CSAI 203 - Fall 2025

Introduction to Software Engineering

Medibook

Team:

Ahmed Ragheb **202301566**

Ammar Yasser **202400G63**

Ahmed Ibrahem **202402177**

Yossef Ahmed **202300216**

1.Introduction

1.1 Purpose of the Document:

His design document describes how the Medi Book system will be structured and implemented. Its purpose is to translate the requirements defined in the SRS into a clear design that guides the development process. The document explains the system architecture, the role of each major component, the chosen design pattern (MVC), and the diagrams that illustrate how the system operates. It serves as a reference for the development team during implementation and ensures that all members share the same understanding of how the system will be built.

1.2 Scope of the Design Phase

His design document describes how the Medi Book system will be structured and implemented. Its purpose is to translate the requirements defined in the SRS into a clear design that guides the development process. The document explains the system architecture, the role of each major component, the chosen design pattern (MVC), and the diagrams that illustrate how the system operates. It serves as a reference for the development team during implementation and ensures that all members share the same understanding of how the system will be built.

1.3 Intended Audience

This document is intended for the project development team, the teaching staff responsible for evaluating the project, and any future contributors who may need to understand the design of Medi Book. It is written to help developers follow the same design approach and to make the system easier to maintain, update, or expand later on.

The document begins with an introduction explaining its purpose and scope. It then presents an overview of the MediBook system and highlights its main functions. The architecture section describes the structure of the system and the technologies used. The detailed design section explains how the MVC pattern is applied and includes UML diagrams, user interface mockups, and the database design. The document ends with a summary of the key design decisions.

1.4 overview

MediBook is a web-based medical appointment booking platform designed to connect patients with doctors quickly and efficiently. The system provides an intuitive interface where users can search for doctors by specialty and location, apply filters such as rating or consultation fee, and view detailed doctor profiles. Patients can browse available appointment slots, select a preferred time, and confirm bookings through a streamlined workflow.

The platform supports three main user roles: **patients**, **doctors**, and **administrators**. Patients can manage their appointments, view booking history, and communicate with clinics. Doctors can manage their schedules, view upcoming appointments, and respond to booking requests. Administrators are responsible for overseeing system operations, verifying doctor accounts, and maintaining platform data consistency.

MediBook follows an MVC architecture, separating the user interface, application logic, and database interactions. The system integrates search functionality, scheduling logic, authentication, and review management to deliver a seamless user experience. By providing a centralized and user-friendly environment for healthcare appointment management, MediBook enhances accessibility, reduces manual coordination, and improves overall efficiency in patient–doctor communication.

2.System Overview

2.1 Brief Description of the System

an organized and user-friendly way. The system allows patients to search for doctors based on specialty, location, or availability, view doctor profiles, and book appointment slots. Each doctor has a profile that includes personal information, available times, and their clinic's details. Patients can manage their upcoming and past bookings through their personal dashboard.

Doctors also have their own dashboard where they can manage their schedules, accept or cancel appointments, and update their information. Additionally, the system provides an admin panel for verifying doctor accounts and monitoring system activity. The goal of MediBook is to simplify the booking process, reduce communication overhead, and provide a smooth experience for both patients and doctors through a clean and straightforward interface.

2.2 Design Goals

The design of the MediBook platform is guided by a set of goals and quality attributes intended to ensure that the system is reliable, maintainable, and user-friendly. These goals define the architectural direction of the system and influence key decisions related to structure, scalability, interface design, and performance.

2.2.1 Usability

A primary goal of MediBook is to provide a simple and intuitive experience for patients, doctors, and administrators. Users should be able to search for doctors, view profiles, and book appointments with minimal effort. Interface layouts, such as the search page and doctor profile screens, are designed for clarity and ease of navigation. Consistency across all screens ensures predictable interaction and reduces user learning time.

2.2.2 Maintainability

The system is structured using an MVC architecture to ensure modularity and ease of updates. Each component—Models, Views, and Controllers—has a clearly defined responsibility, which simplifies code maintenance, debugging, and future enhancements. This design allows new features (e.g., payment integration or mobile support) to be added without disrupting existing modules.

2.2.3 Scalability

MediBook is designed to support growth in the number of users, doctors, clinics, and appointments. The system architecture separates business logic from data storage, allowing database scaling or migration without affecting the user interface. As user demand increases, the platform can accommodate additional features or higher traffic with minimal architectural changes.

2.2.4 Performance

Fast system response is essential for user satisfaction. Search queries, appointment availability checks, and dashboard operations are optimized to respond quickly, even with large datasets. The design ensures efficient database queries and minimal page load times, improving overall system responsiveness.

2.2.5Reliability

The platform is designed to ensure accurate appointment scheduling and conflict-free slot management. Error handling and validation are applied at both the controller and database levels to prevent double bookings, invalid user actions, or system failures. Reliability ensures consistency in doctor availability and booking confirmation.

2.2.6 Security

Security is a key design objective, as the system handles sensitive medical appointment information. User authentication, password hashing, access control, and secure session management ensure that personal data remains protected. Only authorized users can access role-specific features, and administrator privileges are restricted to system oversight tasks.

2.2.7**Extensibility**

The system is designed to support future expansions such as online payment, telemedicine features, mobile apps, or analytical dashboards. The modular design and separation of concerns make it easy to integrate additional services without restructuring core components.

3.Architectural Design:

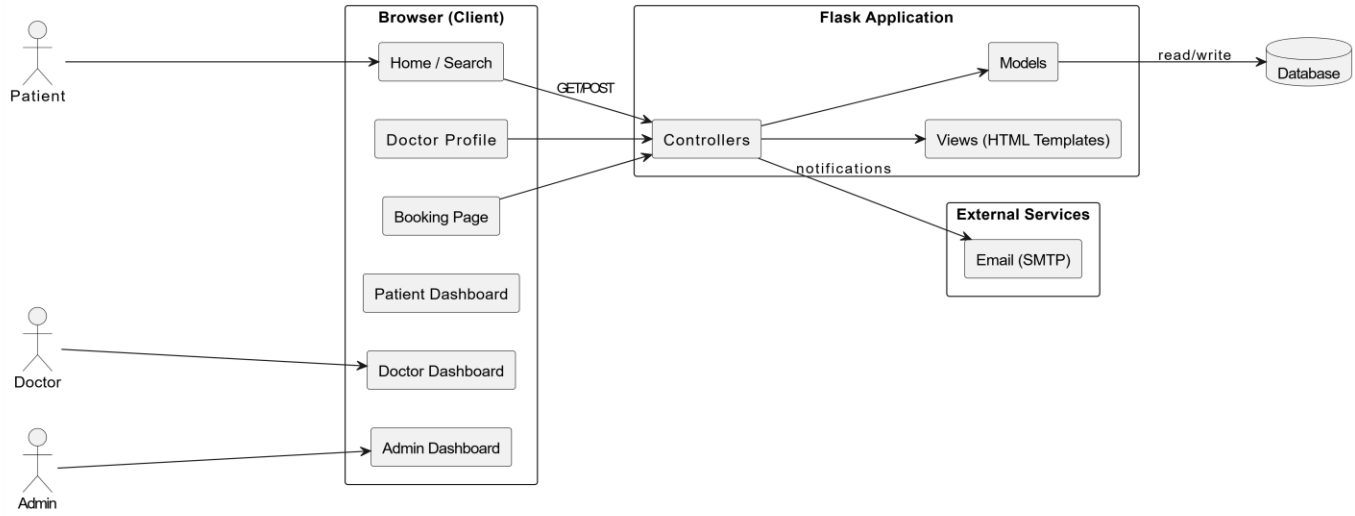
3.1 System architecture diagram

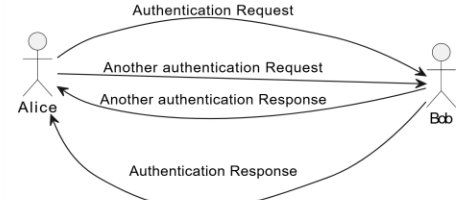
MediBook follows a **monolithic web architecture**, where all main components of the system operate within a single application. Users access the system through a web browser, which sends requests to a Flask backend. The backend processes each request using controller functions, retrieves or updates data through the model layer, and then returns the appropriate HTML view to the user.

The architecture consists of the following parts:

- **Browser (Client):** Displays pages such as search, doctor profiles, booking pages, and dashboards.
- **Controllers (Flask Routes):** Handle requests, apply system logic, and coordinate between the view and model.
- **Models:** Represent the system's data, including users, doctors, appointments, and clinics, and communicate with the database.
- **Views (HTML Templates):** Render the interface shown to users, using simple HTML supported by Jinja templates.
- **Database:** Stores all system information such as profiles, schedules, bookings, and admin data.
- **External Services:** Used for sending confirmation emails or notifications when needed.

This structure makes the system straightforward to implement and maintain, while still allowing clear separation between data, logic, and presentation.





3.2 Discussion of Architectural Style and Components

The architectural design of MediBook is based on the Model–View–Controller (MVC) architectural style. This style is selected for its clear separation of concerns, maintainability, and suitability for interactive web-based applications. The MVC architecture ensures that user interface elements, business logic, and data management remain independent, allowing the system to evolve and scale efficiently.

3.2.1Architectural Style: MVC

Model

The Model layer represents the core data structures and business logic of the system. It manages:

Users (patients, doctors, administrators)

Doctor details (specialization, clinic information, schedule)

Appointments (date, time slot, status)

Reviews and ratings

Notifications

The Model encapsulates validation rules, data transformations, and database interactions. It ensures data integrity and consistency across all system operations such as booking, editing schedules, or submitting reviews.

View

The View layer consists of the user interface components rendered to patients, doctors, and administrators. Examples include:

Search page

Doctor profile page

Appointment booking page

Patient dashboard

Doctor dashboard

Admin management screens

The View layer focuses solely on presenting data and capturing user input. It contains no business logic and relies entirely on controllers for data processing. This separation ensures clear UI design, easier redesigns, and improved user experience.

Controller

The Controller layer contains the system's operational logic and acts as the intermediary between the user interface and the underlying data. Key controllers include:

Auth Controller: login, registration, session management

Search Controller: processes doctor search and filters

Booking Controller: appointment creation, slot validation, cancellation

Doctor Controller: schedule management, appointment review and approval

Admin Controller: doctor verification, user management

Controllers interpret incoming requests, apply the appropriate business rules, interact with the Model, and choose which View to return. This ensures a clean flow of logic and avoids duplication.

3.2.2 Justification for MVC

The choice of MVC is justified by several system requirements:

Separation of Concerns:

UI changes do not affect appointment logic, and logic changes do not affect database structure.

Maintainability:

Each part of the system can be updated or debugged independently.

Scalability:

As the number of users and appointments grows, models and controllers can be optimized or expanded without redesigning the entire system.

Role-based workflows:

Different interfaces for patients, doctors, and admins are easier to manage when views are isolated from logic.

Testability:

Controllers and models can be unit-tested independently without involving UI components.

3.2.3Major System Components

1. User Management Component

Handles:

Registration

Login

Authentication

Role-based access control

This component ensures secure and controlled access for all user types.

2. Doctor Management Component

Includes:

Doctor profiles

Clinic details

Schedules and available slots

Appointment approval or rejection

This component supports doctor-specific workflows and schedule administration.

3. Appointment Management Component

Core of the system; handles:

Slot selection

Appointment creation

Conflict checking

Cancellation and rescheduling

Appointment status updates

It maintains consistency between doctor availability and patient bookings.

4. Search and Filtering Component

Allows patients to search for doctors using:

Specialty

Location

Rating

Fee range

Availability ("open now")

This component optimizes discoverability and improves user navigation.

5. Notification Component

Responsible for delivering:

Appointment confirmations

Status updates

Reminders

Admin notifications

Supports both email/SMS integration (or mock implementations in early phases).

6. Review and Rating Component

Enables patients to:

Rate doctors after appointments

Submit written feedback

View reviews on doctor profiles

Improves transparency and decision-making for future users.

7. Admin Management Component

Used by administrators to:

Verify doctor accounts

Manage users

Oversee system activities

Handle data corrections or disputes

Ensures system integrity and operational oversight.

3.2.3Interaction Between Components

Components interact through the MVC structure:

Views send user input to Controllers (e.g., booking request).

Controllers validate and process the input (e.g., checking availability).

Models store and retrieve data (e.g., saving an appointment).

The Controller returns an updated View (e.g., booking confirmation).

This flow ensures a consistent and predictable system architecture aligned with good software engineering practice.

3.3 Technology Stack and Tools

The development of the MediBook platform relies on a carefully selected set of technologies and tools that support scalability, maintainability, and efficient implementation. The chosen stack aligns with the project's architectural requirements (MVC), supports rapid development, and ensures compatibility with common deployment environments.

3.3.1 Front-End Technologies

HTML5

Used to structure all user interface pages, including search screens, doctor profiles, booking pages, and dashboards.

CSS3

Provides styling, layout, and visual consistency across the platform. Ensures responsiveness and a clean user experience.

JavaScript

Adds interactive behavior to UI elements, such as dynamic form updates, dropdown filtering, and real-time interface responses.

Bootstrap / Basic CSS Framework

Used to streamline layout design and create clean, modern interface components (buttons, forms, grids).

Note: If you used plain CSS only, this sentence can be removed.

3.3.2 Back-End Technologies

Python

The primary programming language for server-side logic due to its readability, extensive libraries, and suitability for rapid development.

Flask Framework

Chosen for its lightweight and flexible nature, ideal for implementing MVC-like architectures. Manages routing, controllers, session handling, form processing, and template rendering.

Jinja2 Templates

Used to populate dynamic data into HTML pages, connecting the controller logic to the view layer.

3.3.3 Database Technologies

SQLite / PostgreSQL

Stores system data, including users, doctors, appointments, schedules, reviews, and notifications.

SQLite is suitable for early development and testing; PostgreSQL supports future scalability.

SQLAlchemy (ORM)

Provides an abstraction layer between Python objects and database tables, simplifying CRUD operations and improving maintainability.

3.3.4 Tools and Development Environment

VS Code / PyCharm

Used as the primary development environment for coding, debugging, and file organization.

Git & GitHub

Used for version control, collaboration support, and maintaining the project repository.

Draw.io / Lucidchart

Used to design UML diagrams, including architecture diagrams, sequence diagrams, class diagrams, and component diagrams for Phase 3.

Figma / Wireframing Tools (optional)

Used for designing UI wireframes and prototypes during the design phase.

3.3.5 Testing and Debugging Tools

Flask Testing Tools / Unittest

Used to create automated unit tests for authentication, booking logic, and controller behavior.

Postman (optional)

Supports testing of API endpoints and request/response flows if the team chooses to expose REST interfaces.

3.3.6 Deployment and Hosting

Localhost / WSGI Servers (e.g., Gunicorn) (future phases)

Used to serve the application during testing and early deployment stages.

Cloud Hosting (optional for future)

Platforms such as Render, Heroku, or AWS can be used to deploy the system in Phase 4–5.

3.3.7 Justification of Technology Choices

Flask provides a clean, flexible foundation for MVC-based systems.

Python ensures rapid development and easy integration with database layers.

SQLAlchemy improves maintainability and prevents raw SQL errors.

HTML/CSS/JS ensure accessibility and responsiveness across devices.

GitHub supports version tracking and collaboration.

UML tools (Draw.io) allow professional documentation throughout Phase 3.

These technologies together create a reliable, scalable, and maintainable environment suitable for MediBook and future enhancements.

4.Detailed Desgin

4.1 User Interface Design

This section describes the main user interface screens of the MediBook platform. The design focuses on clarity, ease of navigation, and a consistent layout that supports efficient interaction for both patients and doctors. The UI aims to minimize user effort by presenting structured information and intuitive actions across all pages.

4.1.1Home Page (Landing Page)

The Home Page is the first screen users encounter when accessing the MediBook platform.

It introduces the application and provides a straightforward starting point for searching doctors.

The layout contains:

A top navigation bar with the application logo and essential links.

A prominent search bar in the center where users can search by doctor name, specialization, or location.

Clean spacing and a minimalistic design to maintain user focus on the search action.

The purpose of this page is to quickly guide users toward finding the medical service they need.

4.1.2Doctor Search Results Page

This page displays a list of doctors that match the user's search criteria.

Key elements include:

A series of doctor "cards," each displaying the doctor's photo, name, specialization, clinic, and rating.

A "View Profile" button for each doctor to access detailed information.

A search/filter bar at the top of the page to refine results without returning to the previous page.

The design supports easy comparison between doctors and smooth navigation through the results.

```
MediBook/
│
├── app.py                  # [Controller] Main Flask application file (Entry Point)
├── config.py               # Configuration settings (Database URI, Secret Keys)
├── requirements.txt        # List of Python dependencies
│
├── models/                 # [MODEL Layer] Database Tables & Logic
│   ├── __init__.py
│   ├── user_model.py       # Handles User,Patient, Doctor data
│   └── appointment_model.py # Handles Appointments & Clinics dat
│
├── controllers/            # [CONTROLLER Layer] Routes & Business Logic
│
├── __init__.py
│
├──auth_routes.py       # Login, Register,Logout logic
│
├──doctor_routes.py     # Search & Profile view logic
│
└── booking_routes.py    # Appointment booking logic
│
├── static/                 # [VIEW Assets] Static files
│
├── css/
│
└── style.css       # Styling for the website
│
├── js/
│
└── script.js       # JavaScript for frontend interactivity
│
└── images/
│
└── logo.png        # Project images
│
└── templates/              # [VIEW Layer] HTML Files
 │
 ├── base.html          # Master layout (Navbar &Footer)
  ├── home.html          # Search page
   ├── login.html         # Login page
   ├──register.html       # Registration page
   ├──doctor_profile.html  # Doctor details & Booking
   └── dashboard.html      # User & Doctor Dashboard
```

4.1.3Doctor Profile Page

The Doctor Profile Page provides detailed information about a selected doctor and allows users to choose an appointment slot.

Displayed elements include:

A doctor photo and header section showing name, specialty, experience, and rating.

Clinic information such as address and contact details.

A schedule section displaying available appointment slots organized by date and time.

A clear “Book Appointment” button to proceed with the booking process.

The interface is structured to help users quickly understand the doctor’s qualifications and availability.

4.1.4Appointment Booking Confirmation Page

This page summarizes all details before the user finalizes an appointment.

It includes:

Appointment information such as selected date, time, doctor, and clinic.

User details to confirm identity and contact information.

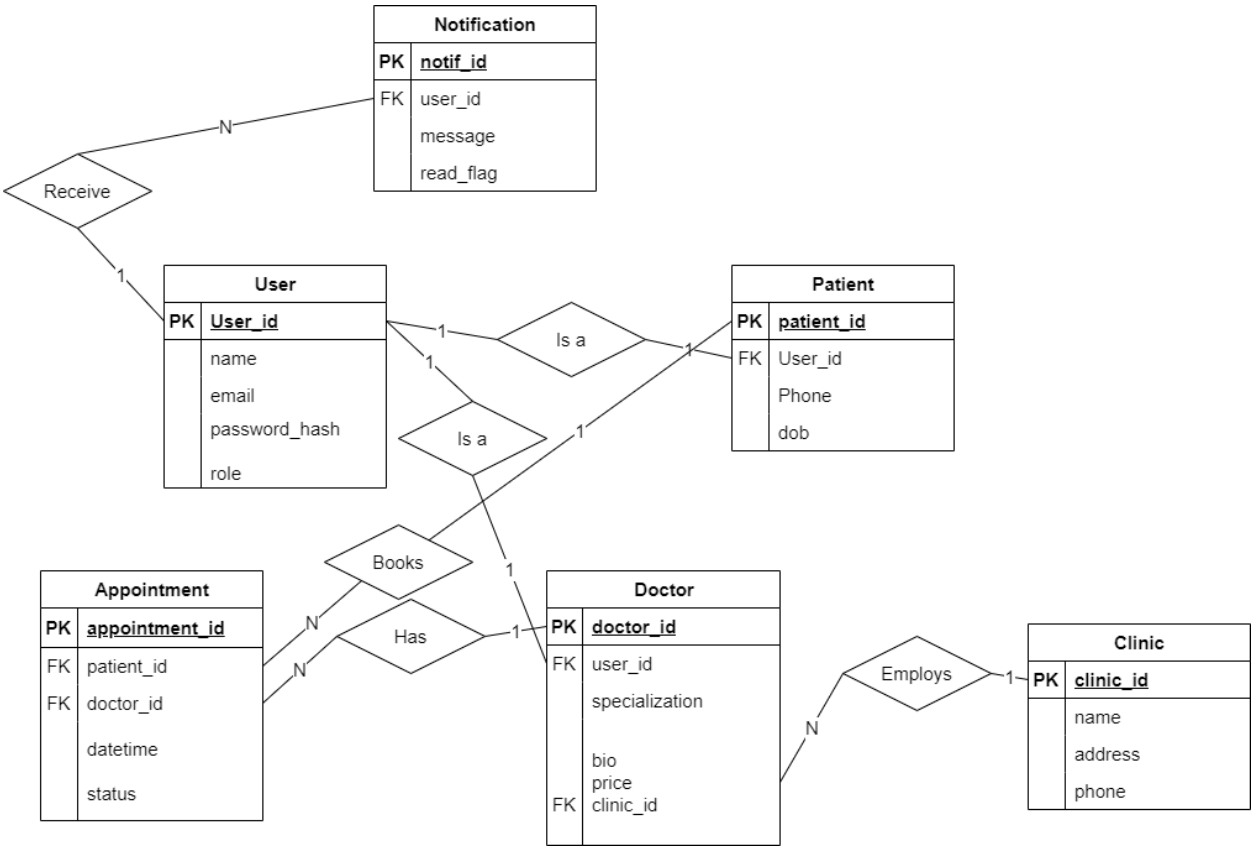
Any additional notes or optional fields relevant to the appointment.

A “Confirm Booking” button to submit the appointment request.

The purpose of this page is to ensure users can verify all details before completing the booking process.

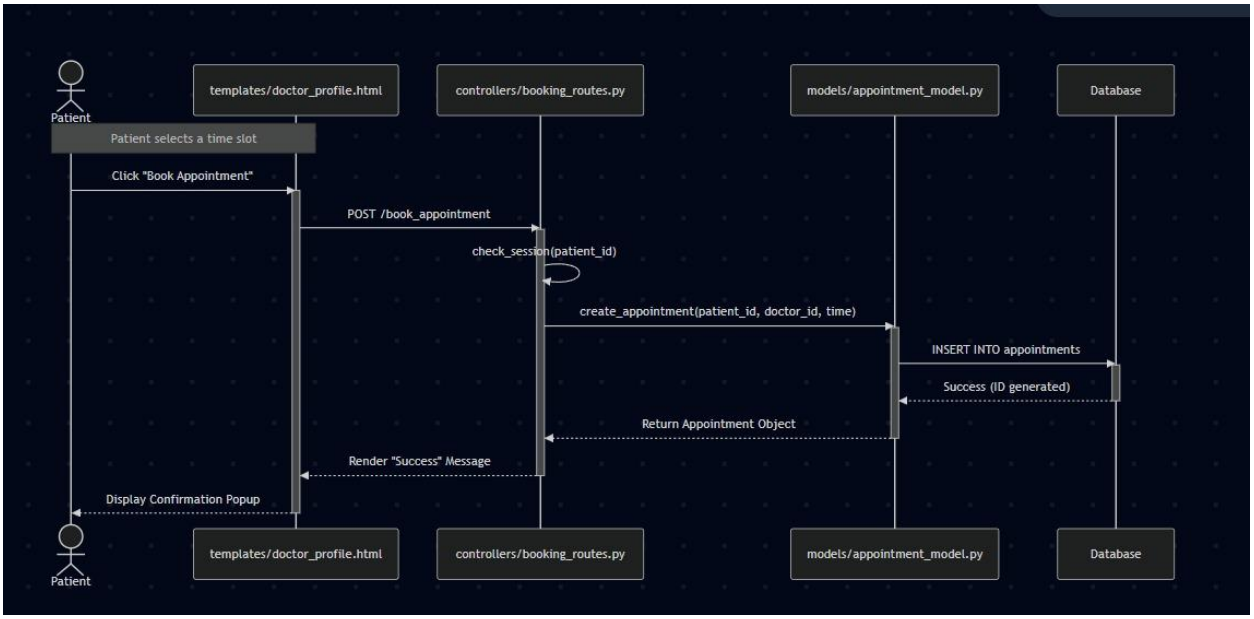
4.2 UML Diagrams

4.2.2 Detailed Class Diagram



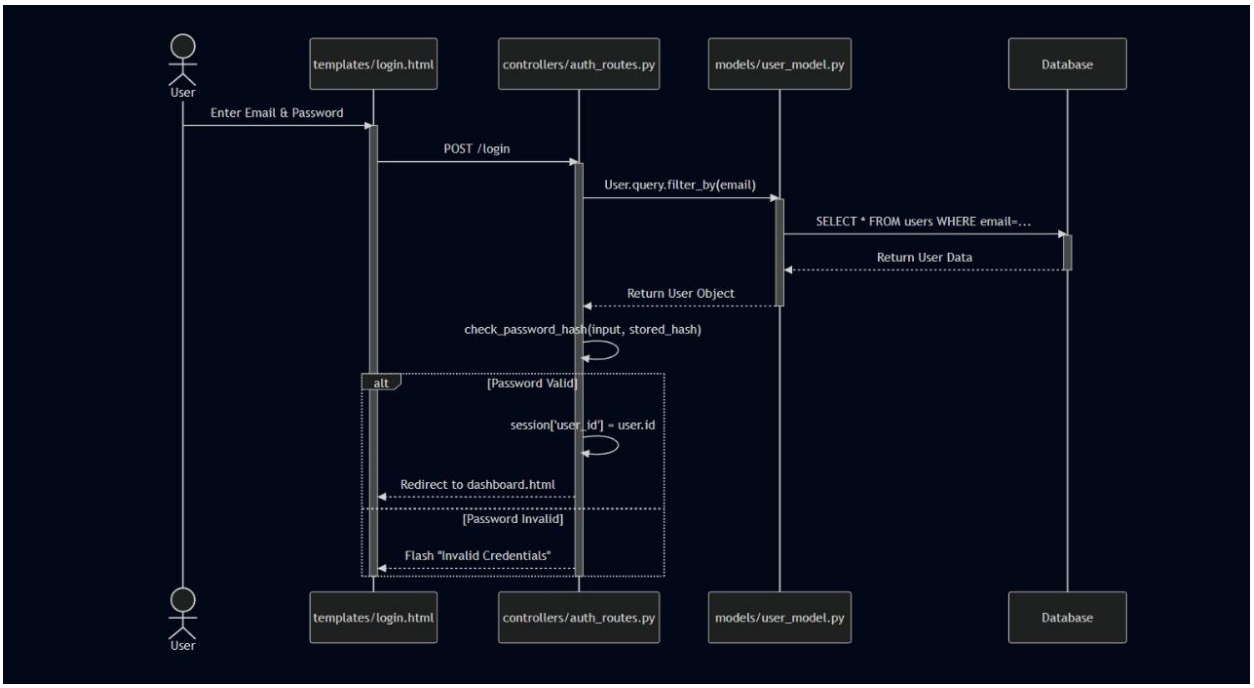
4.2.3 Sequence Diagram

1.Patient Books Appointment



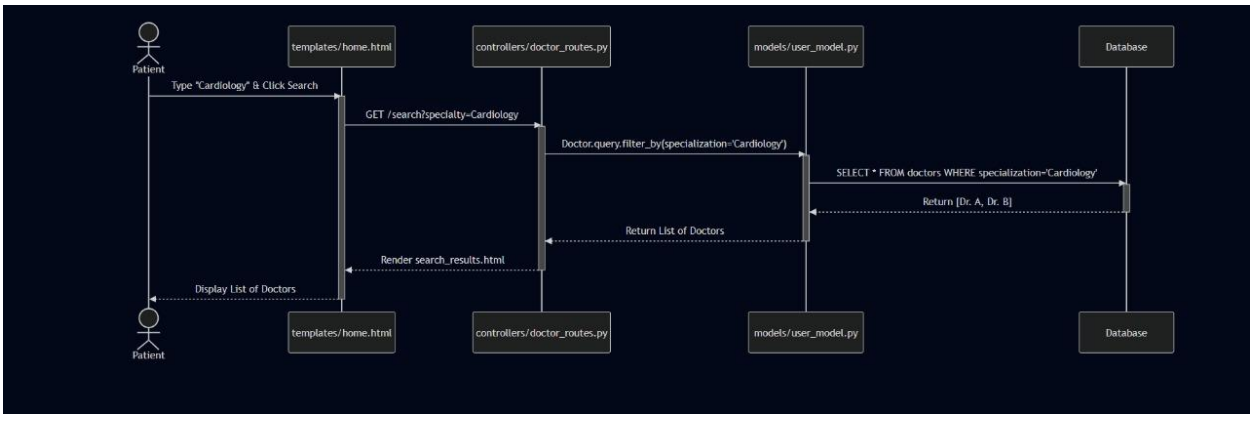
Description:Patient Books Appointment Sequence This sequence illustrates the appointment booking process (FR-05). The interaction begins when a Patient selects a time slot on the doctor_profile.html View. The booking_routes.py Controller receives the POST request, validates the user session, and invokes the create_appointment method in the Appointment Model (appointment_model.py). The Database inserts the new record, and the system renders a success confirmation to the user.

2.User Login



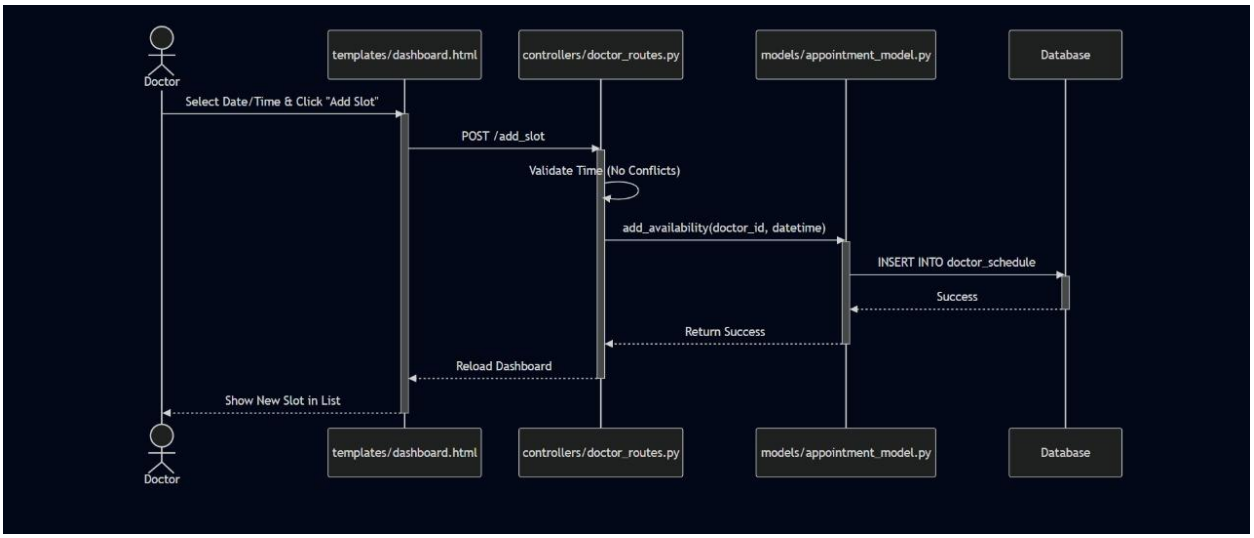
Description: User Login Sequence This diagram depicts the authentication flow for all users (FR-02). The User submits credentials via the login.html View. The auth_routes.py Controller queries the User Model (user_model.py) to retrieve the account associated with the provided email. It then validates the password hash; if successful, a session is established, and the user is redirected to the dashboard.html View.

3. Doctor Search



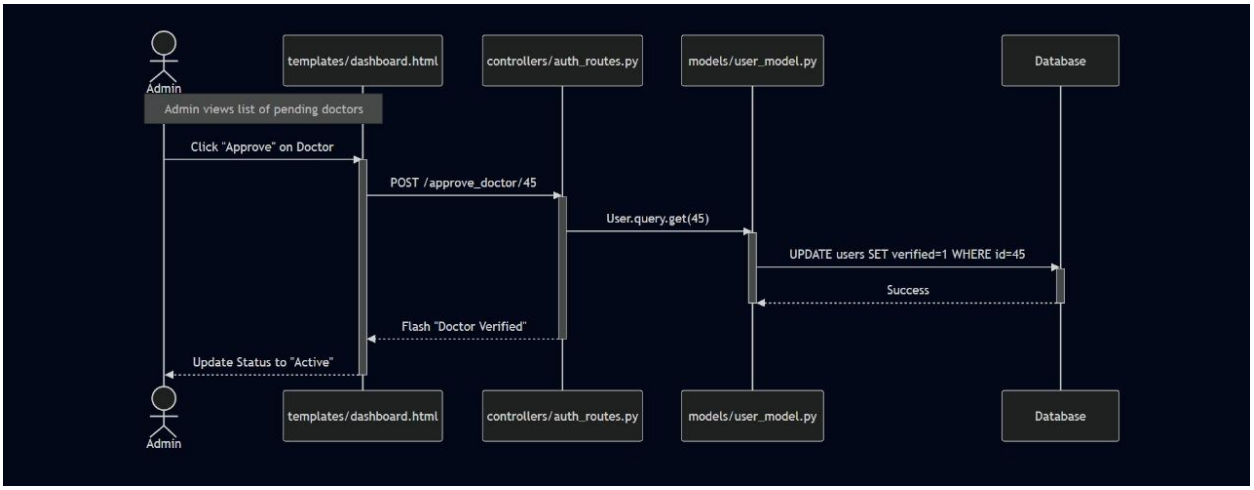
Description: Doctor Search Sequence This sequence demonstrates the search functionality (FR-03). A Patient initiates a search for a specific specialty (e.g., "Cardiology") on the home.html View. The doctor_routes.py Controller processes the GET request and filters the Doctor Model (within user_model.py) based on the specialty. The Database returns matching records, which are passed back to the view to display the search results list.

4. Doctor Adds Available Slot



Description: Doctor Adds Available Slot Sequence This diagram shows how a Doctor manages their schedule (FR-09). The doctor submits a new availability slot via the dashboard.html View. The doctor_routes.py Controller validates the time for conflicts before calling the add_availability method in the Model (appointment_model.py). Upon a successful Database update, the dashboard is reloaded to reflect the newly added slot.

5. Admin Verifies Doctor

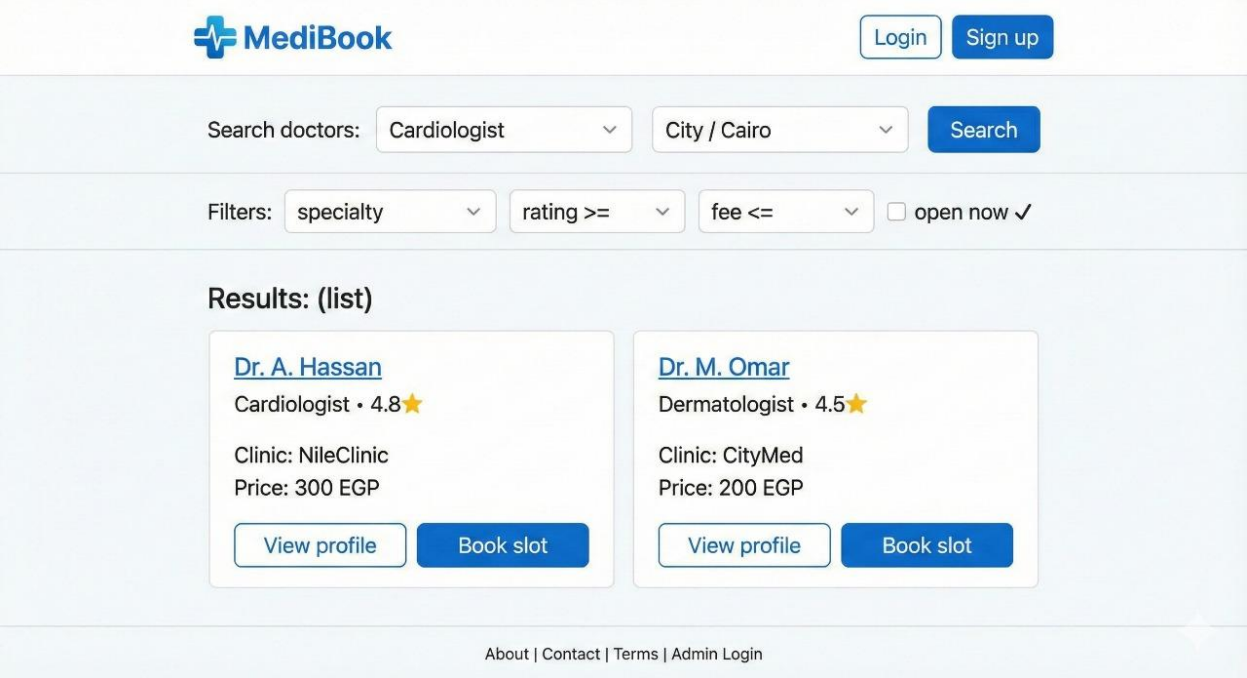


Description: Admin Verifies Doctor Sequence This process outlines the administrative verification of doctor accounts (FR-12). An Admin reviews pending registrations on the dashboard View and triggers an approval action. The auth_routes.py Controller retrieves the specific User Model (user_model.py) and updates the verified attribute to "True" in the Database, enabling the doctor's public profile.

4.3 UI/UX Desgin

4.3.1 Wire frames/mockups

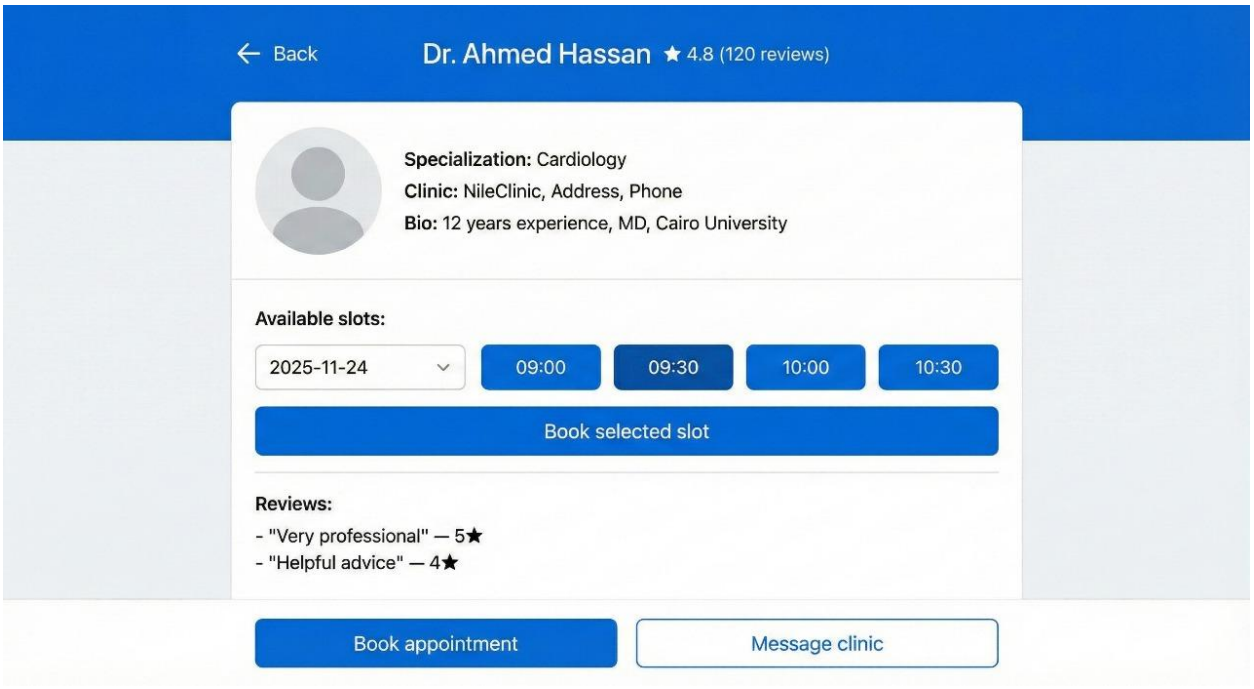
1. Home / Search Page



Description:

A simple landing page that allows users to search for doctors by specialty and location, with additional filter options.
Search results are displayed as doctor cards with quick actions like View Profile and Book Slot.

2. Doctor Profile Page:

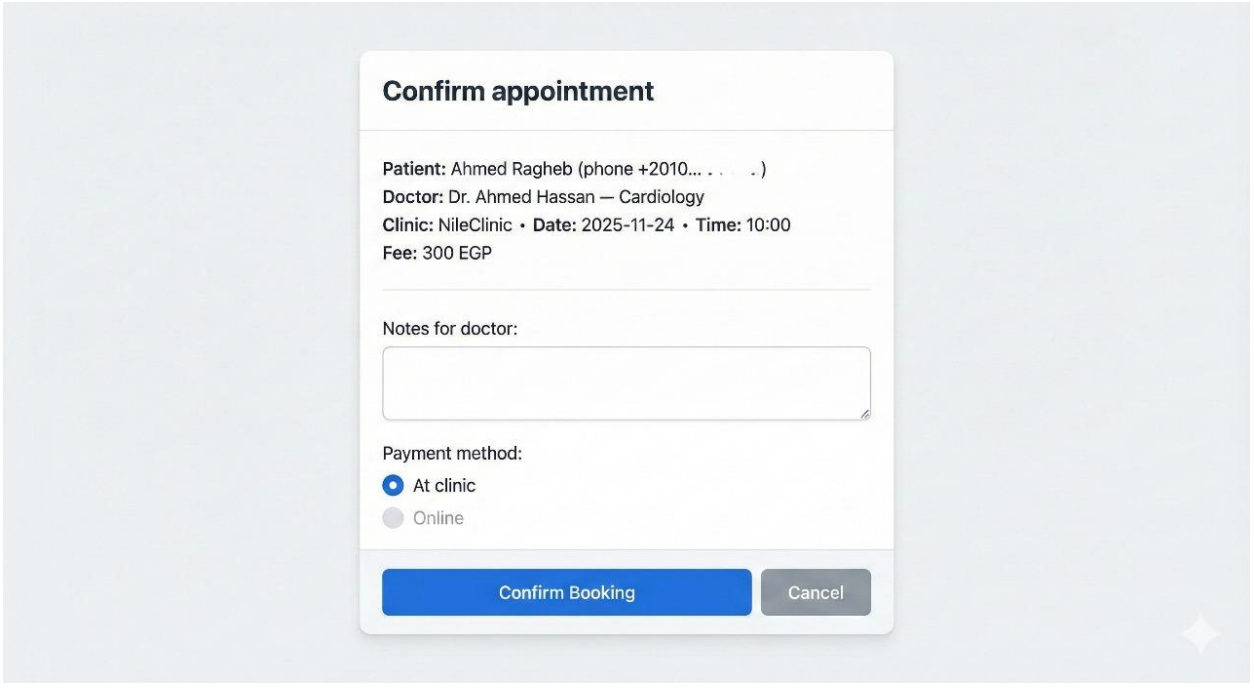


Description:

A detailed doctor page showing specialization, clinic info, bio, reviews, and available time slots.

Users can select a date, choose a slot, and proceed to booking.

3. Confirm Appointment Page:

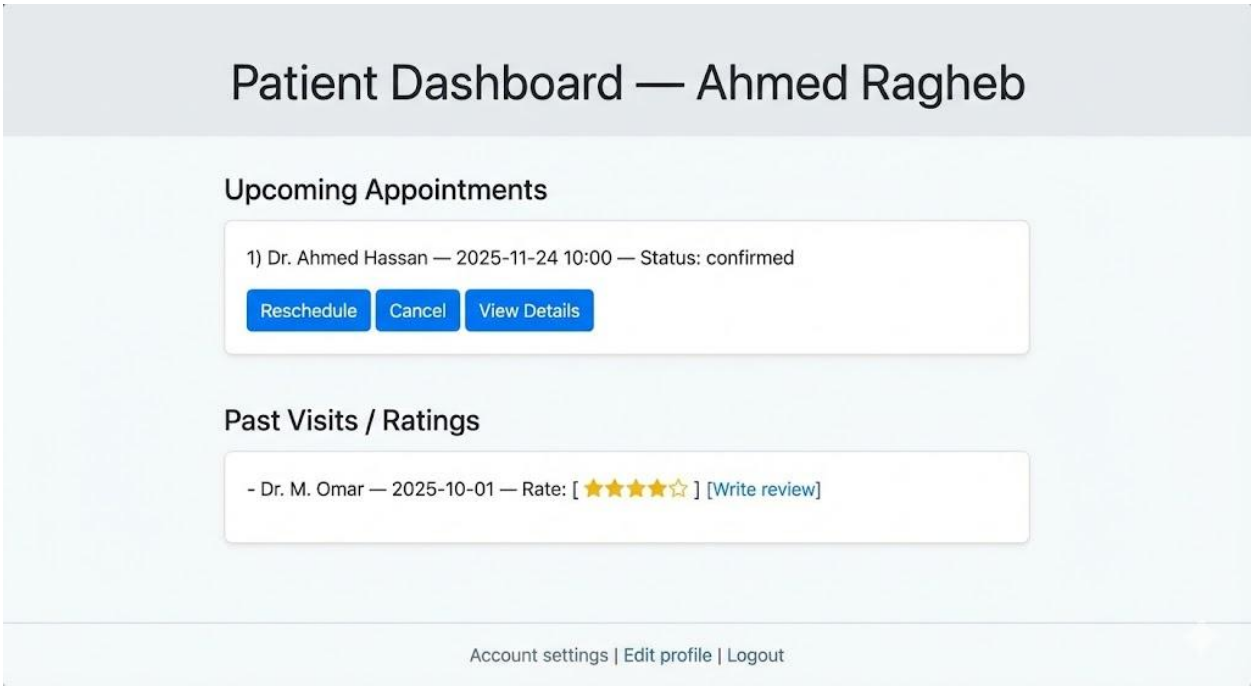


Description:

A confirmation screen summarizing the selected appointment details before finalizing the booking.

It includes a notes section, payment method selection, and buttons to confirm or cancel.

4. Patient Dashboard Page:



Description:

A personal dashboard displaying upcoming appointments with action buttons for rescheduling, canceling, or viewing details.

Past visits are listed with rating options and a link to write a review.

4.4 Data Design

4.4.3 Data Base Schema/ER Diagram

Data base schema

1. Patient Books Appointment

Column Name	Data Type	Description	Constraints
patient_id	Integer	Unique ID for the patient record.	PK, Auto-increment [8]
user_id	Integer	Links the patient to the Users table.	FK (Ref Users), Not Null [9]
dob	Date	Date of Birth.	Not Null, Must be valid date [10]

2. User Login

Column Name	Data Type	Description	Constraints
user_id	Integer	Unique ID for every user (Patient, Doctor, Admin).	PK, Auto-increment, Not Null [2]
email	String	User's email for login and notifications.	Unique, Not Null, Max 100 chars [3]
password_hash	String	Encrypted password string.	Not Null, 60-255 chars [4]
name	String	Full name of the user.	Not Null, 3-100 chars [5]
phone	String	Phone number for contact/SMS.	Unique, Valid format [6]
role	Enum	User type ('patient', 'doctor', 'admin').	Not Null, Default: 'patient' [7]

3. Doctor Search

Column Name	Data Type	Description	Constraints
doctor_id	Integer	Unique ID for the doctor record.	PK, Auto-increment [11]
user_id	Integer	Links the doctor to the Users table.	FK (Ref Users), Not Null [12]
clinic_id	Integer	Links the doctor to their clinic.	FK (Ref Clinics), Not Null [13]
specialization	String	Medical specialty (e.g., Cardiology).	Not Null, from set list [14]
bio	Text	Short professional biography.	Optional, Max 1000 chars [15]
price	Decimal	Consultation fee per visit.	Not Null, $\ge$ 0.00 [16]

4. Doctor Adds Available Slot

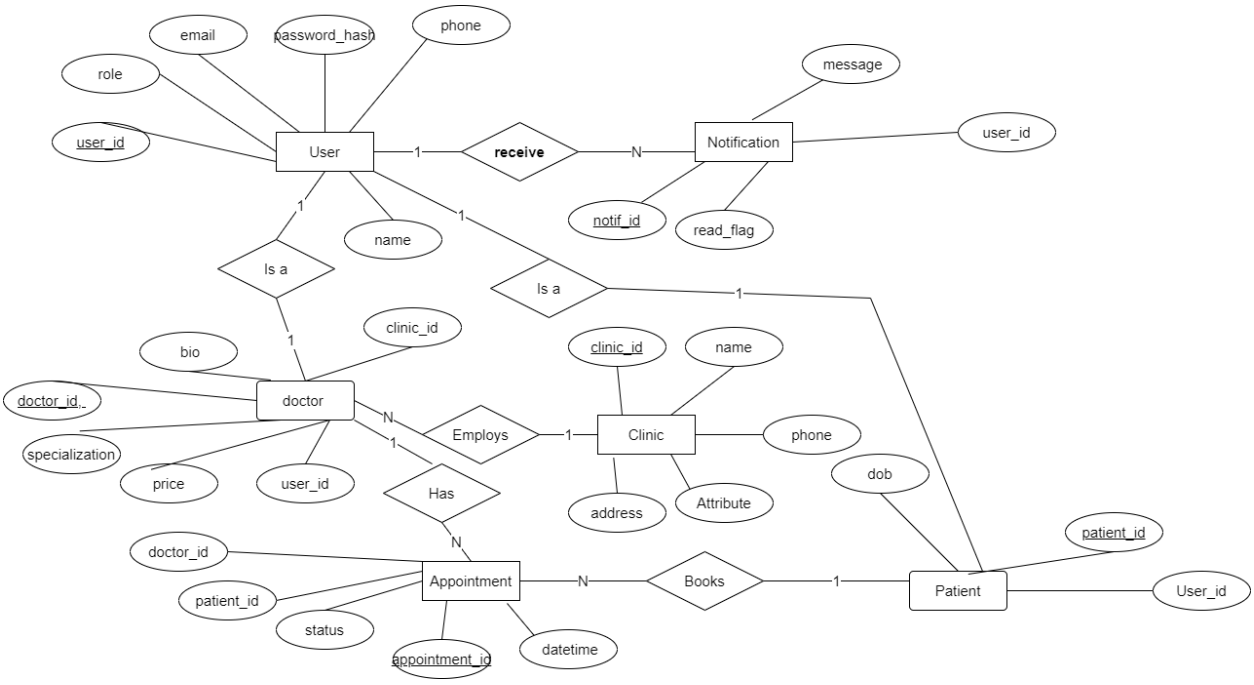
Column Name	Data Type	Description	Constraints
clinic_id	Integer	Unique ID for the clinic.	PK, Auto-increment [17]
clinic_name	String	Name of the clinic or hospital.	Not Null, 3-150 chars [18]
address	String	Physical location address.	Not Null, Max 200 chars [19]
phone	String	Contact number for the clinic.	Optional [20]

5. Admin Verifies Doctor

Column Name	Data Type	Description	Constraints
appointment_id	Integer	Unique ID for the booking.	PK, Auto-increment [21]
patient_id	Integer	The patient who booked the slot.	FK (Ref Patients), Not Null [22]
doctor_id	Integer	The doctor being visited.	FK (Ref Doctors), Not Null [23]
datetime	DateTime	Date and time of the visit.	Not Null, Future date [24]

Column Name	Data Type	Description	Constraints
status	Enum	Booking state (pending, confirmed, cancelled).	Default: 'pending' [25]
payment_method	Enum	How the patient will pay (online, at_clinic).	Not Null [26]
payment_status	Enum	Status of payment (pending, paid).	Default: 'pending' [27]
rating	Integer	Rating given by patient (1-5).	Optional, 1-5 scale [28]
review_text	Text	Written feedback from patient.	Optional, Max 500 chars [29]

ER Diagram



5. Conclusion

5.1 Summary of Design Phase

The Design Phase for the MediBook Doctor Appointment System has successfully defined the technical blueprint required for implementation. We have established a Monolithic Architecture utilizing Python Flask for the backend and HTML/CSS for the frontend, ensuring a robust and unified codebase.

Key design decisions and outputs include:

Adoption of the MVC Pattern: We have clearly mapped the system components into Models (SQLAlchemy database tables), Views (Jinja2 templates), and Controllers (Flask routes) to ensure separation of concerns and maintainability.

Comprehensive Data Design: A relational database schema has been defined (including Users, Doctors, Patients, and Appointments), supported by a detailed Data Dictionary and Entity-Relationship (ER) Diagram to manage data integrity.

Interaction Modeling: Through five critical Sequence Diagrams (Booking, Login, Search, Scheduling, and Verification), we have visualized exactly how data flows between the user interface and the server.

User Interface Strategy: We have outlined the necessary wireframes and file structures to support a user-friendly experience for both patients and doctors.