Full Stack Development with MERN

Project Documentation

Book a Doctor App

1. Introduction

Project Title: Book a Doctor App

Description: The **Book a Doctor App** is a comprehensive full-stack solution built using the MERN (MongoDB, Express.js, React, Node.js) stack. It is designed to modernize the healthcare appointment process, offering a secure and scalable platform where patients can easily find and book appointments with verified doctors. The system provides role-based dashboards for patients, doctors, and administrators to manage registration, scheduling, and platform governance efficiently.

Team Members: Manoj Kumar E

2. Project Overview

Purpose: The primary goal of the Book a Doctor App is to provide a modern, efficient, and scalable platform for connecting patients with verified healthcare professionals, facilitating online appointment booking, and managing medical records securely.

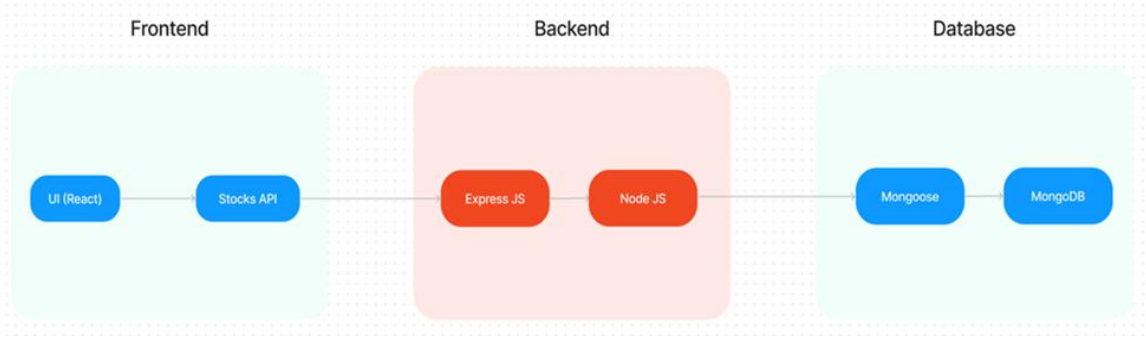
Features:

1. **Patient Registration & Profile Creation:** Secure sign-up using email and password, and profile creation for storing personal and medical information.
2. **Doctor Browsing & Filtering:** Ability to search and filter doctors based on specialty, location, and live availability.

3. **Appointment Booking & Management:** A user-friendly interface for selecting dates/times, uploading documents, and receiving automated confirmations/reminders.

4. **Doctor's Dashboard:** Tools for doctors to manage their availability, view bookings, and securely access/update patient records.

5. **Admin Controls:** Features for admins to approve doctor registrations, oversee the platform, enforce policies, and manage users with **Role-based Access Control (RBAC)**.



3. Architecture

Frontend: The frontend utilizes a client-server model and employs **Tailwindcss** and **Material UI** to create a responsive and modern user interface. **Axios** is used for seamless API communication with the backend.

Backend: The backend is powered by **Express.js** running on **Node.js**. It follows the **Model-View-Controller (MVC)** architectural pattern to ensure separation of concerns, scalability, and maintainability.

> **Key Technologies:** Authentication is secured using **JWT** for session management and **bcrypt** for password hashing. **Moment.js** manages date and time functionalities for accurate scheduling.

Database: **MongoDB** provides scalable data storage for all application data, including user profiles, doctor details, and appointments. **Mongoose** is used to

implement the Model layer, providing a schema-based solution for data modeling and database operations.

4.ER Diagram

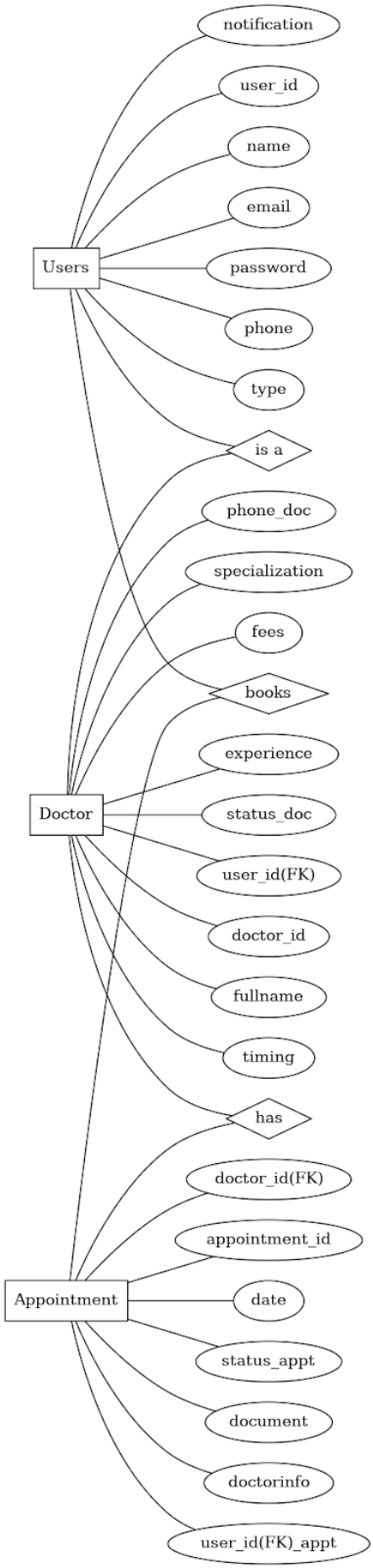
The Entity-Relationship (ER) diagram for the Book a Doctor app represents three key entities: Users, Doctors, and Appointments, with their respective attributes and relationships.

The Users collection holds basic user information, including _id, name, email, notification, password, isdoctor (to differentiate between patients and doctors), type, and phone. The isdoctor field identifies users who are doctors, while others are treated as patients or admins.

The Doctors collection stores information specific to doctors, such as their _id, userID (acting as a foreign key referencing the Users collection), fullname, email, timings, phone, address, specialisation, status, experience, and fees. The userID` links each doctor to their corresponding user account.

The Appointments collection stores details about appointments, including the _id, doctorInfo (foreign key referencing the Doctors collection), date, userInfo (foreign key referencing the Users collection), document (medical records or other files), and status (e.g., pending, confirmed). This collection maintains the relationship between users and doctors for each appointment.

The relationships are as follows: one User can be linked to one Doctor (one-to-one), a User can have multiple Appointments (one-to-many), and a Doctor can handle multiple Appointments (one-to-many). The foreign keys userID in the Doctors collection and doctorInfo and userInfo in the Appointments collection establish these connections, enabling the app to manage the interactions between patients and doctors effectively.


notification
user_id
name
email
Users
password
phone
type
is a
phone_doc
specialization
fees
books
experience
Doctor
status_doc
user_id(FK)
doctor_id
fullname
timing
has
doctor_id(FK)
appointment_id
Appointment
date
status_appt
document
doctorinfo
user_id(FK)_appt

5. Setup Instructions

Prerequisites:

Node.js (LTS recommended)

MongoDB

A code editor (e.g., VSCode)

Installation:

Clone the Repository: (Insert Git clone command here)

Server Setup:

Navigate to the Server folder in your terminal.

Initialize a Node.js project: npm init -y.

Install dependencies: npm install (once you have defined packages).

Create a .env file for environment variables (e.g., MongoDB connection string, JWT secret).

Client Setup:

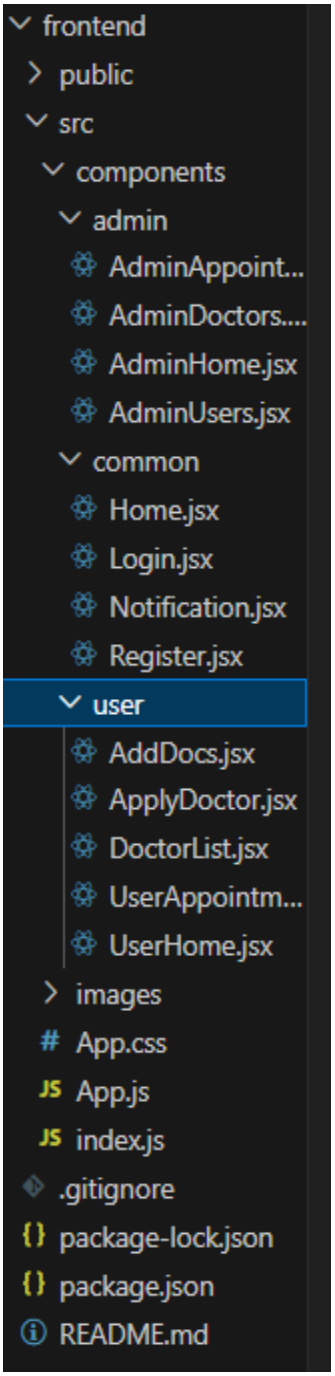
Navigate to the Client folder.

Create a React app: npm create vite@latest . -- --template react (Select React and JavaScript variant).

Install dependencies: npm install.

6. Folder Structure

Client: The React frontend structure includes src, with separate folders for components, which are further categorized into admin, common, and user views. This organization helps separate components based on user role.



Server: The Node.js/Express.js backend is organized following the MVC pattern.

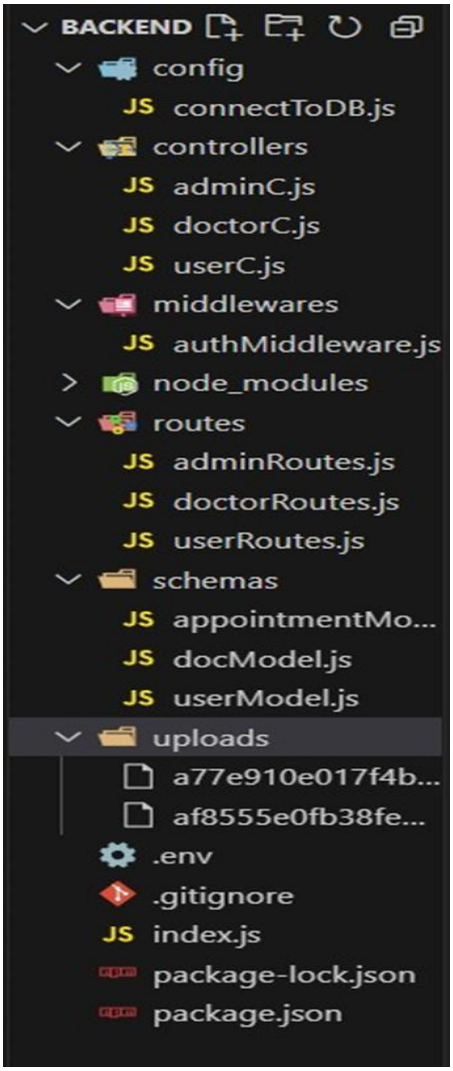
config: Contains database connection logic (connectToDB.js).

controllers: Contains the logic for adminC.js, doctorC.s, and userC.js.

middlewares: Includes authentication middleware (authMiddleware.js).

routes: Defines the API endpoints for adminRoutes.js, doctorRoutes.js, and userRoutes.js.

schemas (Models): Contains the Mongoose schemas, such as appointmentModel.js, docModel.js, and userModel.js.

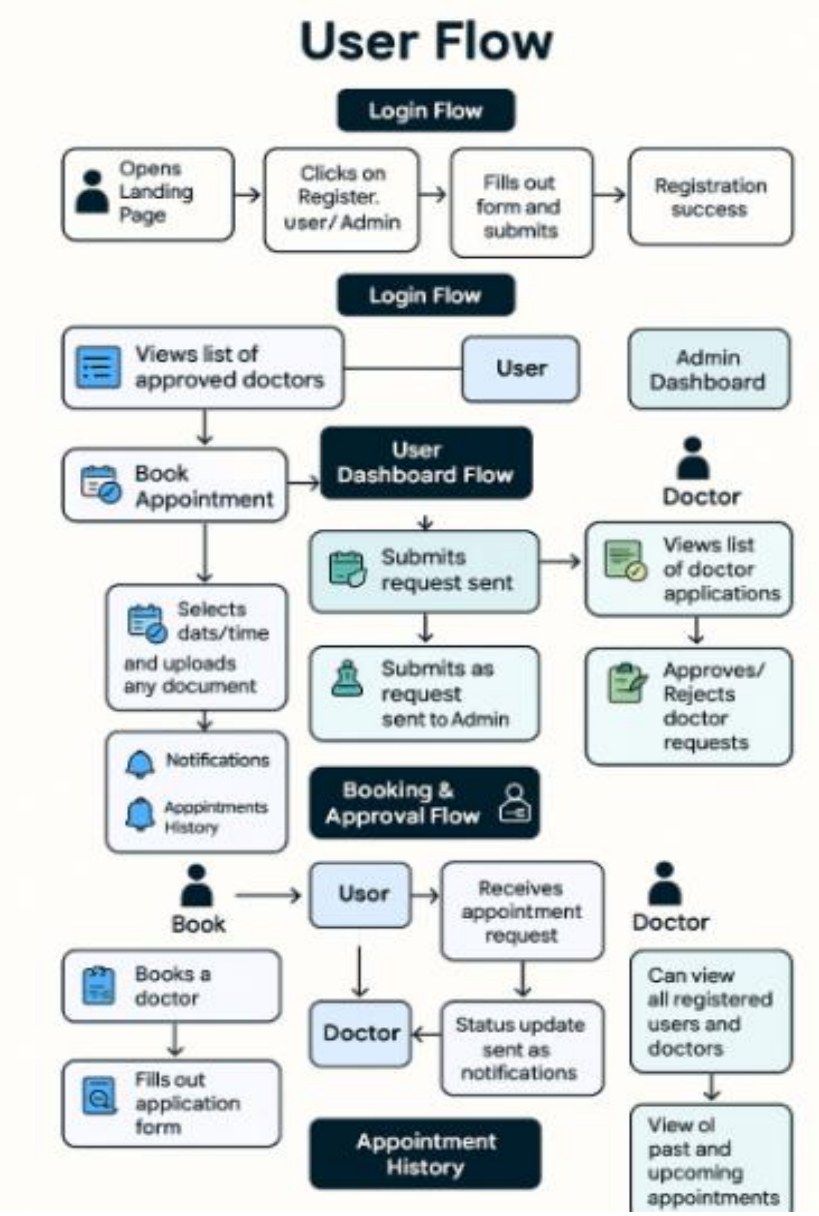


7. Running the Application

Frontend: Navigate to the client directory and run: npm run dev (or npm start if configured).

Backend: Navigate to the server directory and run: npm start (This command assumes you have configured a start script in package.json).

User Flow:



8. API Documentation

Endpoint	Method	Description	Parameters (Request Body/Query)	Example Response
/api/users/register	POST	Registers a new user/patient.	name, email, password, phone	Success message & JWT token
/api/doctors	GET	Fetches a list of approved doctors.	specialty (query param for filtering)	Array of doctor objects
/api/appointments/book	POST	Books a new appointment.	doctorId, date, time, document.	Confirmation status

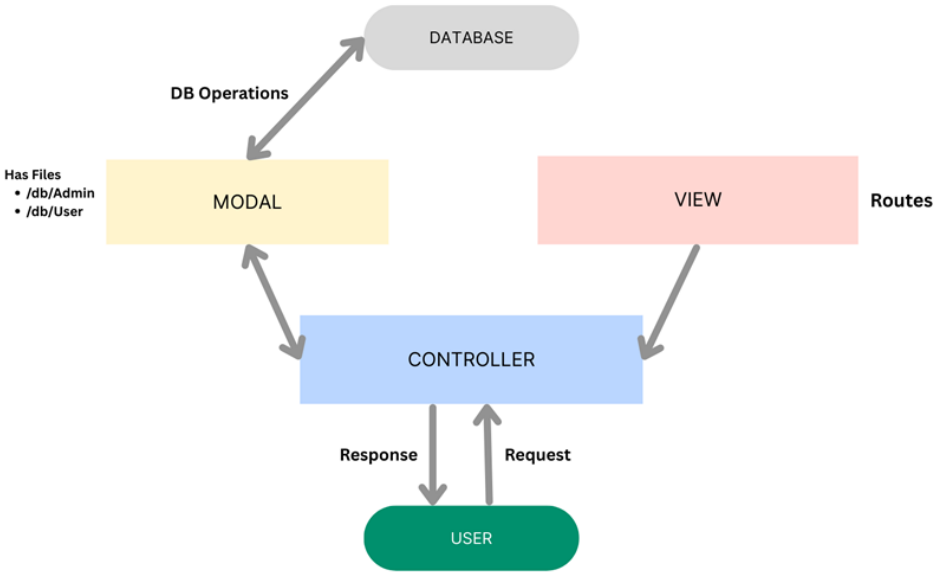
9. Authentication

The application implements a secure authentication and authorization system:

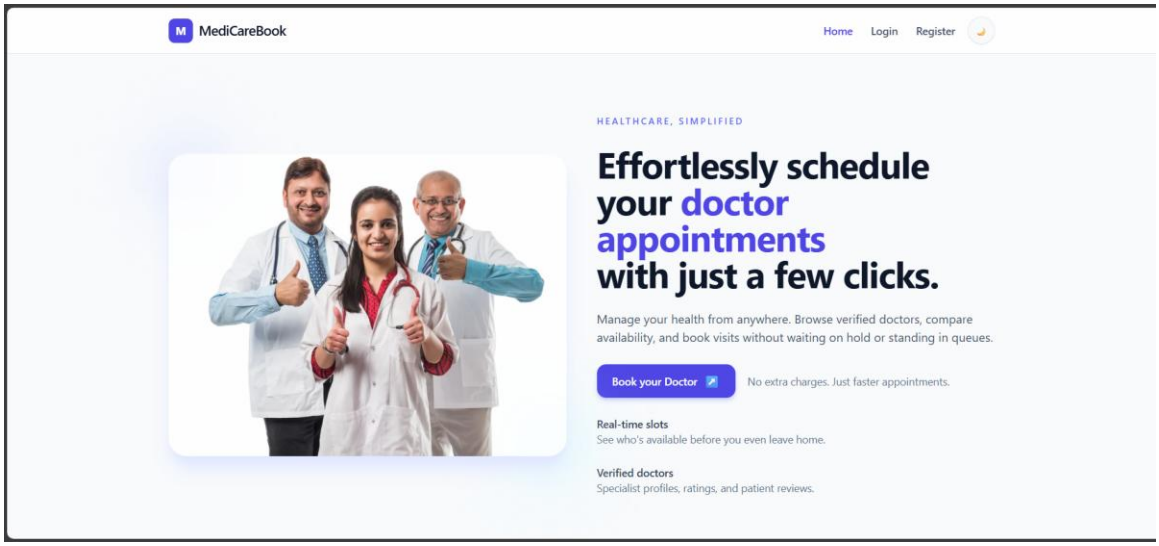
User Authentication: Users sign up with an email and password. Passwords are securely stored using **bcrypt** hashing.

Session Management: After successful login, a **JSON Web Token (JWT)** is issued to the user for session management. This token is sent with subsequent requests to verify the user's identity.

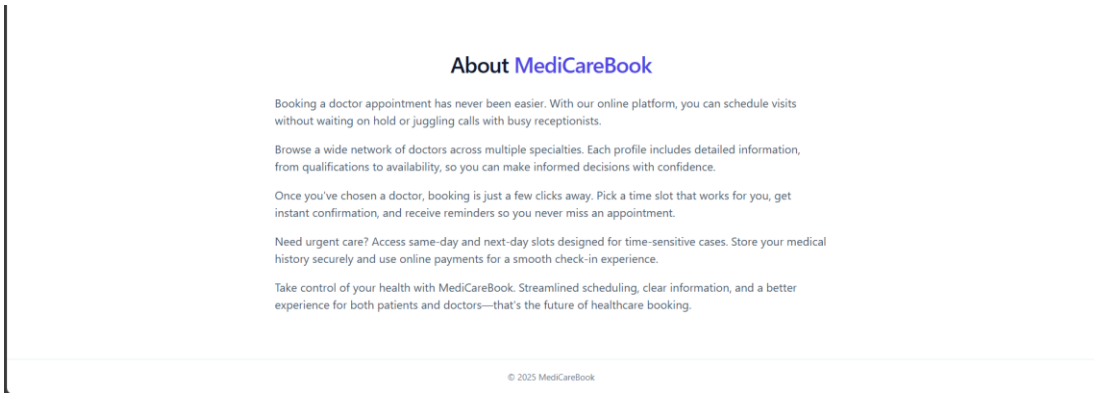
Role-based Access Control (RBAC): Authorization is handled using RBAC. The isdoctor field in the Users collection differentiates between patients and doctors. The server uses middleware to check the user's role before allowing access to role-specific routes (e.g., admin approvals, doctor dashboard updates).



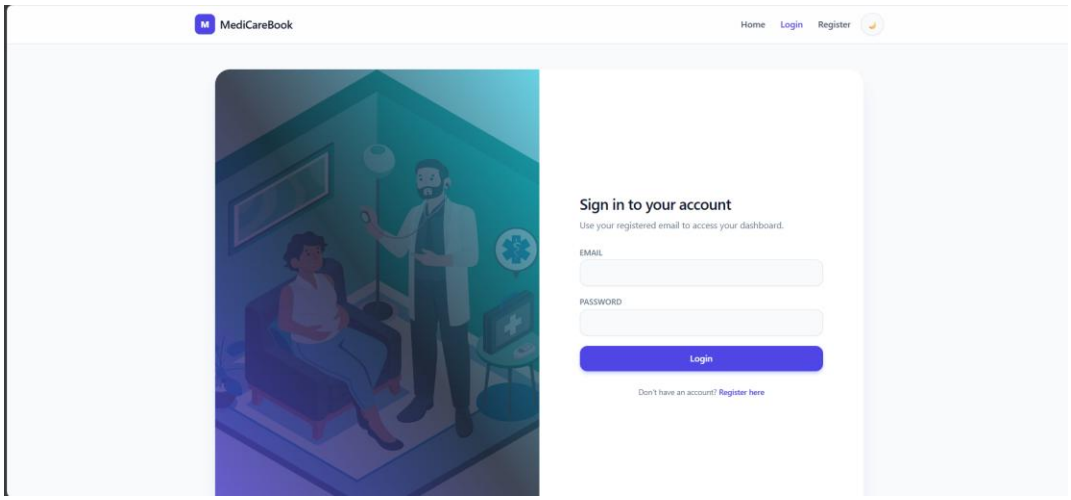
10. Screenshots or Demo



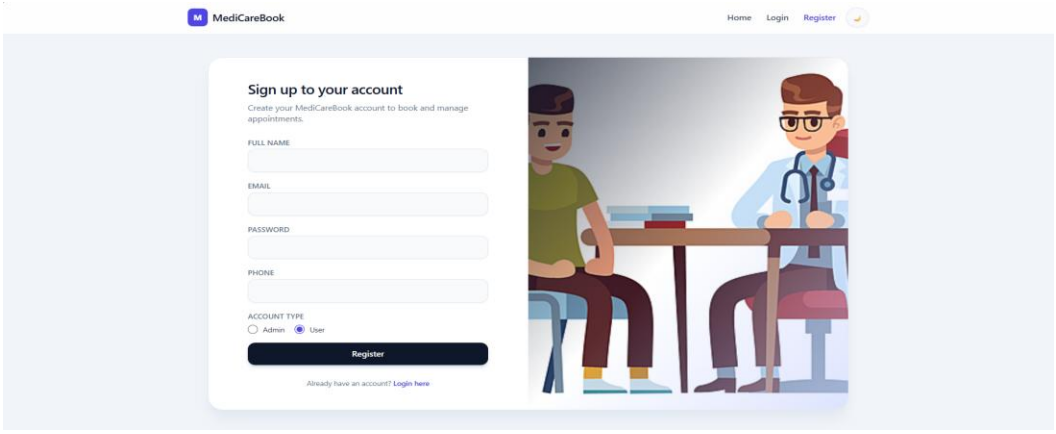
Landing Page



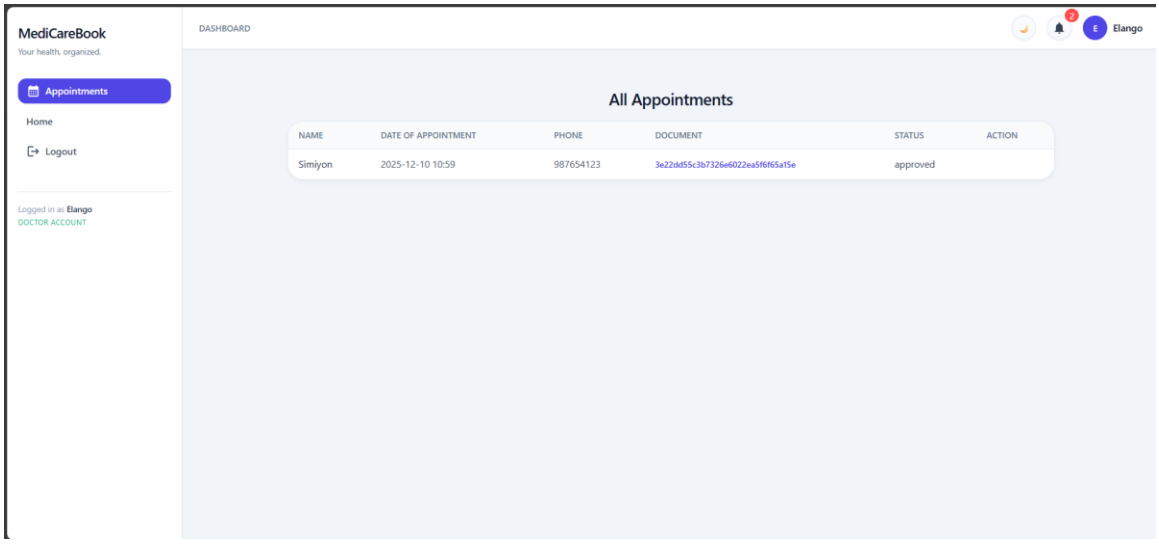
About Page



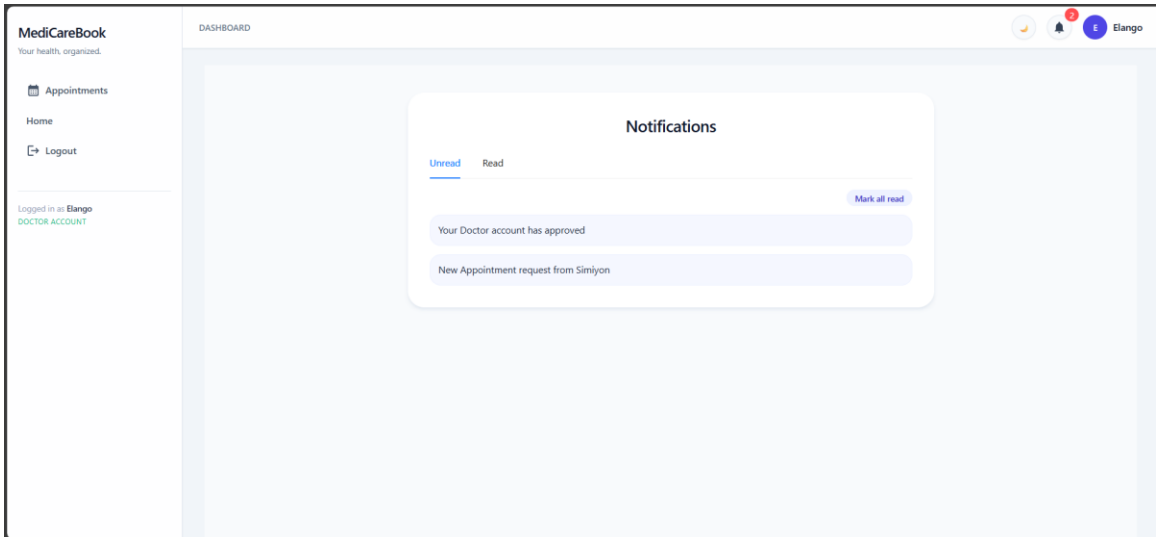
Login Page



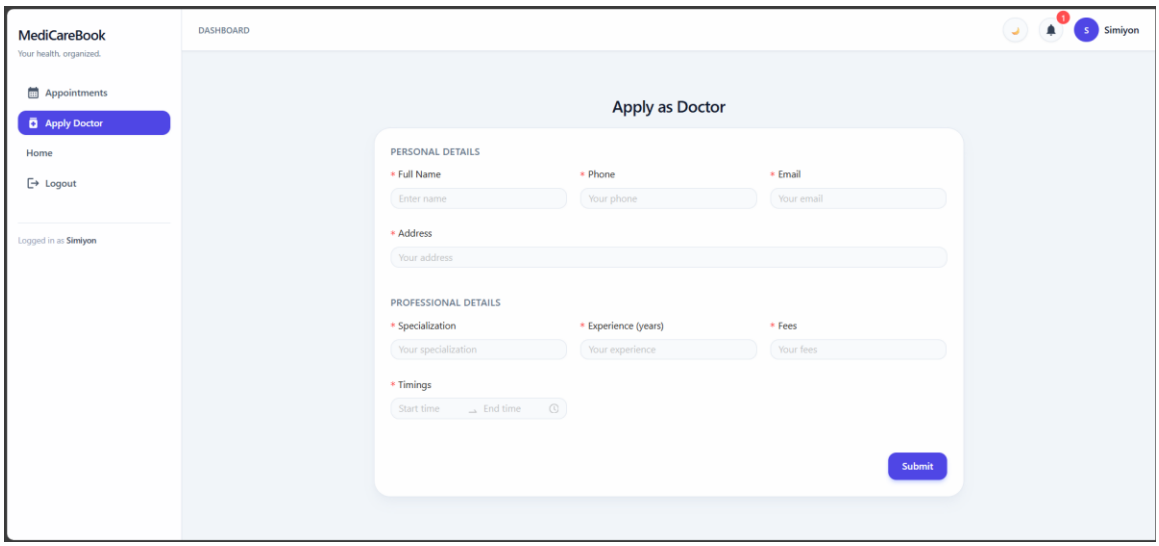
Sign-up Page



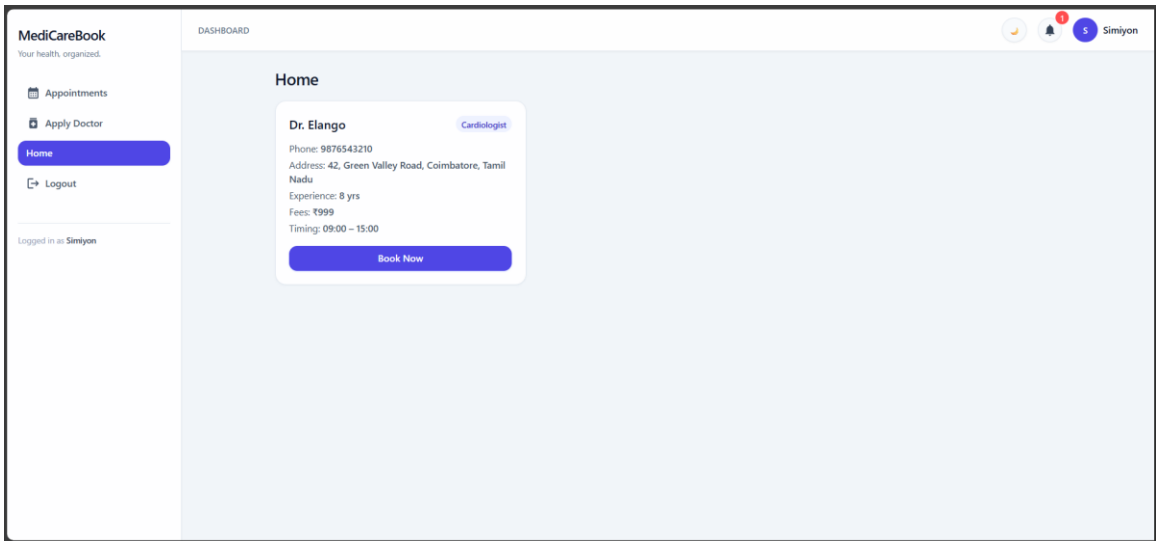
Doct Appointment page



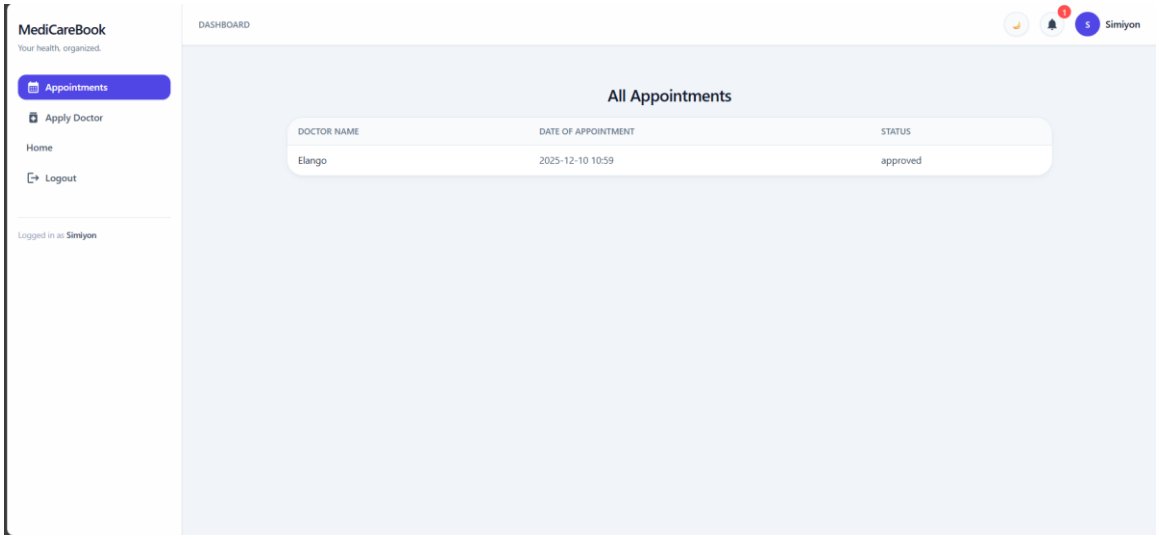
Doct Notification page



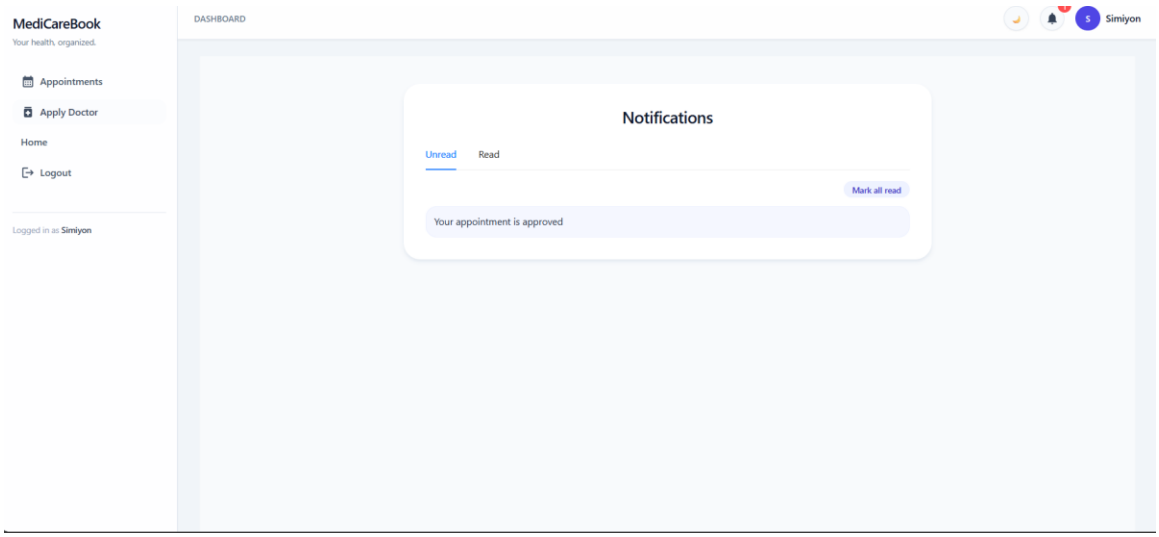
Apply a Doct Page



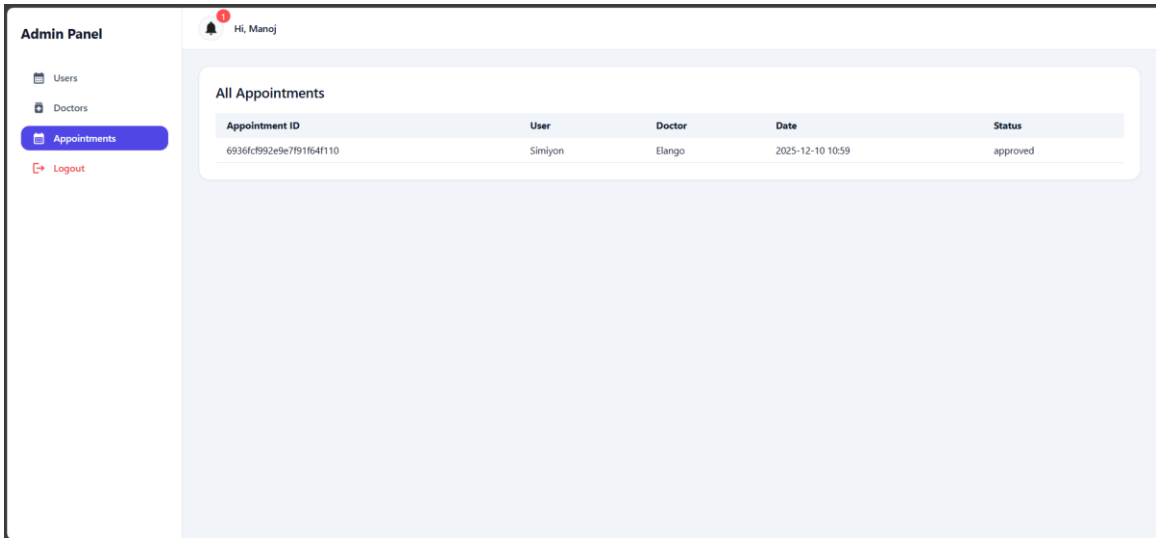
Patient Home Page



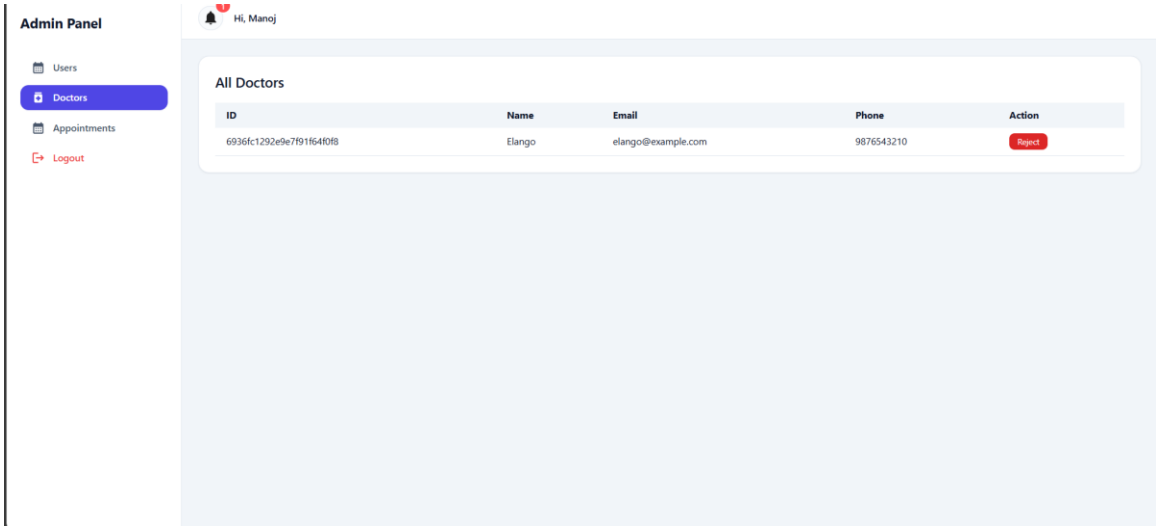
Patient Appointment Page



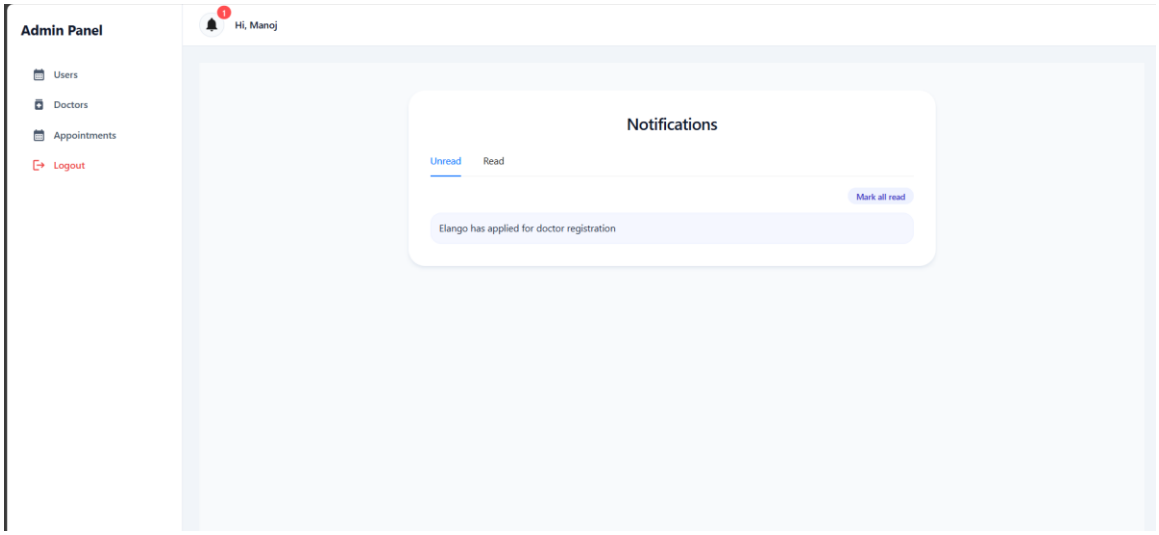
Patient Notification Page



Admin Appointment page



Admin Doctors List Page



Admin Notification

11. Future Enhancements

Implementing payment gateway integration for appointment fees.

Developing an integrated video consultation feature.

Expanding filtering options for doctors (e.g., by ratings/reviews).