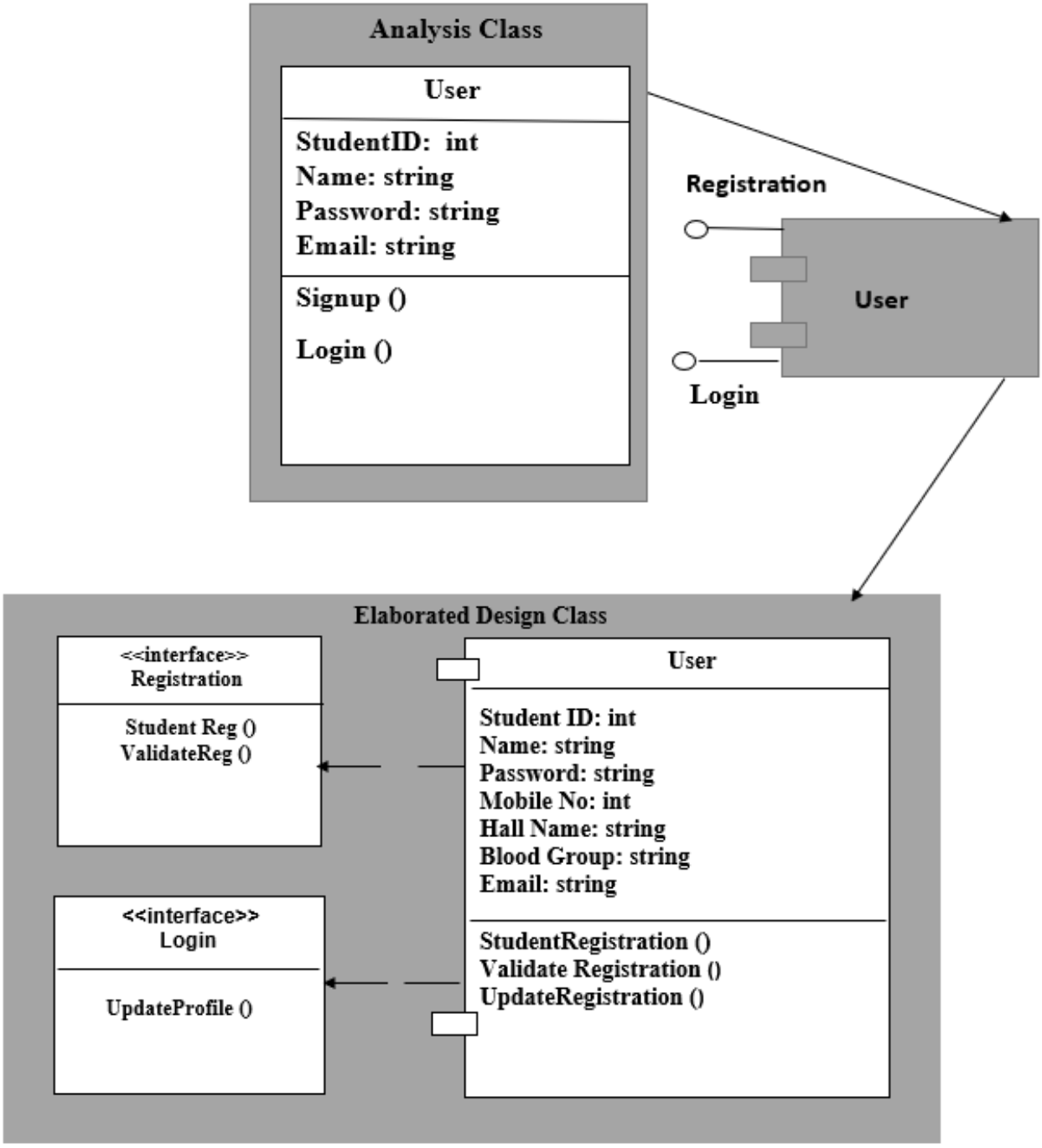


Figure 5.2: User component for the CUET Hospital management System

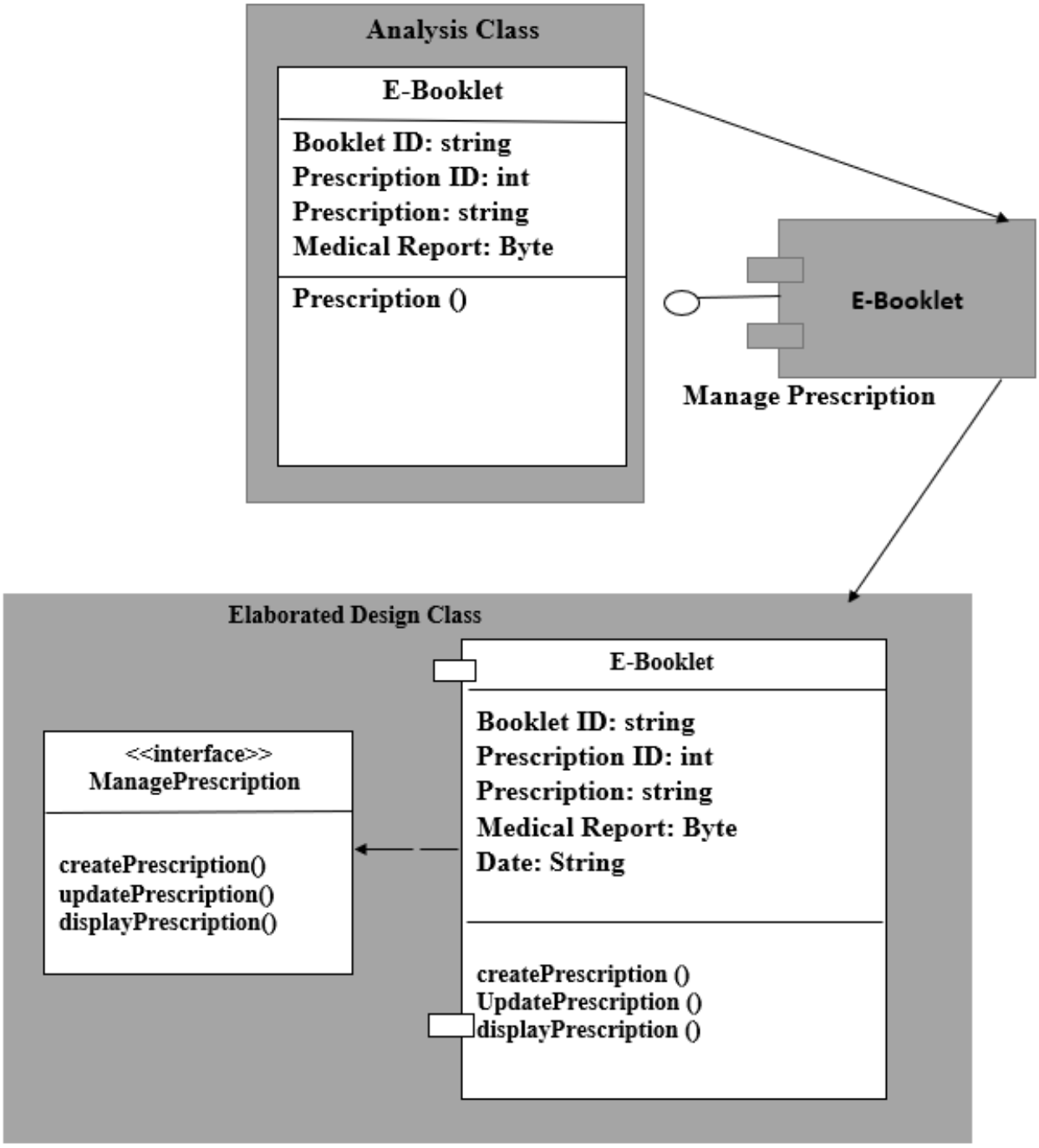


Figure 5.3: E-booklet Component of CUET Hospital management System

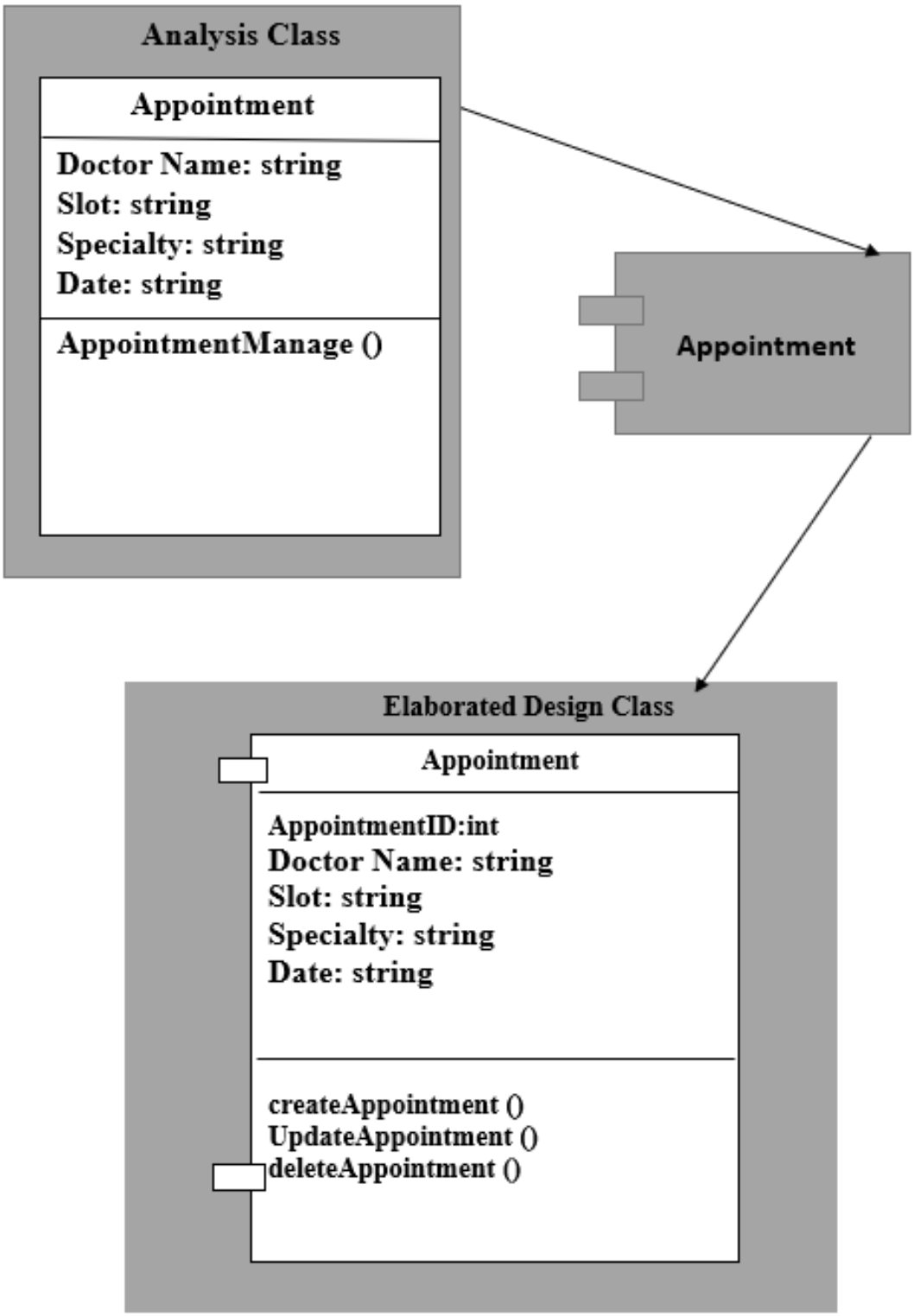


Figure 5.4: Appointment Component of Hospital management System

5.2 UI Design

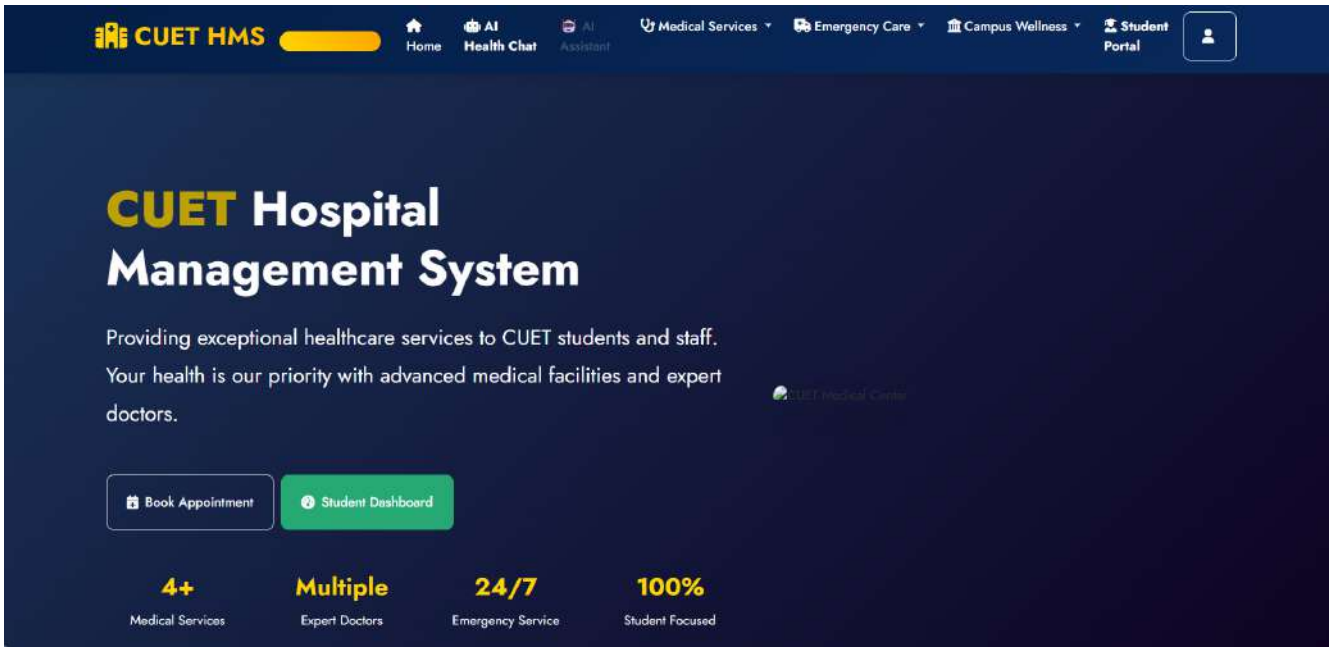


Figure 5.5: Homepage of CUET Hospital management System

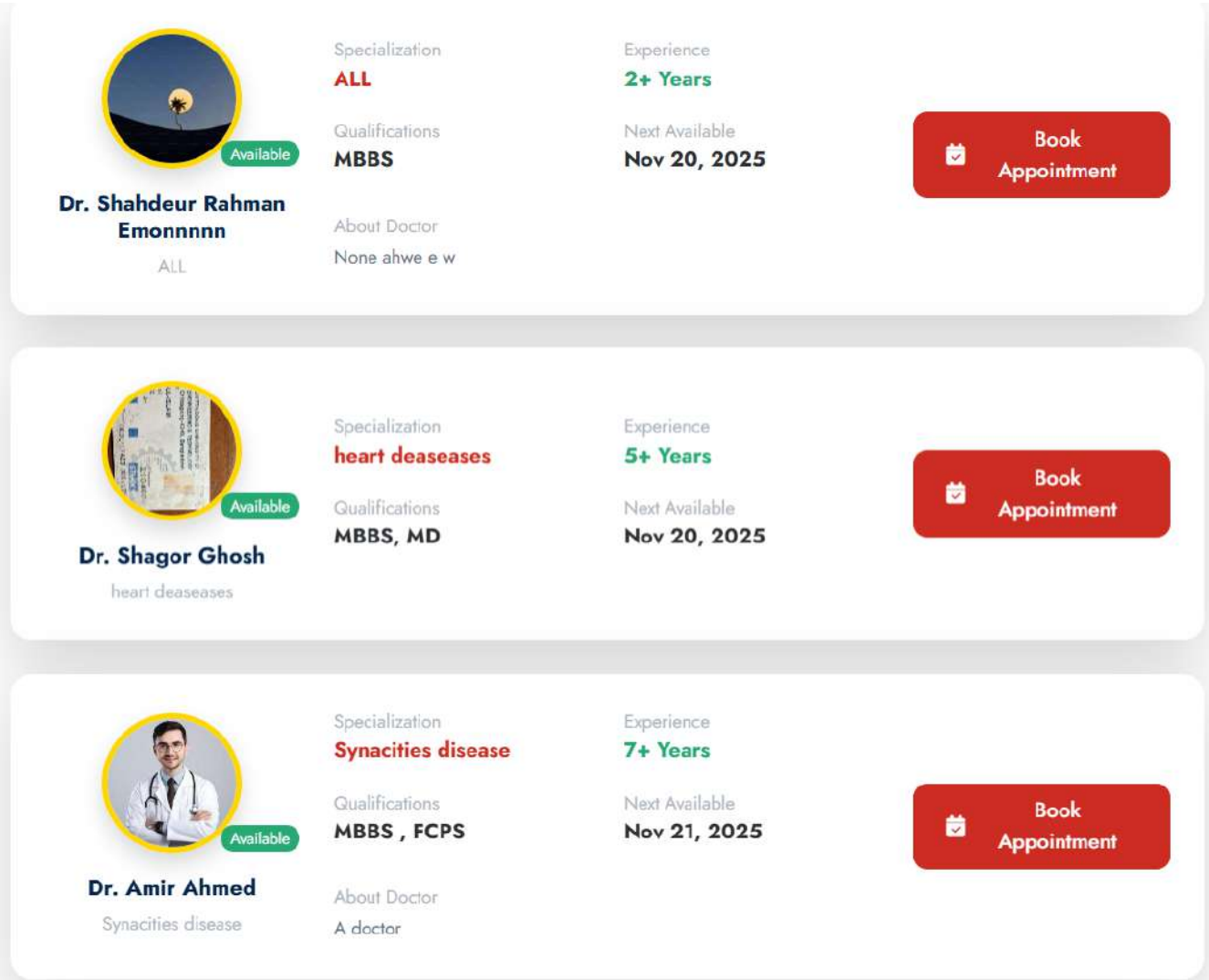


Figure 5.6: Doctor Appointment in student Dashboard of CUET Hospital management System

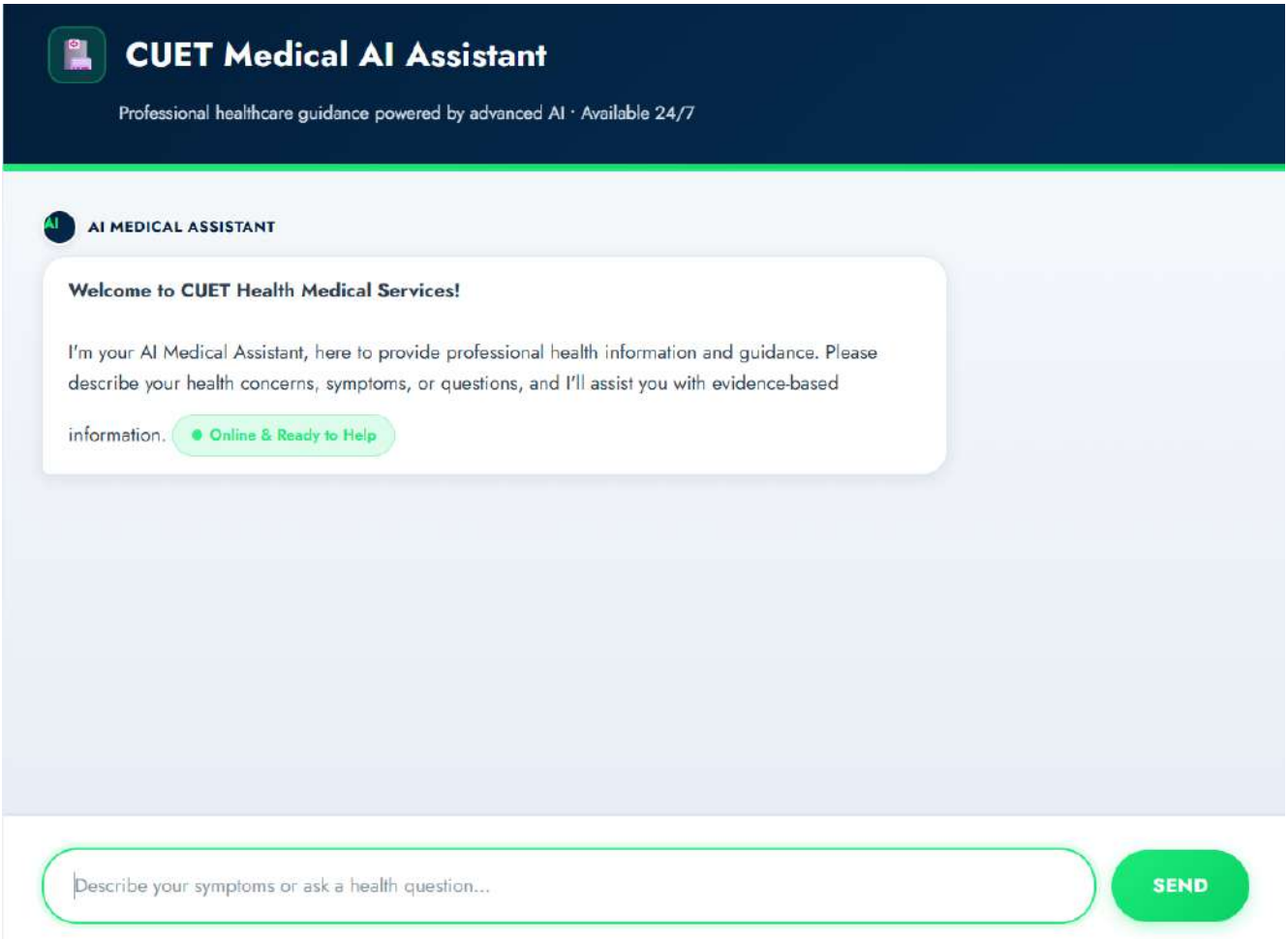


Figure 5.7: AI chatbot of CUET Hospital management System

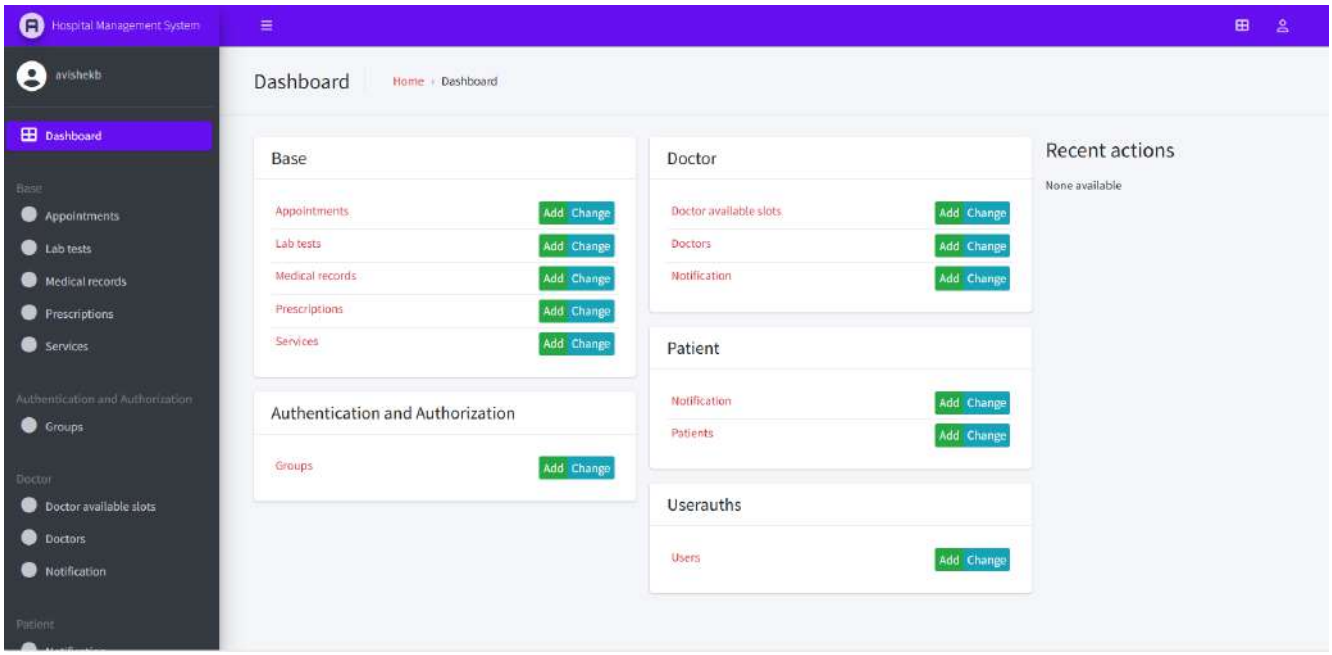


Figure 5.8: Admin Dashboard of CUET Hospital management System

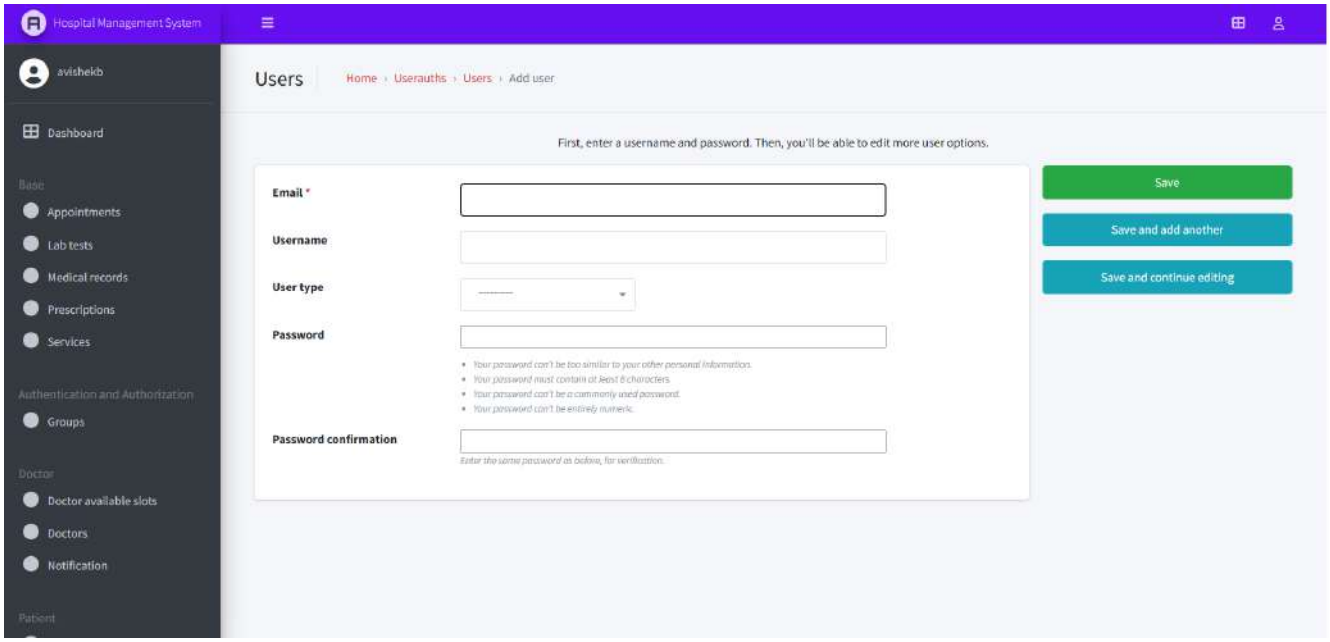


Figure 5.9: Patient or Doctor adding by admin of CUET Hospital management System

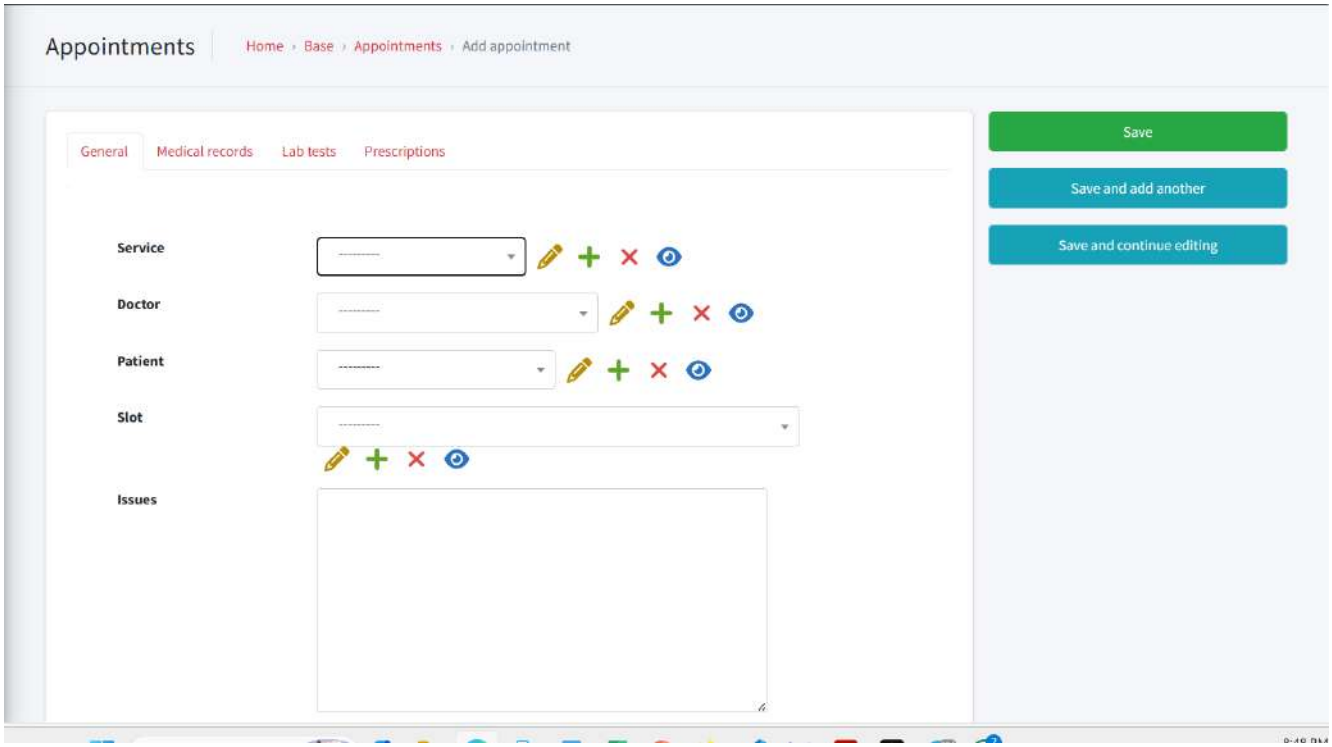


Figure 5.10: Add appointment by admin of CUET Hospital management System

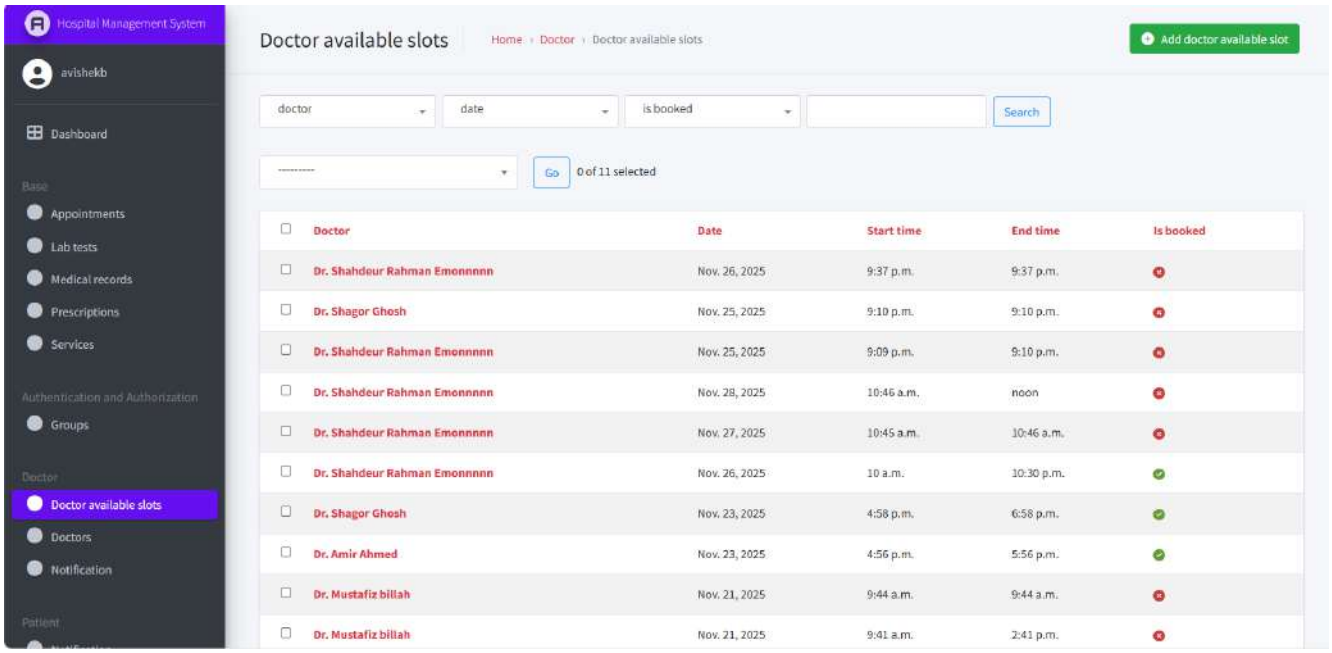


Figure 5.11: Doctor slot by admin of CUET Hospital management System

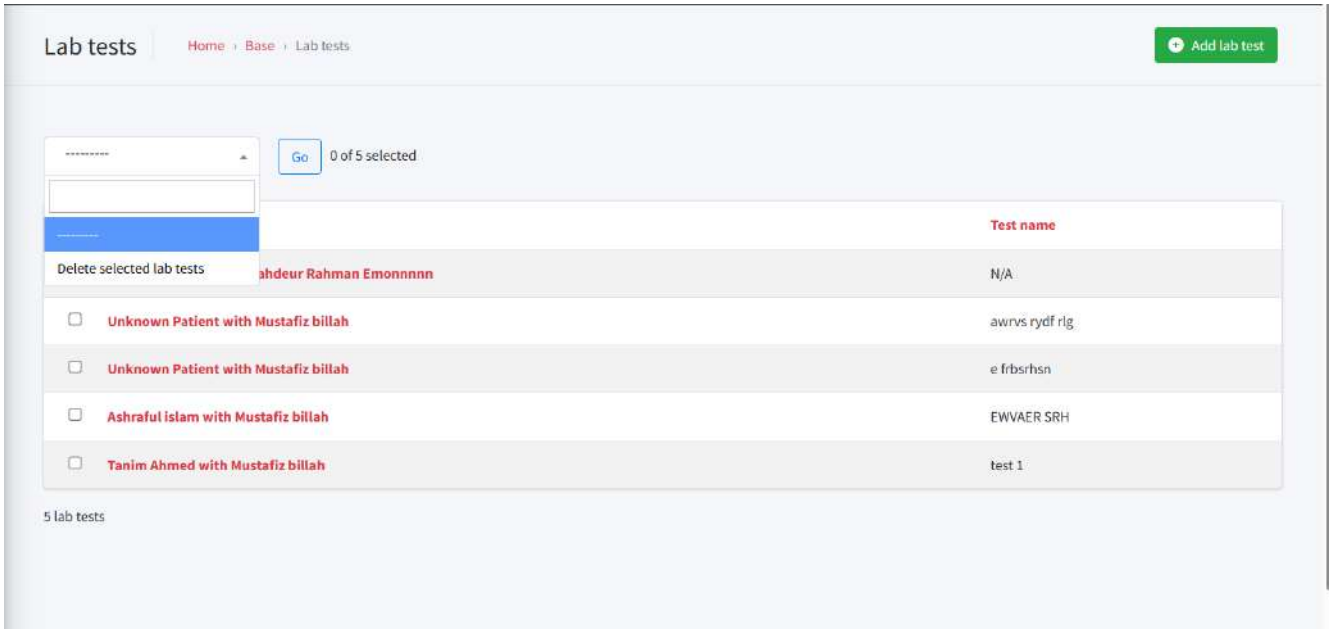


Figure 5.12: Lab Test by admin of CUET Hospital management System

Testing and Sustainability Plan

6.1 Requirements/specifications-based system level test cases

System-level test cases will focus on validating the functional and non-functional requirements outlined in the SRS. These tests include:

- Verifying that students can successfully register and access their E-booklets.
- Ensuring that administrators can manage doctor schedules and update medical records.
- Testing the performance under varying loads and ensuring the system meets the specified uptime and response time.

6.2 Traceability of test cases to use cases

The test cases are traced to the use cases identified during the system design. For example, the "Student Registration" use case will have corresponding test cases that verify input validation, successful registration, and error handling. Similarly, the "View E-Booklet" use case will have test cases ensuring that users can access their medical records correctly.

6.3 Techniques used for test generation

Automated testing tools such as Selenium for UI testing and Postman for API testing will be used to generate and execute test cases. Additionally, unit testing will be performed using Jest to ensure that individual components function as expected.

6.4 Assessment of the goodness of your test suite

The test suite will be evaluated based on:

- **Coverage:** Ensuring all system requirements and use cases are covered by the test cases.
- **Redundancy:** Minimizing overlap in test cases to improve efficiency.
- **Effectiveness:** Verifying that the tests accurately identify defects in the system.

6.5 Sustainability Plan

To ensure the long-term viability of the CUET Hospital Management System, the following measures will be taken:

- **Regular System Updates:** Regular updates and patches will be applied to address security vulnerabilities and add new features.
- **Data Backup:** Daily backups will be taken to safeguard against data loss.
- **Scalability:** The system will be designed to scale with increasing user numbers, ensuring it can handle future growth without performance degradation.

6.5.1 Scalability

The system will be designed to scale by leveraging cloud services such as AWS or Google Cloud for hosting. The database will be optimized for horizontal scaling, ensuring that as the number of students and medical records increases, the system can maintain performance levels.

Data Flow Diagram

7.1 Data flow diagram design following pattern

A Data Flow Diagram (DFD) is a visual representation of how data moves within a system. It uses symbols to depict processes, data stores, and data flows, illustrating how input data is transformed into output information. DFDs help identify the major functions of a system, clarify data movement, and support both system analysis and design by offering a structured understanding of system functionalities.

7.1.1 First Level DFD for the System

The first level DFD shown in Figure 7.1 represents the CUET Medical Center Web Application. Students first register in the system and must be approved by administrators before accessing their digital E-booklet. Doctors create medical prescriptions and store them inside the unique student E-booklet. Additionally, doctor availability information is updated and published through the doctor roster module.

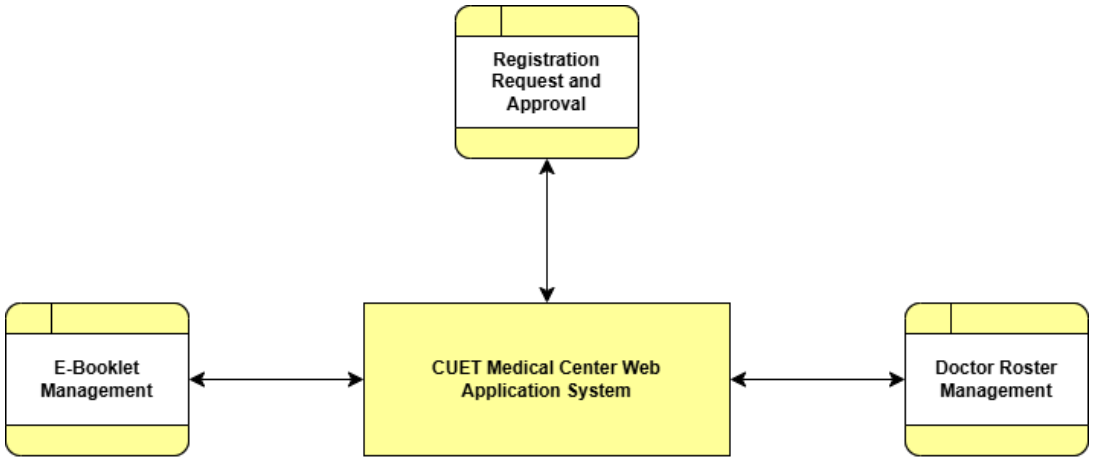


Figure 7.1: First Level Data Flow Diagram for the CUET Hospital Management System

7.1.2 DFD for Administrator

The Data Flow Diagram (DFD) shown in Figure 7.2 illustrates the administrative workflow of the CUET Hospital management System. Administrators access the system through a secure login interface. Upon successful authentication, admins can update their passwords, manage doctor rosters, handle E-booklet operations, and approve or manage student registration and login-related processes.

This DFD clearly outlines the core actions performed by administrators, showing how data flows between admin users and the system's major modules.

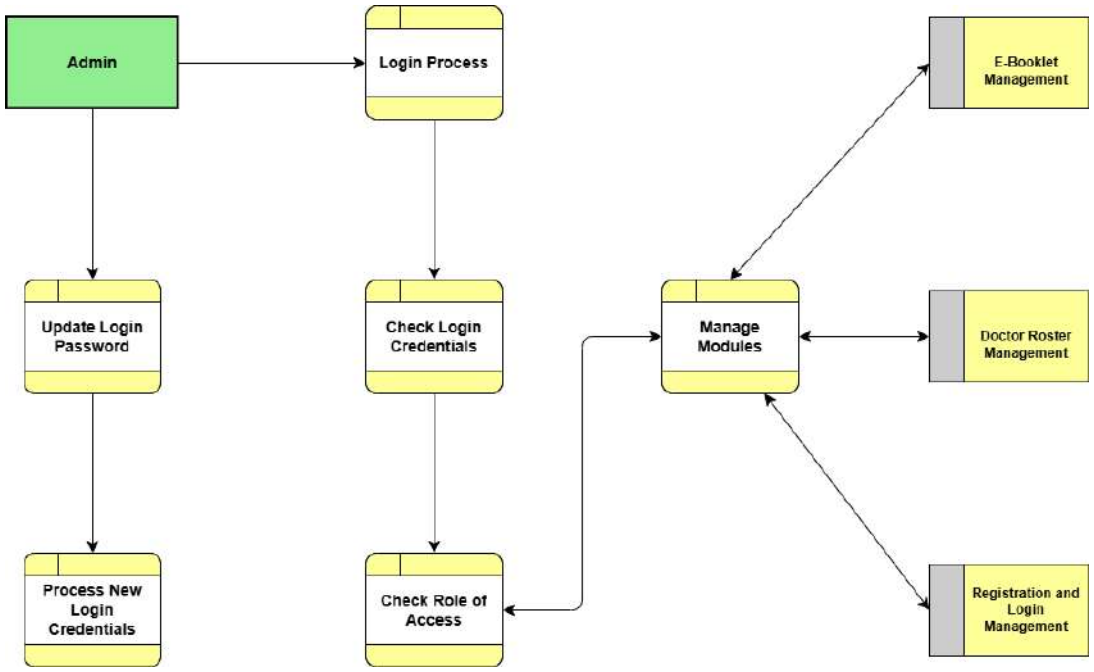


Figure 7.2: Data Flow Diagram for Administrator

7.1.3 DFD for Student

The Data Flow Diagram (DFD) shown in Figure 7.3 represents the initial workflow of the student side of the CUET Hospital management System. Students begin by registering on the platform, providing the necessary information required for identity verification. Once the registration request is submitted, the system forwards it to the administrator for approval.

Upon approval, students gain access to their digital E-booklet where they can view their medical records. Students can also manage their profiles, check doctor availability through the doctor roster, and access other essential medical information related to their care.

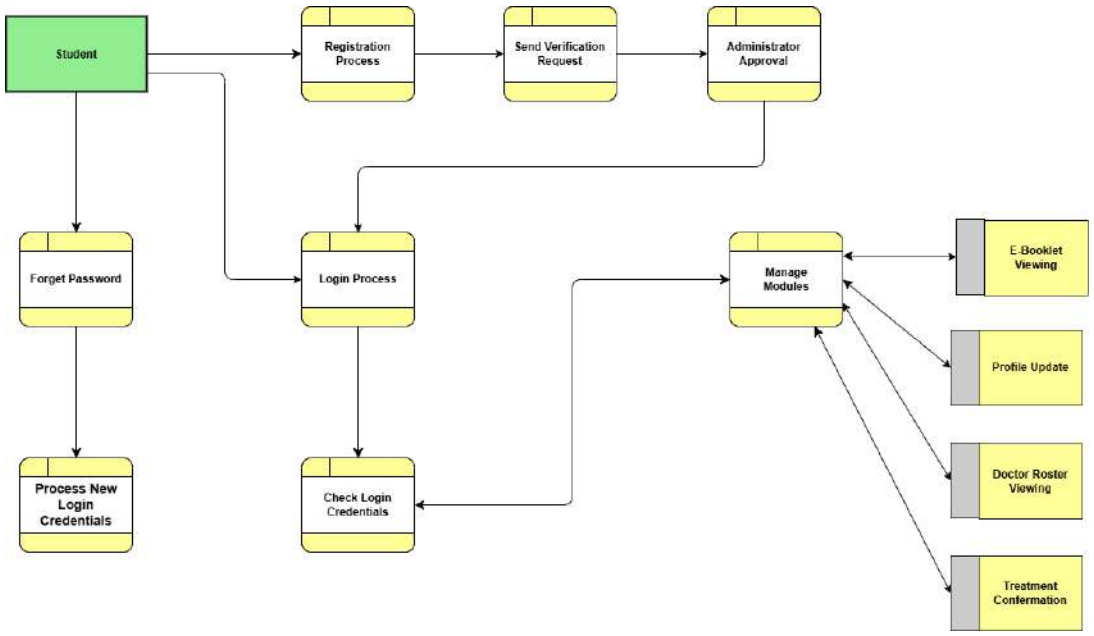


Figure 7.3: Data Flow Diagram for Student

Entity Relationship Diagram

8.1 Entity Relationship diagram design following pattern

8.1.1 List of Entity

- Doctor
- Student
- Ebooklet
- DaySlot

8.1.2 Attributes of the Entities

- **Student:** student_id, student_name, department, email, phone_no., permanent_address, image, gender, hall_name, room_no., blood_group, transaction_id, verified
- **Doctor:** doctor_id, doctor_name, specialization, phone_no., address
- **Ebooklet:** booklet_id, student_id, day, slot, month, year
- **DaySlot:** id, day, slot

8.1.3 Key attribute of the entities

- **Student:** student_id
- **Doctor:** doctor_id
- **Ebooklet:** booklet_id
- **DaySlot:** id

8.1.4 Relationship Diagram

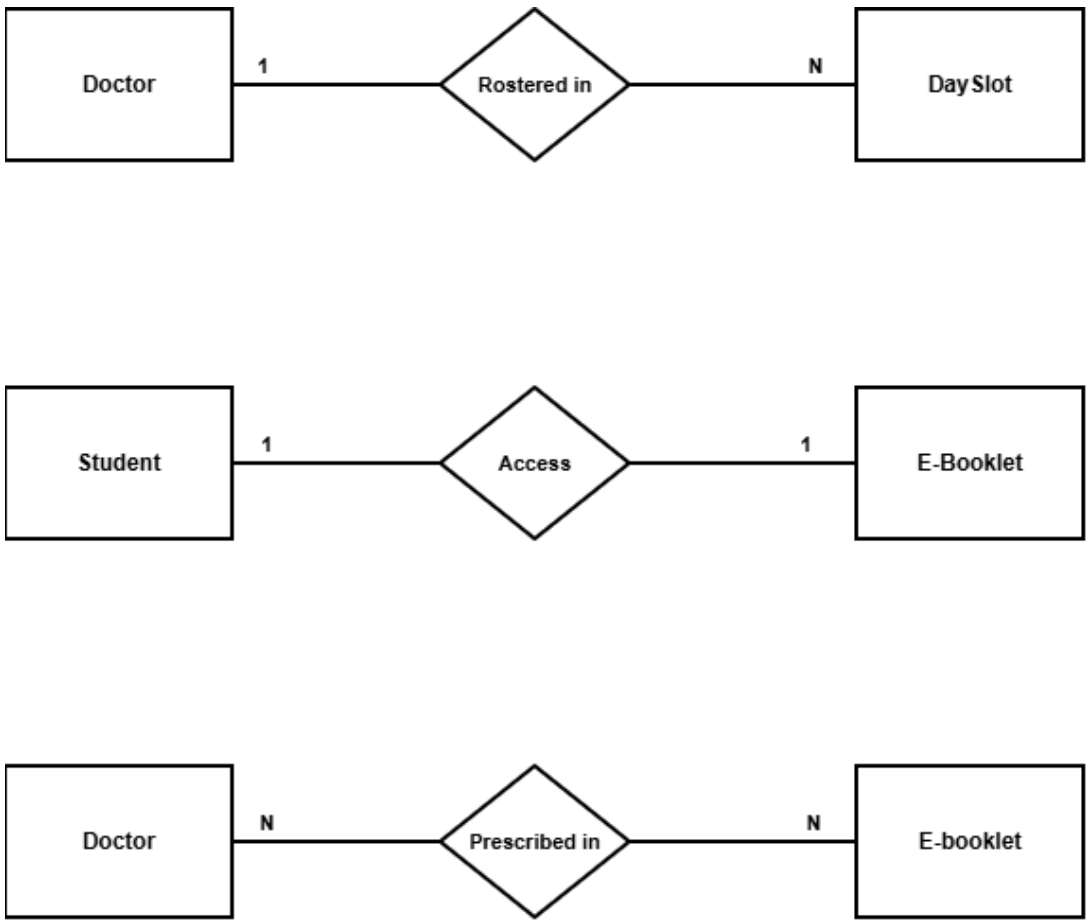


Figure 8.1: Relationship Diagram of CUET Hospital Management System

8.1.5 ER Diagram

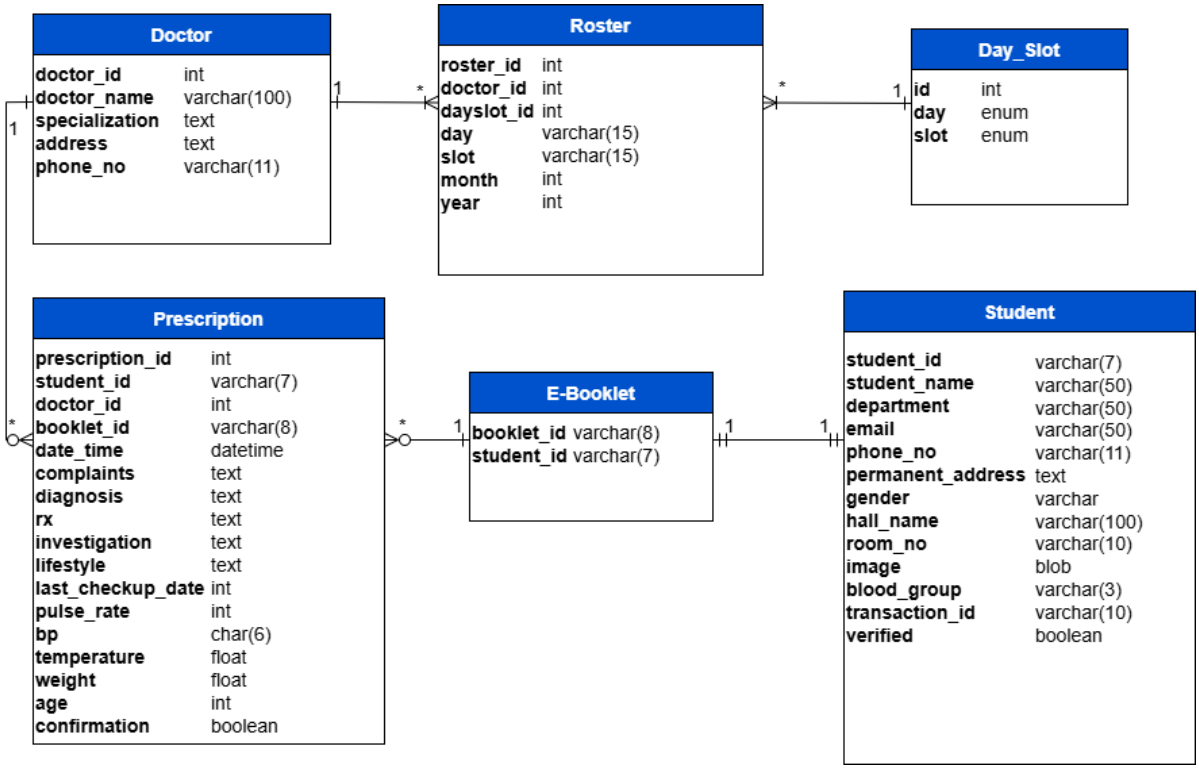


Figure 8.2: Entity Relationship Diagram of CUET Medical Center Application

Acknowledgement

We would like to earnestly acknowledge the sincere efforts and valuable time given by

- **Mr. Md. Saki Kowsar**
  Assistant Professor
  Department of Computer Science and Engineering, CUET
- **Ayesha Banu**
  Lecturer
  Department of Computer Science and Engineering, CUET

Their valuable guidance and feedback has helped us in completing this project.

Bibliography

[1] Pressman, R. S., & Maxim, B. R. (2014). *Software Engineering: A Practitioner's Approach* (8th ed.). McGraw-Hill Education.

[2] Schach, S. R. (2010). *Object-Oriented and Classical Software Engineering* (8th ed.). McGraw-Hill Education.

CUET Hospital Management System API Documentation

Version: 1.0.0
Last Updated: November 2025
Base URL: http://localhost:8000

Table of Contents

Overview

The CUET Hospital Management System API provides a comprehensive solution for managing medical appointments, patient records, doctor schedules, and healthcare services at Chittagong University of Engineering & Technology (CUET). Built with Django, it offers session-based authentication and role-based access control for patients, doctors, and administrators.

Key Features

- **Appointment Management**: Slot-based scheduling system
- **Medical Records**: Comprehensive patient history tracking
- **Lab Tests & Prescriptions**: Complete diagnostic and treatment documentation

- **Real-time Notifications**: Stay updated on appointment status
- **AI Chat Assistant**: Intelligent medical query support
- **E-Booklet**: Digital medical history for patients

Base URL

```
Development: http://localhost:8000
Production: [To be configured]
```

Getting Started

Prerequisites

- Python 3.8+
- Django 4.x
- Active session cookie for authenticated endpoints

Quick Start

1. Register a new user

bash

```bash
curl -X POST "http://localhost:8000/auth/sign-up/?type=Patient" \
  -d "username=johndoe" \
  -d "email=john@example.com" \
  -d "password=secure123" \
  -d "first_name=John" \
  -d "last_name=Doe" \
  -d "student_id=2021001" \
  -d "department=Computer Science" \
  -d "mobile=+8801712345678"
```

2. Login

bash

```bash
curl -X POST "http://localhost:8000/auth/sign-in/" \
  -d "email=john@example.com" \
  -d "password=secure123" \
  -c cookies.txt
```

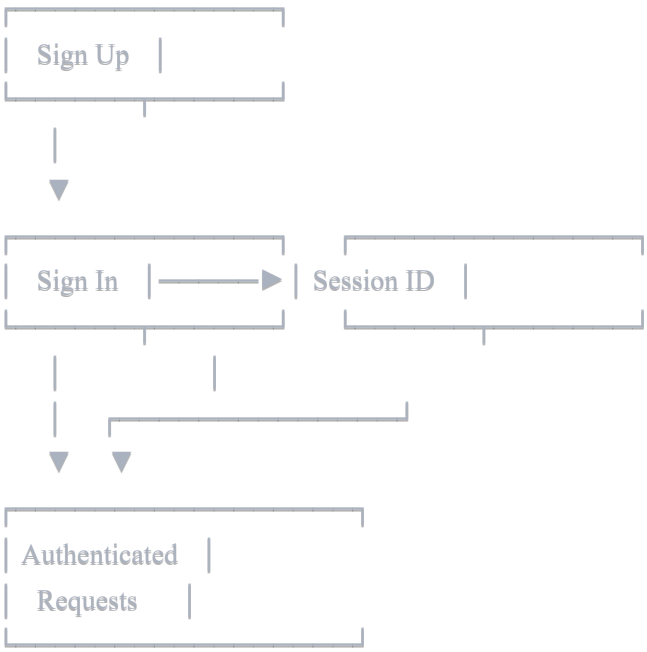
3. Book an appointment

```bash
curl -X POST "http://localhost:8000/book-appointment/1/5/" \
  -b cookies.txt \
  -d "first_name=John" \
  -d "email=john@example.com" \
  -d "slot_id=12" \
  -d "issues=Regular checkup"
```

Authentication

The API uses **session-based authentication** with Django's built-in authentication system.

Authentication Flow



Session Management

- Sessions expire after 2 weeks of inactivity
- Session ID stored in `sessionid` cookie
- CSRF protection enabled for all POST requests
- Secure and HttpOnly flags set in production

Required Headers

For authenticated endpoints:

```
Cookie: sessionid=<your_session_id>
```

For POST requests:

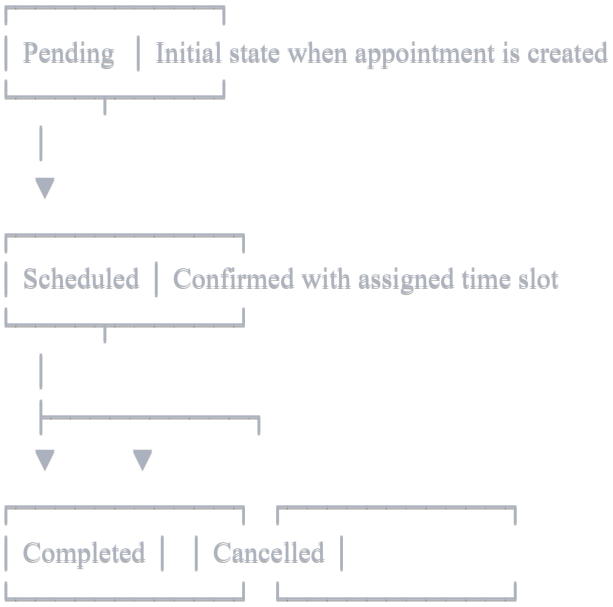
```
Cookie: sessionid=<your_session_id>
X-CSRFToken: <csrf_token>
```

Core Concepts

User Roles

Role	Description	Access Level
Patient	Students receiving medical care	Personal appointments, medical records, notifications
Doctor	Medical professionals	Manage appointments, add diagnoses, prescriptions, lab tests
Admin	System administrators	Full system access, user management

Appointment Lifecycle



Status Transitions

- **Pending → Scheduled**: Automatic upon booking with available slot

- **Scheduled → Completed**: Doctor or patient marks as complete
- **Scheduled → Cancelled**: Either party cancels the appointment
- **Cancelled → Scheduled**: Patient can reactivate if slot available

Slot Management

- Time slots are managed by doctors
- Each slot can hold one appointment
- Cancelled appointments free up slots automatically
- Slots have specific date and time ranges

API Reference

Authentication APIs

Register User

Endpoint: `POST /auth/sign-up/`

Register a new patient or doctor account.

Query Parameters:

Parameter	Type	Required	Default	Description
type	string	No	Patient	User type: `Patient` or `Doctor`

Request Body (Patient):

Field	Type	Required	Description	Example
username	string	Yes	Unique username	johndoe
email	string	Yes	Valid email address	john@example.com
password	string	Yes	Minimum 8 characters	SecurePass123
student_id	string	Yes	Student ID number	2021001
first_name	string	Yes	First name	John
last_name	string	Yes	Last name	Doe
department	string	Yes	Academic department	Computer Science
hall	string	No	Residence hall	Hall A
room_no	string	No	Room number	305
mobile	string	Yes	Phone number	+8801712345678
gender	string	Yes	Gender	Male/Female/Other
dob	date	Yes	Date of birth (YYYY-MM-DD)	2000-01-15
blood_group	string	No	Blood group	A+

Request Body (Doctor):

Field	Type	Required	Description	Example
username	string	Yes	Unique username	drsarah
email	string	Yes	Valid email address	sarah@hospital.com
password	string	Yes	Minimum 8 characters	SecurePass123
full_name	string	Yes	Full name	Dr. Sarah Johnson
mobile	string	Yes	Phone number	+8801812345678
specialization	string	Yes	Medical specialization	Cardiology
qualifications	string	Yes	Medical qualifications	MD, MRCP
years_of_experience	integer	Yes	Years of practice	10

Example Request (Patient):

```bash
```

```
curl -X POST "http://localhost:8000/auth/sign-up/?type=Patient" \
  -H "Content-Type: application/x-www-form-urlencoded" \
  -d "username=johndoe" \
  -d "email=john@example.com" \
  -d "password=SecurePass123" \
  -d "student_id=2021001" \
  -d "first_name=John" \
  -d "last_name=Doe" \
  -d "department=Computer Science" \
  -d "mobile=+8801712345678" \
  -d "gender=Male" \
  -d "dob=2000-01-15" \
  -d "blood_group=A+"
```

Response:

```
302 Found
Location: /auth/sign-in/
```

Error Responses:

Status	Description
400	Invalid input data or duplicate username/email
500	Server error during registration

Login

Endpoint: `POST /auth/sign-in/`

Authenticate user and create session.

Request Body:

Field	Type	Required	Description
email	string	Yes	Registered email
password	string	Yes	User password

Example Request:

```
bash
```

```
curl -X POST "http://localhost:8000/auth/sign-in/" \
  -c cookies.txt \
  -d "email=john@example.com" \
  -d "password=SecurePass123"
```

Response:

```
302 Found
Location: /
Set-Cookie: sessionid=abc123xyz; HttpOnly; Path=/
```

Error Responses:

Status	Description
401	Invalid credentials
400	Missing email or password

Logout

Endpoint: `GET /auth/sign-out/`

Terminate user session.

Authentication: Required

Example Request:

bash

```
curl -X GET "http://localhost:8000/auth/sign-out/" \
  -b cookies.txt
```

Response:

```
302 Found
Location: /auth/sign-in/
```

Public APIs

Home Page

Endpoint: `GET /`

Displays available medical services and doctor schedules.

Example Request:

```bash
curl -X GET "http://localhost:8000/"
```

Response: HTML page with services and available appointment slots

---

Service Details

Endpoint: `GET /service/<service_id>/`

Get detailed information about a specific medical service.

Path Parameters:

Parameter	Type	Description
service_id	integer	Service identifier

Example Request:

```bash
curl -X GET "http://localhost:8000/service/1/"
```

Response: HTML page with service details, doctors, and pricing

---

Book Appointment

Endpoint: `GET, POST /book-appointment/<service_id>/<doctor_id>/`

Book an appointment with a specific doctor for a service.

Path Parameters:

Parameter	Type	Description
service_id	integer	Service identifier
doctor_id	integer	Doctor identifier

Request Body (POST):

Field	Type	Required	Description	Example
first_name	string	Yes	Patient's first name	John
last_name	string	Yes	Patient's last name	Doe
email	string	Yes	Contact email	john@example.com
mobile	string	Yes	Phone number	+8801712345678
gender	string	Yes	Gender	Male
address	string	No	Residential address	123 Main St
dob	date	Yes	Date of birth	2000-01-15
blood_group	string	No	Blood group	A+
slot_id	integer	Yes	Time slot ID	12
issues	text	Yes	Medical concerns	Chest pain
symptoms	text	No	Current symptoms	Shortness of breath

Example Request:

bash

```
curl -X POST "http://localhost:8000/book-appointment/1/5/" \
  -d "first_name=John" \
  -d "last_name=Doe" \
  -d "email=john@example.com" \
  -d "mobile=+8801712345678" \
  -d "gender=Male" \
  -d "dob=2000-01-15" \
  -d "slot_id=12" \
  -d "issues=Regular checkup"
```

Response:

```
302 Found
Location: /appointment-success/123/
```

Error Responses:

Status	Description
404	Service or doctor not found
400	Slot unavailable or invalid data

Appointment Success

Endpoint: `GET /appointment-success/<appointment_id>/`

Display appointment confirmation details.

Path Parameters:

Parameter	Type	Description
appointment_id	integer	Appointment identifier

Example Request:

```bash
curl -X GET "http://localhost:8000/appointment-success/123/"
```

Response: HTML page with appointment confirmation

AI Chat Interface

Endpoint: `GET /chat/`

Access AI-powered medical chat assistant.

Example Request:

```bash
curl -X GET "http://localhost:8000/chat/"
```

Response: HTML chat interface

Ask AI

Endpoint: `POST /ask-ai/`

Get AI-generated responses to medical queries.

Request Body:

Field	Type	Required	Description
question	string	Yes	Medical question

Example Request:

```bash
curl -X POST "http://localhost:8000/ask-ai/" \
  -H "Content-Type: application/json" \
  -d '{"question": "What are the symptoms of flu?"}'
```

Example Response:

```json
{
  "answer": "Common flu symptoms include fever, cough, sore throat, body aches, headache, and fatigue. Symptoms typically
  "status": "success"
}
```

Patient APIs

All patient endpoints require authentication and patient role.

Patient Dashboard

Endpoint: `GET /patient/`

Authentication: Required (Patient)

Display patient dashboard with overview of appointments and notifications.

Example Request:

```bash
curl -X GET "http://localhost:8000/patient/" \
  -b cookies.txt
```

Response: HTML dashboard with appointments, notifications, and quick actions

---

Appointments List

Endpoint: `GET /patient/appointments`

Authentication: Required (Patient)

Get all appointments for the logged-in patient.

Example Request:

```bash
curl -X GET "http://localhost:8000/patient/appointments" \
  -b cookies.txt
```

Response: HTML page with filterable appointments list

Appointment Details

Endpoint: `GET /patient/appointments/<appointment_id>/`

Authentication: Required (Patient)

View detailed information about a specific appointment including medical records, lab tests, and prescriptions.

Path Parameters:

Parameter	Type	Description
appointment_id	integer	Appointment identifier

Example Request:

```bash
curl -X GET "http://localhost:8000/patient/appointments/123/" \
  -b cookies.txt
```

Response: HTML page with complete appointment details

Error Responses:

Status	Description
404	Appointment not found
403	Not authorized to view this appointment

Cancel Appointment

Endpoint: `GET /patient/cancel_appointment/<appointment_id>/`

Authentication: Required (Patient)

Cancel a scheduled appointment and free up the time slot.

Path Parameters:

Parameter	Type	Description
appointment_id	integer	Appointment to cancel

Example Request:

```bash
curl -X GET "http://localhost:8000/patient/cancel_appointment/123/" \
  -b cookies.txt
```

Response:

```
302 Found
Location: /patient/appointments/123/
```

Activate Appointment

Endpoint: `GET /patient/activate_appointment/<appointment_id>/`

Authentication: Required (Patient)

Reactivate a cancelled appointment if the slot is still available.

Path Parameters:

Parameter	Type	Description
appointment_id	integer	Appointment to activate

Example Request:

```bash
```

```
curl -X GET "http://localhost:8000/patient/activate_appointment/123/" \
  -b cookies.txt
```

Response:

```
302 Found
Location: /patient/appointments/123/
```

Error Responses:

Status	Description
400	Slot no longer available

Complete Appointment

Endpoint: `GET /patient/complete_appointment/<appointment_id>/`

Authentication: Required (Patient)

Mark an appointment as completed from patient side.

Path Parameters:

Parameter	Type	Description
appointment_id	integer	Appointment to complete

Example Request:

```bash
curl -X GET "http://localhost:8000/patient/complete_appointment/123/" \
  -b cookies.txt
```

Response:

```
302 Found
Location: /patient/appointments/123/
```

E-Booklet

Endpoint: `GET /patient/e-booklet/`

Authentication: Required (Patient)

View complete medical history with all completed appointments.

Example Request:

```bash
curl -X GET "http://localhost:8000/patient/e-booklet/" \
  -b cookies.txt
```

Response: HTML page with chronological medical history

Notifications

Endpoint: `GET /patient/notifications/`

Authentication: Required (Patient)

Get list of unread notifications.

Example Request:

```bash
curl -X GET "http://localhost:8000/patient/notifications/" \
  -b cookies.txt
```

Response: HTML page with notifications list

Mark Notification as Read

Endpoint: `GET /patient/mark_noti_seen/<id>/`

Authentication: Required (Patient)

Mark a specific notification as read.

Path Parameters:

Parameter	Type	Description
id	integer	Notification identifier

Example Request:

```bash
curl -X GET "http://localhost:8000/patient/mark_noti_seen/45/" \
  -b cookies.txt
```

Response:

```
302 Found
Location: /patient/notifications/
```

Patient Profile

Endpoint: GET, POST /patient/profile/

Authentication: Required (Patient)

View and update patient profile information.

Request Body (POST):

Field	Type	Required	Description
first_name	string	Yes	First name
last_name	string	Yes	Last name
hall	string	No	Residence hall
room_no	string	No	Room number
mobile	string	Yes	Phone number
address	string	No	Full address
gender	string	Yes	Gender
dob	date	Yes	Date of birth
blood_group	string	No	Blood group
image	file	No	Profile picture

Example Request:

```bash
```

```
curl -X POST "http://localhost:8000/patient/profile/" \
  -b cookies.txt \
  -F "first_name=John" \
  -F "last_name=Doe" \
  -F "mobile=+8801712345678" \
  -F "image=@profile.jpg"
```

Response: Updated profile page with success message

Doctor APIs

All doctor endpoints require authentication and doctor role.

Doctor Dashboard

Endpoint: `GET /doctor/`

Authentication: Required (Doctor)

Display doctor dashboard with pending appointments overview.

Example Request:

```
bash

curl -X GET "http://localhost:8000/doctor/" \
  -b cookies.txt
```

Response: HTML dashboard with today's appointments and statistics

---

Doctor Appointments

Endpoint: `GET /doctor/appointments/`

Authentication: Required (Doctor)

Get all appointments for the logged-in doctor (excluding completed).

Example Request:

```
bash

curl -X GET "http://localhost:8000/doctor/appointments/" \
  -b cookies.txt
```

Response: HTML page with appointments list

Appointment Details

Endpoint: `GET /doctor/appointments/<appointment_id>/`

Authentication: Required (Doctor)

View detailed appointment information with medical data entry forms.

Path Parameters:

Parameter	Type	Description
appointment_id	integer	Appointment identifier

Example Request:

```bash
curl -X GET "http://localhost:8000/doctor/appointments/123/" \
  -b cookies.txt
```

Response: HTML page with appointment details and medical forms

---

Cancel Appointment

Endpoint: `GET /doctor/cancel_appointment/<appointment_id>/`

Authentication: Required (Doctor)

Cancel appointment and automatically free the time slot.

Path Parameters:

Parameter	Type	Description
appointment_id	integer	Appointment to cancel

Example Request:

```bash
```

```
curl -X GET "http://localhost:8000/doctor/cancel_appointment/123/" \
  -b cookies.txt
```

Response:

```
302 Found
Location: /doctor/appointments/123/
```

Complete Appointment

Endpoint: `GET /doctor/complete_appointment/<appointment_id>/`

Authentication: Required (Doctor)

Mark appointment as completed and free the time slot.

Path Parameters:

Parameter	Type	Description
appointment_id	integer	Appointment to complete

Example Request:

bash

```
curl -X GET "http://localhost:8000/doctor/complete_appointment/123/" \
  -b cookies.txt
```

Response:

```
302 Found
Location: /doctor/appointments/123/
```

Add Medical Report

Endpoint: `POST /doctor/add_medical_report/<appointment_id>/`

Authentication: Required (Doctor)

Add medical diagnosis and treatment record to appointment.

Path Parameters:

Parameter	Type	Description
appointment_id	integer	Appointment identifier

Request Body:

Field	Type	Required	Description
diagnosis	text	Yes	Medical diagnosis
treatment	text	Yes	Treatment plan

Example Request:

```bash
curl -X POST "http://localhost:8000/doctor/add_medical_report/123/" \
  -b cookies.txt \
  -d "diagnosis=Type 2 Diabetes Mellitus" \
  -d "treatment=Metformin 500mg twice daily, dietary modifications"
```

Response:

```
302 Found
Location: /doctor/appointments/123/
```

Edit Medical Report

Endpoint: `POST /doctor/edit_medical_report/<appointment_id>/<medical_report_id>/`

Authentication: Required (Doctor)

Update existing medical record.

Path Parameters:

Parameter	Type	Description
appointment_id	integer	Appointment identifier
medical_report_id	integer	Medical report identifier

Request Body:

Field	Type	Required	Description
diagnosis	text	Yes	Updated diagnosis
treatment	text	Yes	Updated treatment

Example Request:

```bash
curl -X POST "http://localhost:8000/doctor/edit_medical_report/123/45/" \
  -b cookies.txt \
  -d "diagnosis=Type 2 Diabetes Mellitus - Well Controlled" \
  -d "treatment=Continue Metformin, add exercise regimen"
```

Response:

```
302 Found
Location: /doctor/appointments/123/
```

Add Lab Test

Endpoint: `POST /doctor/add_lab_test/<appointment_id>/`

Authentication: Required (Doctor)

Add laboratory test order and results.

Path Parameters:

Parameter	Type	Description
appointment_id	integer	Appointment identifier

Request Body:

Field	Type	Required	Description
test_name	string	Yes	Name of lab test
description	text	No	Test description
result	text	No	Test results

Example Request:

```bash
```

```
curl -X POST "http://localhost:8000/doctor/add_lab_test/123/" \
  -b cookies.txt \
  -d "test_name=HbA1c" \
  -d "description=Glycated hemoglobin test" \
  -d "result=6.5% - Borderline diabetic range"
```

Response:

```
302 Found
Location: /doctor/appointments/123/
```

Edit Lab Test

Endpoint: `POST /doctor/edit_lab_test/<appointment_id>/<lab_test_id>/`

Authentication: Required (Doctor)

Update existing lab test information.

Path Parameters:

Parameter	Type	Description
appointment_id	integer	Appointment identifier
lab_test_id	integer	Lab test identifier

Request Body:

Field	Type	Required	Description
test_name	string	Yes	Name of lab test
description	text	No	Test description
result	text	No	Test results

Example Request:

bash

```
curl -X POST "http://localhost:8000/doctor/edit_lab_test/123/67/" \
  -b cookies.txt \
  -d "test_name=HbA1c Follow-up" \
  -d "result=5.8% - Normal range"
```

Response:

```
302 Found
Location: /doctor/appointments/123/
```

Add Prescription

Endpoint: `POST /doctor/add_prescription/<appointment_id>/`

Authentication: Required (Doctor)

Add medication prescription for patient.

Path Parameters:

Parameter	Type	Description
appointment_id	integer	Appointment identifier

Request Body:

Field	Type	Required	Description
medications	text	Yes	List of prescribed medications

Example Request:

```bash
curl -X POST "http://localhost:8000/doctor/add_prescription/123/" \
  -b cookies.txt \
  -d "medications=1. Metformin 500mg - Twice daily after meals\n2. Aspirin 75mg - Once daily\n3. Atorvastatin 10mg - Once
```

Response:

```
302 Found
Location: /doctor/appointments/123/
```

Edit Prescription

Endpoint: `POST /doctor/edit_prescription/<appointment_id>/<prescription_id>/`

Authentication: Required (Doctor)

Update existing prescription.

Path Parameters:

Parameter	Type	Description
appointment_id	integer	Appointment identifier
prescription_id	integer	Prescription identifier

Request Body:

Field	Type	Required	Description
medications	text	Yes	Updated medication list

Example Request:

```bash
curl -X POST "http://localhost:8000/doctor/edit_prescription/123/89/" \
  -b cookies.txt \
  -d "medications=1. Metformin 850mg - Twice daily\n2. Aspirin 75mg - Once daily"
```

Response:

```
302 Found
Location: /doctor/appointments/123/
```

Doctor Payments

Endpoint: `GET /doctor/payments/`

Authentication: Required (Doctor)

View all completed appointments with payment information.

Example Request:

```bash
curl -X GET "http://localhost:8000/doctor/payments/" \
  -b cookies.txt
```

Response: HTML page with payments list and total earnings

Doctor Notifications

Endpoint: `GET /doctor/notifications/`

Authentication: Required (Doctor)

Get unread notifications for the doctor.

Example Request:

```bash
curl -X GET "http://localhost:8000/doctor/notifications/" \
  -b cookies.txt
```

Response: HTML page with notifications list

Doctor Profile

Endpoint: `GET, POST /doctor/profile/`

Authentication: Required (Doctor)

View and update doctor profile and availability.

Request Body (POST):

Field	Type	Required	Description
full_name	string	Yes	Full name
mobile	string	Yes	Phone number
country	string	No	Country
bio	text	No	Professional biography
specialization	string	Yes	Medical specialization
qualifications	string	Yes	Medical qualifications
years_of_experience	integer	Yes	Years of practice
next_available_appointment_date	date	No	Next available date
image	file	No	Profile picture

Example Request:

```bash
curl -X POST "http://localhost:8000/doctor/profile/" \
  -b cookies.txt \
  -F "full_name=Dr. Sarah Johnson" \
  -F "specialization=Cardiology" \
  -F "years_of_experience=12" \
  -F "image=@doctor_photo.jpg"
```

Response: Updated profile page with success message

Error Handling

Error Response Format

All errors return appropriate HTTP status codes with user-friendly messages in the response.

```html
<!-- HTML Error Page -->
<div class="error-message">
  <h2>Error: Invalid Request</h2>
  <p>The appointment slot is no longer available.</p>
</div>
```

Common Error Scenarios

Scenario	Status Code	Description
Invalid credentials	401	Email or password incorrect
Unauthorized access	403	User lacks permission for resource
Resource not found	404	Appointment, user, or service not found
Validation error	400	Invalid input data
Slot unavailable	400	Time slot already booked
Session expired	401	User needs to log in again
Server error	500	Internal server error occurred

Status Codes

HTTP Status Codes Used

Code	Meaning	Usage
200	OK	Successful GET request
302	Found	Successful redirect after POST
400	Bad Request	Invalid input or business logic error
401	Unauthorized	Authentication required or failed
403	Forbidden	Insufficient permissions
404	Not Found	Resource doesn't exist
500	Internal Server Error	Server-side error

Appointment Status Values

Status	Description
`pending`	Appointment created, awaiting confirmation
`scheduled`	Confirmed with time slot
`completed`	Appointment finished
`cancelled`	Appointment cancelled by either party

Changelog

Version 1.0.0 (November 2025)

Added

- Complete authentication system (sign-up, sign-in, sign-out)
- Appointment booking and management
- Patient dashboard and medical history (E-Booklet)
- Doctor dashboard with appointment management
- Medical records management (diagnosis, treatment)
- Lab test ordering and results
- Prescription management
- Real-time notifications system

- AI chat assistant for medical queries
- Profile management for patients and doctors
- Slot-based scheduling system

Security

- Session-based authentication
- CSRF protection on all forms
- HttpOnly and Secure cookie flags
- Role-based access control
- Input validation and sanitization

Support

Getting Help

- **Documentation Issues**: Report inaccuracies or suggestions
- **API Bugs**: Submit detailed bug reports with request/response examples
- **Feature Requests**: Propose new features or improvements

Best Practices

1. **Always handle errors gracefully** in your client application
2. **Store session cookies securely** and don't expose them
3. **Validate user input** before sending requests
4. **Use HTTPS in production** to encrypt data in transit
5. **Implement request timeouts** to handle slow responses
6. **Cache static data** like service lists to reduce API calls
7. **Test in development environment** before deploying changes

---

End of Documentation

For questions or support, contact the development team.

Project Name	Hospital Management System (Django)
Module Name	Authentication, Patient, Appointment, Billing
Created By	(Avishek Biswas)
Reviewed By	(Ashraful Islam and Raul Saha)
Date of Creation	(19-01-2026)
Date of Review	(19-01-2026)

Table 1: Requirements/specifications-based system level test cases

TC ID	**Scenario**	**Steps**	**Test Data**	**Expected Result**	**Status**
TC-Auth-Login-01	Verify valid login	1. Open Login page. 2. Enter valid credentials. 3. Click Login.	user: admin pass: Admin@123	User logged in successfully and redirected to dashboard.	Passed
TC-Auth-Login-02	Verify invalid login blocked	1. Open Login page. 2. Enter incorrect password. 3. Click Login.	user: admin pass: wrong123	Error message displayed; user not logged in.	Passed
TC-Auth-Logout-01	Verify logout	1. Login. 2. Click Logout. 3. Try opening dashboard URL.	N/A	Session ends; login required again.	Passed
TC-Role-Access-01	Verify role-based access	1. Login as patient. 2. Try opening admin-only page.	role: patient	Access denied (403) or redirected.	Passed
TC-Patient-Create-01	Verify add patient	1. Open Add Patient. 2. Fill required fields. 3. Click Save.	Name: Bahul Saha Age: 30	Patient record saved successfully.	Passed
TC-Patient-Create-02	Verify validation	1. Open Add Patient. 2. Leave phone empty. 3. Click Save.	Phone: empty	Validation error shown.	Passed
TC-Appt-Book-01	Verify appointment	1. Open Create Appointment. 2. Select data. 3. Confirm booking.	Doctor: D005 Time: 10AM	Appointment created successfully.	Passed

Continued on next page

TC ID	Scenario	Steps	Test Data	Expected Result	Status
TC-Appt-Book-02	Double booking	1. Book Doc D005 at 10AM. 2. Try booking same slot again.	Doctor: D005 Time: 10AM	System blocks duplicate booking.	Passed
TC-Bill-Gen-01	Verify billing	1. Open billing page. 2. Add services. 3. Generate invoice.	Cons.: 500 Test: 800	Invoice total (1300) is correct.	Passed