Docspot: seamless appointment booking for health

# Introduction

**Project Title:** Docspot: seamless appointment booking for health

**Team Members and Their Roles:**

Member-1: - G Eswar Sai Sandeep - [Backend Development & Project Implementation & Execution]

Member-2: - B Devi – [Frontend Development & Database Development]

Member-3: - K Jaya Shankar Sai– [Project Setup and Configuration & Frontend Development]

Member-4: - K Sankar – [Backend Development & Project Setup and Configuration]

# Project Overview

## Purpose

DocSpot is a MERN-based appointment booking application designed to connect users with multiple healthcare providers. It allows patients to explore available doctors and clinics, book appointments, and track them conveniently. For healthcare providers, it offers a platform to manage schedules and process appointments, while administrators oversee operations.

## Features

->Features multiple healthcare provider listings with a wide range of specialties and services.

->Allows users to enter appointment and payment details for their bookings.

->Ensures a safe and streamlined transaction experience for patients.

->Provides confirmation and detailed information after an appointment is booked.

->Enables administrators to manage providers and appointments efficiently.

# Application Flow

## User Flow

-Register for an account and log in.

-Browse doctors and add appointments to the booking cart.

-Enter appointment and payment details to complete the booking.

-Track appointment history through the profile section.

## HealthCare Provider Flow

-Authenticate and request admin approval.

-List, update, and manage appointment slots and services.

## Admin Flow

1. Log in to the admin dashboard.
2. Manage users, healthcare providers, appointments, and services

# Architecture Frontend

->Built with React.js, the frontend delivers a responsive interface for users and healthcare providers.

->Pages include doctor listings, booking cart, checkout, appointment history, and dashboards.

## Backend

Developed using Node.js and Express.js, the backend handles core logic including user authentication, appointment handling, booking processing, and database communication.

## Database

Uses MongoDB for storing all structured data.

Key Schemas:

user Schema: User details (username, email, password).

Provider Schema: Healthcare provider information (name, specialty, description).

Appointments Schema: Appointment records (user Id, provider Id, date, time).

Cart Schema: Temporary booking cart data linked to a user.

Admin Schema: Admin data (categories, promoted providers).

Healthcare Provider Schema: Healthcare provider data including profile and service catalogue.

# Setup Instructions

## Prerequisites

* Node.js (v14+)
* MongoDB (Compass or Atlas)
* npm or yarn

## Installation

* Create project folder and initialize package. Json.
* Install dependencies: npm install express mongoose dotenv cors body-parser or yarn add express mongoose dotenv cors body-parser.
* Create index.js and set up Express server.
* Configure MongoDB and environment variables in a .env file (e.g., MONGO\_URI=your\_mongodb\_url).

# Folder Structure

## Client

* src/components: React components for UI
* src/pages: Individual pages (Home, Booking Cart, Checkout)
* src/context: State management using Context API

## Server

* server/index: Route handlers for users, providers,appointments, admin.
* server/Schema: Mongoose schemas and models
* server/index: Logic for handling API calls
* server/: Authentication and error handling logic

# Running the Application

The project uses concurrently to streamline development by running both frontend and backend servers in parallel from the root directory.

## To Start the Full Stack Application

• Run: npm start

o This command triggers scripts defined in the root package. Json, simultaneously

launching:

▪ **Frontend**: React app located in /client

▪ **Backend**: Express server located in /server

o No need to navigate into individual folders manually—everything is wired up for

convenience and efficiency.

# API Documentation

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Endpoint** | **Description** | |
| **POST** | /api/register |  | Register a new user with email and password. performs validation and hashes passwords using p byscript. |
| **POST** | /api/login | h | Authenticates a user and returns a JWT token. |
| **GET** | /api/products | i | Retrieves a list of available products. Supports |
|  |  | r | filtering and pagination (optional). |
| **POST** | /api/orders |  | Submits a new order. Requires authentication. |
| **GET** | /api/orders/:id | . | Fetches details for a specific order by ID. Secured e accessible by order owner or admin. |

# Authentication

User authentication is managed via **JSON Web Tokens (JWT)**. The backend issues a token upon successful login, which must be included in the Authorization header of protected requests.

**Access Control:**

* Regular Users: Can view healthcare providers, book appointments, and view their own appointments.
* Admins: Have extended privileges like viewing all appointments and managing providers and services.

# User Interface

The frontend is built with React, focusing on simplicity and responsiveness. Key features include:

* **Navigation**: Seamless transitions between views using React Router.
* **Pages**:
  + Home
  + Healthcare Provider Listing
  + Service Items
  + Booking Cart
  + Checkout
  + Admin/User Dashboards

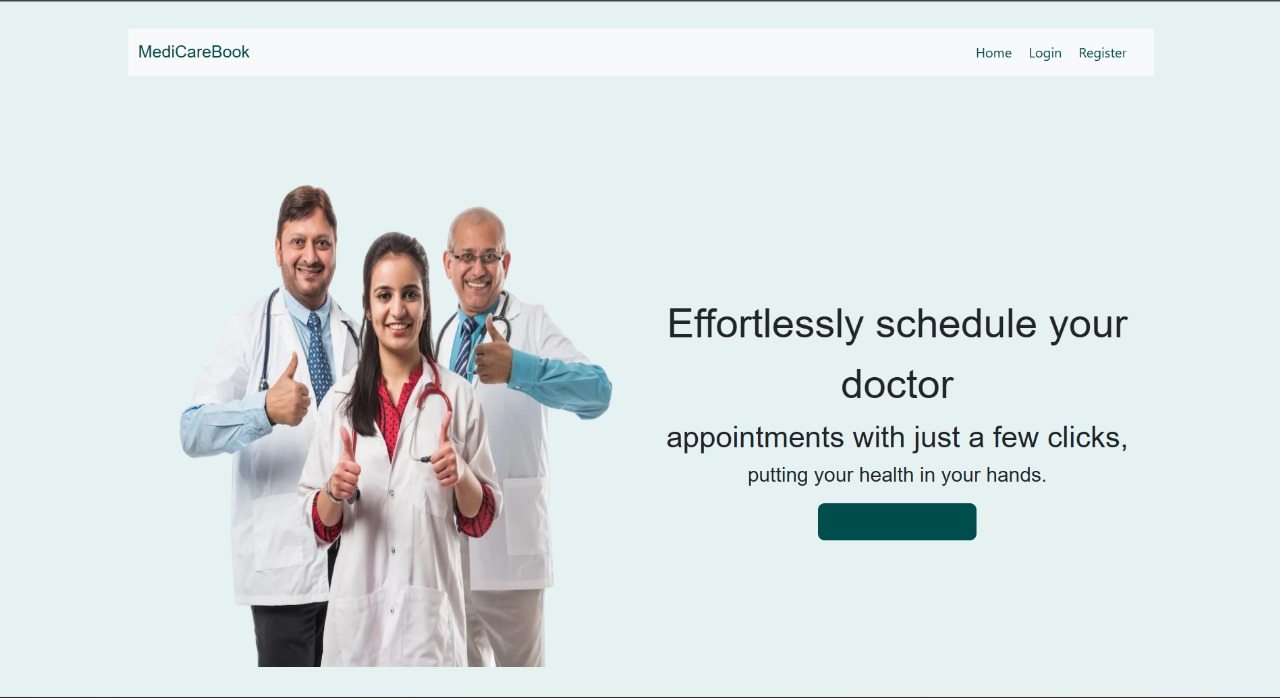
The layout is responsive and styled with Bootstrap, ensuring an optimal experience across devices.

# Testing

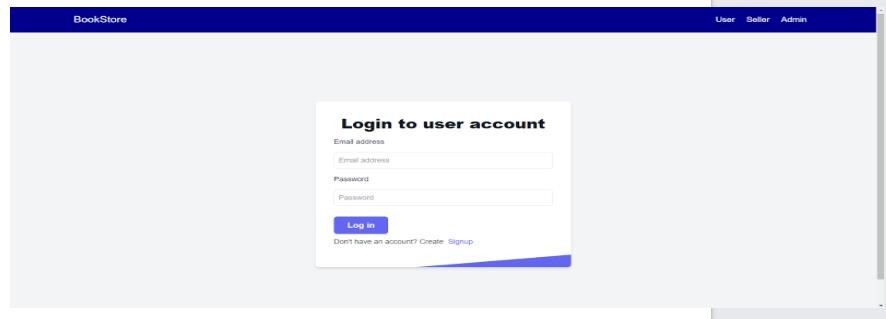
* **Manual Testing**: Conducted during development by interacting with the frontend and verifying backend responses.
* **Postman**: Used to test and document APIs, verify headers, request bodies, and token validity.
* **Future Enhancement**: Add unit and integration tests using frameworks like Jest or Mocha.

# Screenshots or Demo

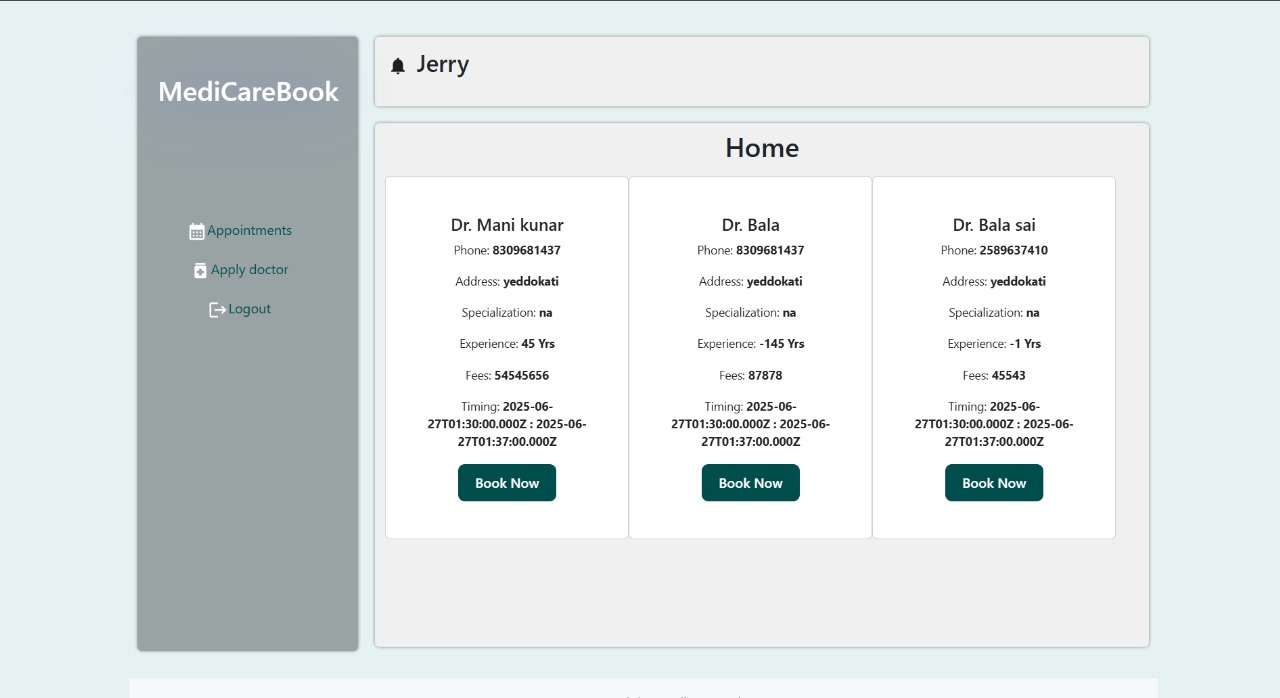
* **Landing page**



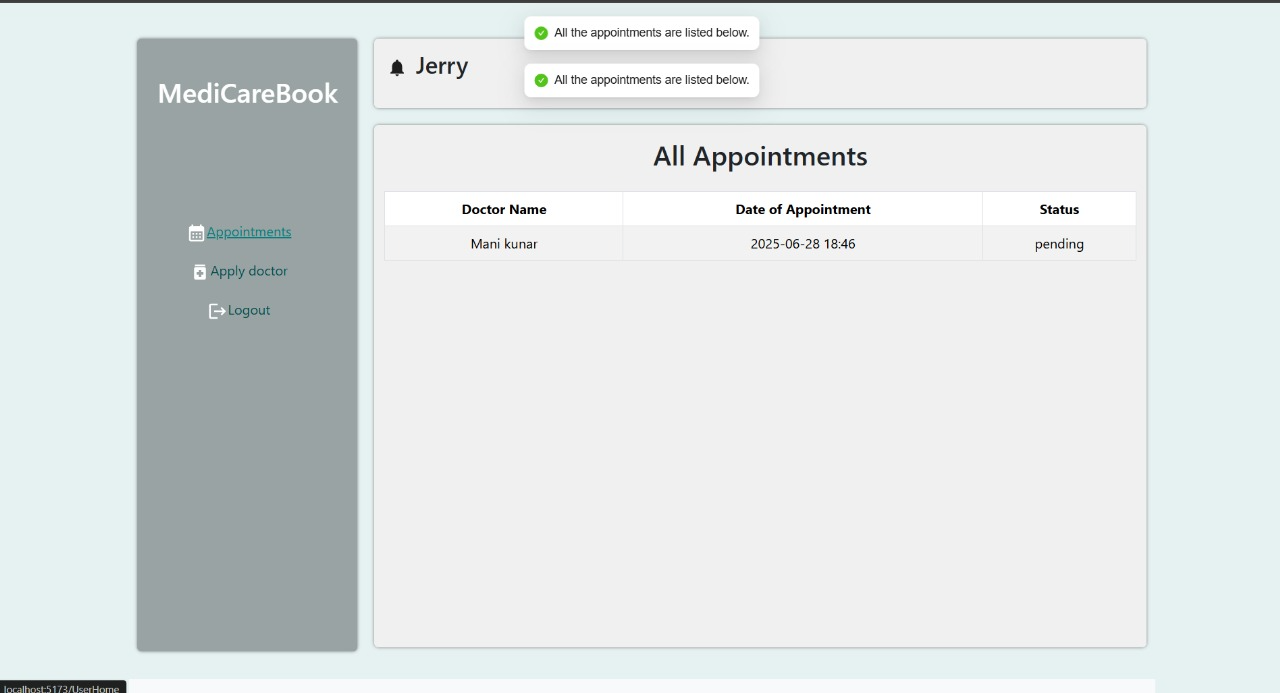
* **Login Page**



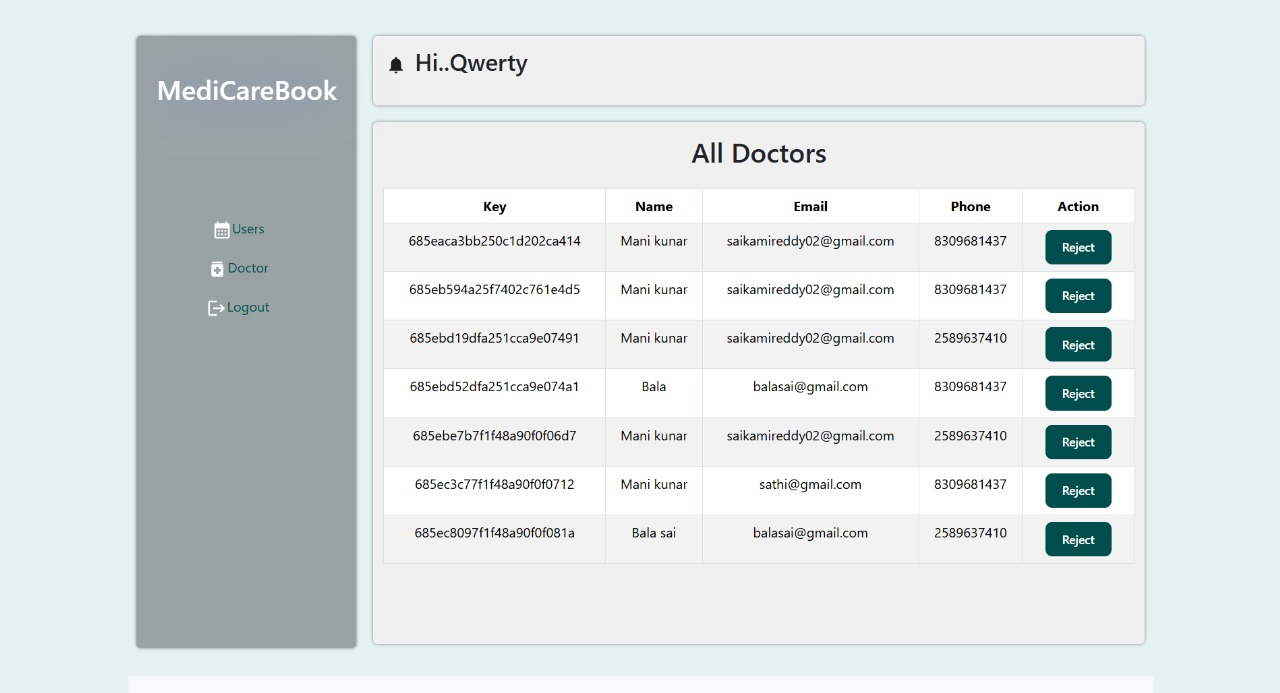
**Home Page**



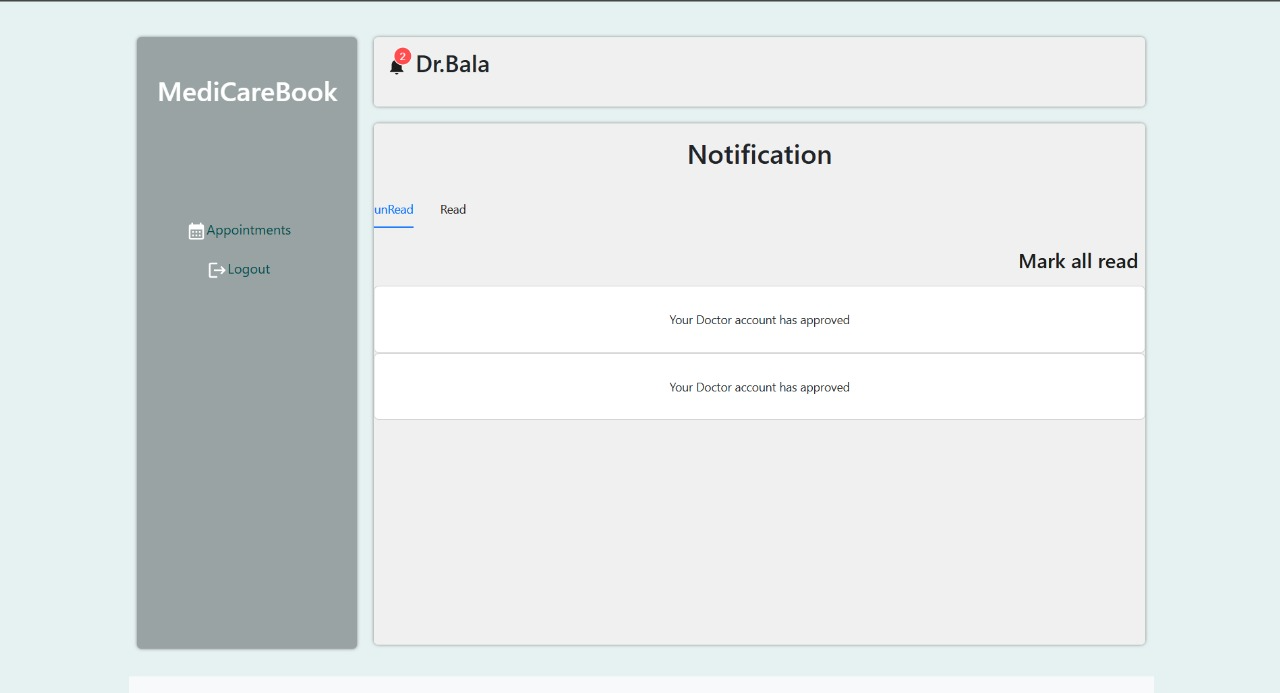
* **Appointment Page**



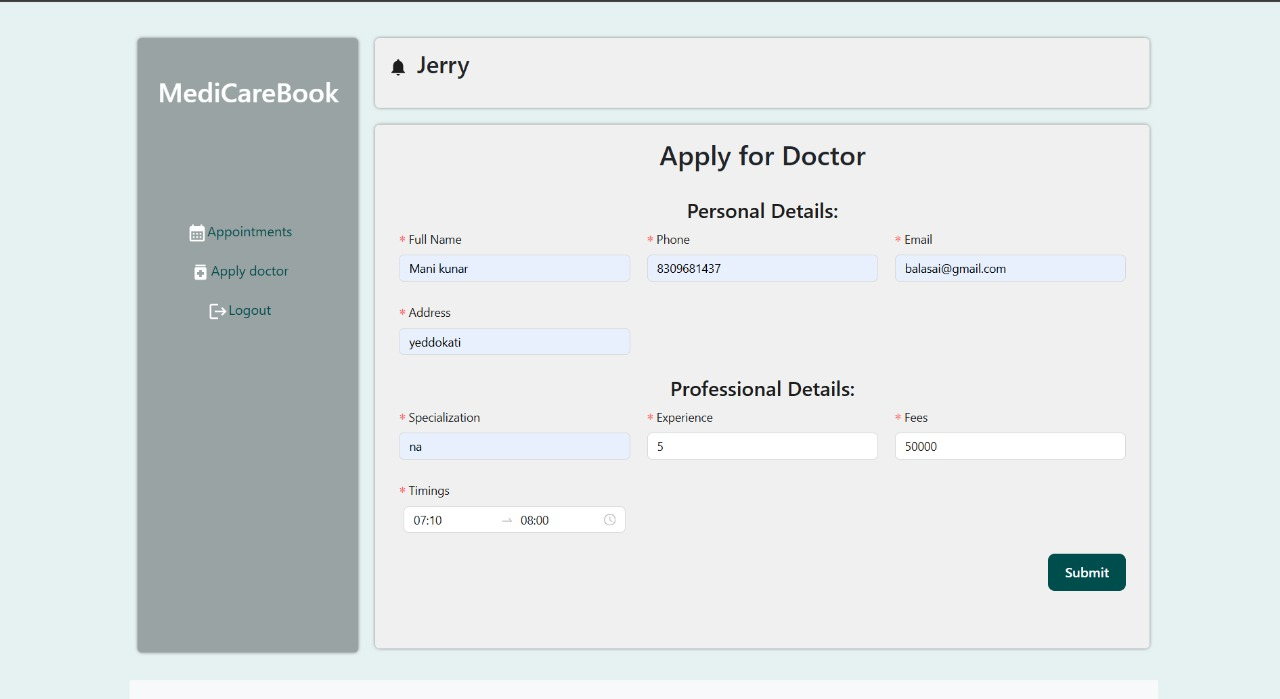
**Doctors Page**



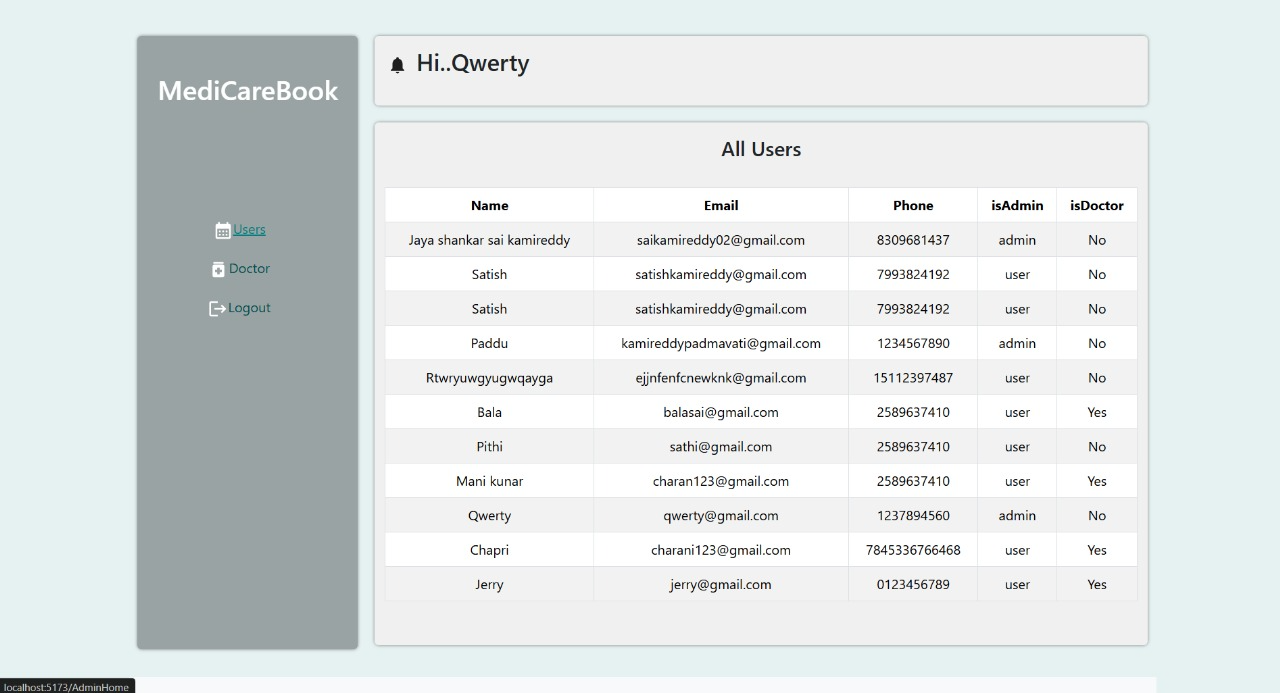
* **Notification Page**

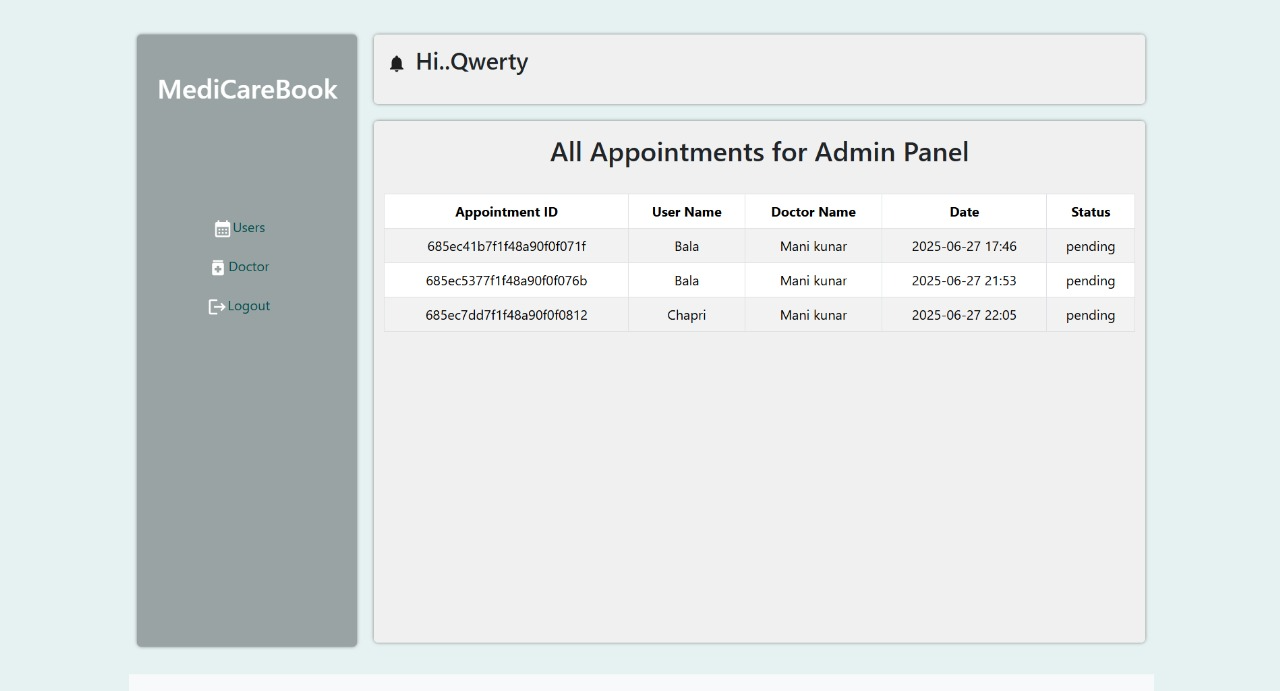


**Apply Page**



* **User Page**



Known Issues

* Lack of automated testing (unit/integration).
* No continuous integration or deployment pipeline in place.
* Error handling and form validation can be improved further.
* Provider and appointment data are not paginated.

## Future Enhancements

\* Integrate **payment gateway** (e.g., Stripe, Razor pays) for secure transactions.

\* Implement real-time appointment status updates using WebSockets (e.g., Socket.IO).

\* Add push notifications for users on appointment activity.

\* Set up CI/CD pipeline for automated deployment and testing.

\* Improve admin panel with analytics and provider insights.

\* Add user profile management (update info, view past appointments).