Full Stack Development with MERN

Docspot

## Introduction

• **Project Title:** **[Docspot: Seamless Appointment Booking For Health]**

**Team ID:** LTVIP2025TMID53081

• **Team Members:** **Team Size:** 4

**Team Leader:** Addala Ayyappa - Responsible for project planning,

coordination and final integration.

**Team member:** A Kartheek - Ensured the app works well on all devices

**Team member:** A Janakiram - Worked on the logic behind the app

and bug fixing.

**Team member:** A Manaswini - Helped with documentation and

deployment and system testing.

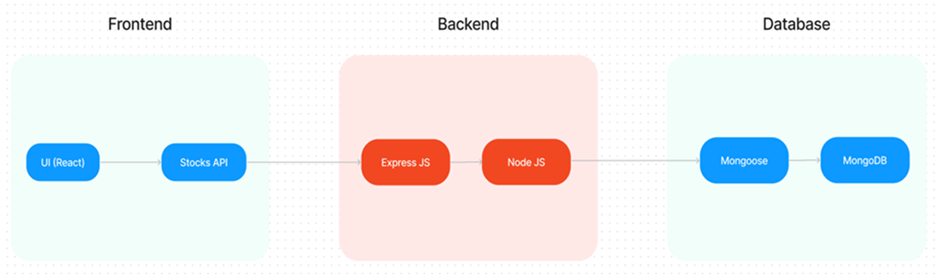
## 2. Project Overview

**Purpose**:  
The DocSpot project aims to create an intuitive and efficient online platform that seamlessly connects patients with healthcare providers. The system simplifies the appointment booking process by allowing users to search for doctors by specialty, location, and availability, book appointments, manage schedules, and receive timely notifications. It addresses the challenges of traditional appointment systems by providing a user-friendly, secure, and accessible healthcare booking solution for patients, doctors, and administrators.

**Features**:

* **Patient Registration & Profile Management**: Secure sign-up and profile creation with personal and medical details.
* **Doctor Search & Filtering**: Browse and filter doctors by specialty, location, and real-time availability.
* **Appointment Booking & Management**: Easy scheduling of appointments with document uploads and automated notifications.
* **Doctor Dashboard**: Doctors can manage their availability, appointments, and patient records securely.
* **Admin Controls**: Admin panel for approving doctor registrations, managing users, and platform governance.
* **Notifications**: Automated email and SMS reminders for appointments to reduce no-shows.
* **Secure Authentication & Data Protection**: Utilizes JWT and bcrypt for secure login and password management, with encrypted data handling.

## 3. Architecture

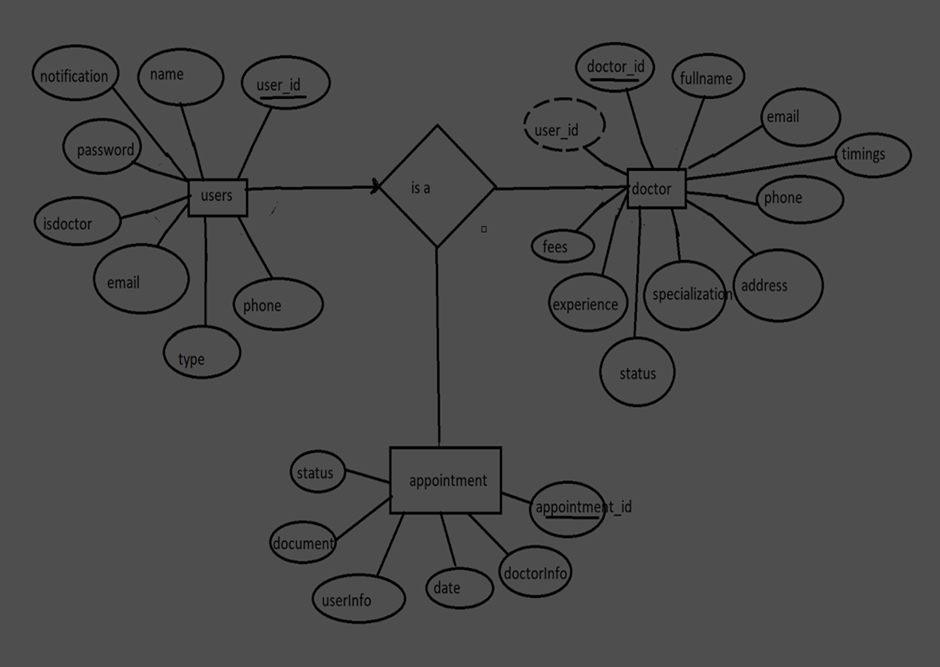


**Frontend**:  
The frontend is built using **React.js**, a component-based JavaScript library that facilitates dynamic and responsive user interfaces. The architecture follows a modular design with reusable components for functionalities like doctor browsing, appointment booking, and user profile management. **React Router** manages seamless navigation between pages. Styling is achieved using **Bootstrap**, **Material UI**, and **Ant Design** for a consistent, modern, and responsive look across devices. **Axios** handles API requests for smooth communication with the backend.

**Backend**:  
The backend is developed using **Node.js** with the **Express.js** framework to create a RESTful API. Express manages HTTP request routing, middleware processing, and business logic execution. The server handles user authentication, appointment scheduling, doctor and patient data management, and notification services. Security is enforced using **JWT** for token-based authentication and **bcrypt** for password hashing. The backend exposes endpoints to support frontend operations like user login, appointment creation, and admin controls.

**Database**:  
The database is implemented with **MongoDB**, a NoSQL document-oriented database storing data in flexible JSON-like documents. The schema includes three main collections:

* **Users**: Stores user details including login credentials, role flags (doctor/patient/admin), and contact information.
* **Doctors**: Contains doctor-specific information linked to user accounts via userID, including specialization, availability, experience, and fees.
* **Appointments**: Records appointment details linking users and doctors, including appointment date, status, and uploaded documents. Relationships are managed through references (foreign keys) for efficient queries and data integrity.

**ER Diagram**:  


## 4. Setup Instructions

### Prerequisites

* **Node.js & npm**: JavaScript runtime and package manager for running the backend and frontend.  
  Download: <https://nodejs.org/en/download/>
* **MongoDB**: NoSQL database for storing application data (local or MongoDB Atlas).  
  Download: <https://www.mongodb.com/try/download/community>
* **Git**: Version control tool to clone the project repository.  
  Download: <https://git-scm.com/downloads>

### Installation

1. **Clone the Repository**:

* git clone https://github.com/docspot/docspot.git  
  cd docspot

1. **Setup Backend**:  
   Navigate to the backend directory and install dependencies:

* cd backend  
  npm install
* Create a .env file in the backend folder with:
* PORT=5000  
  MONGO\_URI=<your\_mongodb\_connection\_string>  
  JWT\_SECRET=<your\_jwt\_secret\_key>
* Start the backend server:
* npm start
* The backend API will be available at http://localhost:5000.

1. **Setup Frontend**:  
   Open a new terminal, navigate to the frontend directory, and install dependencies:

* cd frontend  
  npm install
* Start the frontend server:
* npm start
* The frontend will be accessible at http://localhost:3000.

## 5. Folder Structure

### Client (Frontend)

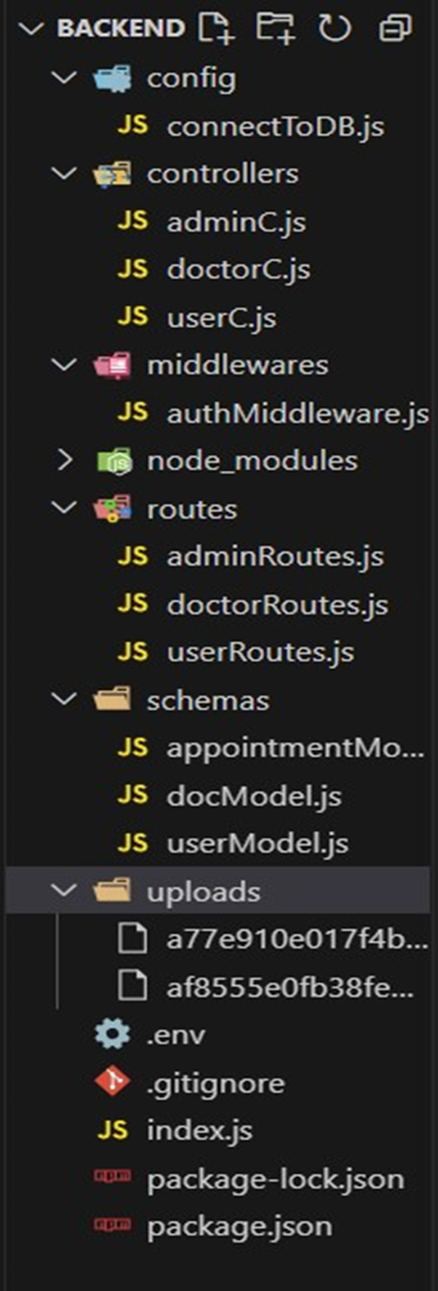
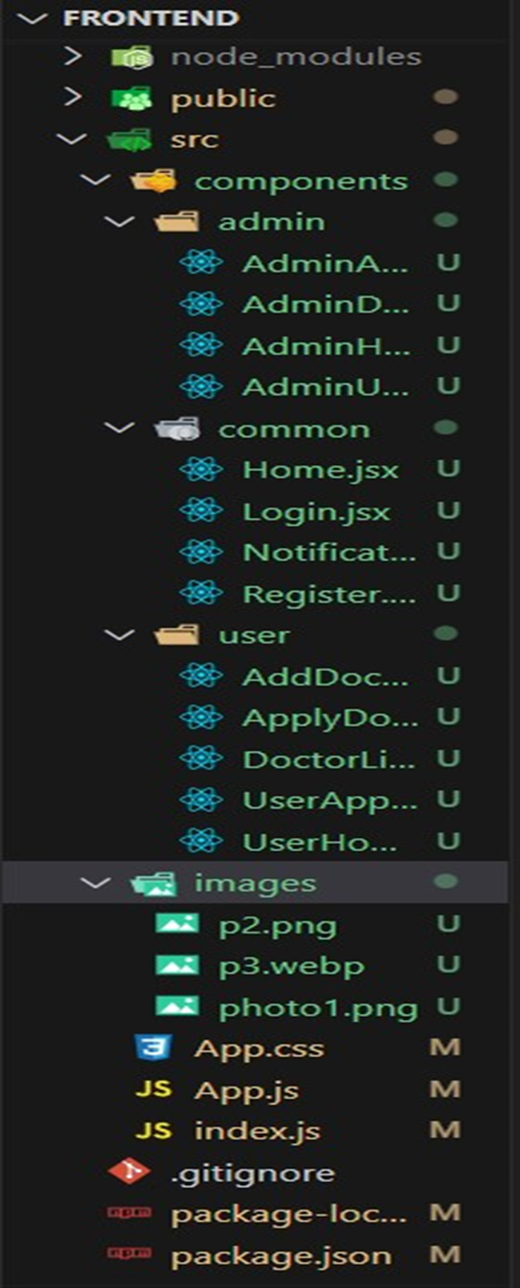
The React frontend is organized for modular development and maintainability:

* public/: Static assets like HTML templates and images.
* src/: Main source folder containing React components.
  + components/: Reusable UI components (e.g., doctor listings, appointment forms).
  + pages/: Full-page components (e.g., Dashboard, Login, Booking History).
  + services/: API service functions for backend interaction using Axios.
* App.js: Root component handling routing and global state.
* index.js: Entry point for rendering the React application.
* .env: Frontend environment variables.
* package.json: Frontend dependencies and scripts.  
  Frontend Folder Structure

### Server (Backend)

The Node.js backend is structured for scalability and clear API flow:

* config/: Configuration files (e.g., database connection, environment variables).
* controllers/: Business logic for API endpoints (users, doctors, appointments).
* models/: Mongoose schemas for database collections (Users, Doctors, Appointments).
* routes/: Express route handlers mapping endpoints to controllers.
* middleware/: Custom middleware for authentication, error handling, and logging.
* uploads/: Folder for securely storing uploaded medical documents.
* server.js: Entry point initializing the Express server and MongoDB connection.
* .env: Backend environment variables (e.g., database URI, JWT secrets).
* package.json: Backend dependencies and scripts.  
  Backend Folder Structure



## 6. Running the Application

1. **Frontend**:  
   Navigate to the frontend directory and run:

* npm start
* Accessible at http://localhost:3000.

1. **Backend**:  
   Navigate to the backend directory and run:

* npm start
* Accessible at http://localhost:5000.

Ensure both servers run simultaneously for full functionality.

## 7. API Documentation

The backend exposes RESTful API endpoints to support frontend operations. Key endpoints include:

### User Registration

* **POST** /api/users/register
* **Request Body**:
* {  
   "name": "John Doe",  
   "email": "john@example.com",  
   "password": "password123",  
   "type": "patient"  
  }
* **Response**:
* {  
   "success": **true**,  
   "message": "User registered successfully."  
  }

### User Login

* **POST** /api/users/login
* **Request Body**:
* {  
   "email": "john@example.com",  
   "password": "password123"  
  }
* **Response**:
* {  
   "token": "<jwt\_token>",  
   "user": {  
   "\_id": "user\_id",  
   "name": "John Doe",  
   "email": "john@example.com",  
   "type": "patient"  
   }  
  }

### Fetch Doctors List

* **GET** /api/doctors
* **Query Parameters**: Optional filters (e.g., specialization, location)
* **Response**:
* [  
   {  
   "\_id": "doctor\_id",  
   "fullname": "Dr. Smith",  
   "specialisation": "Cardiology",  
   "experience": "10 years",  
   "fees": 500  
   },  
   **...**  
  ]

### Book Appointment

* **POST** /api/appointments
* **Request Body**:
* {  
   "doctorId": "doctor\_id",  
   "userId": "user\_id",  
   "date": "2025-07-01T10:00:00Z",  
   "documents": ["document\_url\_1", "document\_url\_2"],  
   "status": "pending"  
  }
* **Response**:
* {  
   "success": **true**,  
   "message": "Appointment booked successfully."  
  }

**Other Endpoints**: Include routes for appointment status updates, doctor registration approval, and user profile management.

## 8. Authentication

DocSpot uses **JWT (JSON Web Tokens)** for secure authentication and authorization:

* **User Login**: The backend generates a JWT upon successful login, containing user identification and role information.
* **Token Storage**: The frontend stores the token securely (in memory or local storage) and includes it in the Authorization header for API requests.
* **Authorization**: Backend middleware verifies the JWT on protected routes to ensure user authentication and authorization.
* **Password Security**: Passwords are hashed using **bcrypt** before storage, preventing plaintext leaks.
* **Session Management**: Stateless token authentication eliminates server-side sessions, improving scalability.

## 9. User Interface

The DocSpot app features a clean, responsive, and intuitive interface:

* **Patient Dashboard**: Searchable doctor list with filters for specialty and location.  
  User Dashboard
* **Appointment Booking Form**: Select available dates, upload documents, and confirm bookings.  
  Book Doctor
* **Doctor Dashboard**: View upcoming appointments, update statuses, and manage patient records.  
  Doctor Dashboard
* **Admin Panel**: Verify doctor registrations, manage users, and oversee platform operations.  
  Admin Dashboard  
  Admin Approve Doctor
* **Notifications**: Alerts for appointment confirmations, cancellations, and reminders via email and SMS.
* **Landing Page**: Welcoming interface for users.  
  Landing Page
* **Register Page**: User-friendly registration form.  
  Register Page
* **Login Page**: Secure login interface.  
  Login Page
* **Booking History**: View and manage past and upcoming appointments.  
  All History

## 10. Scenario-based Case Study

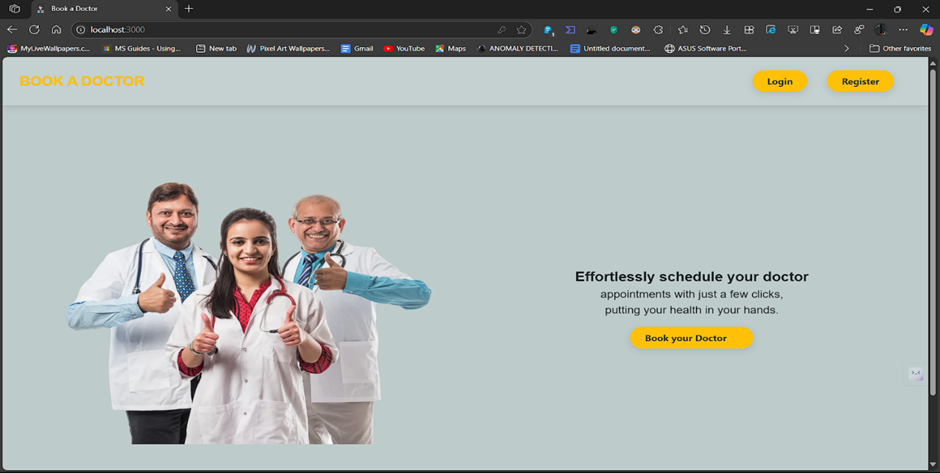
**Scenario: Booking an Appointment with a Doctor**

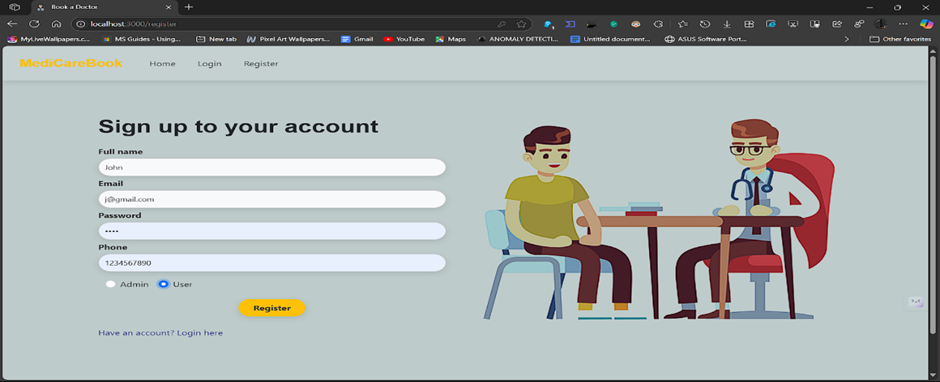
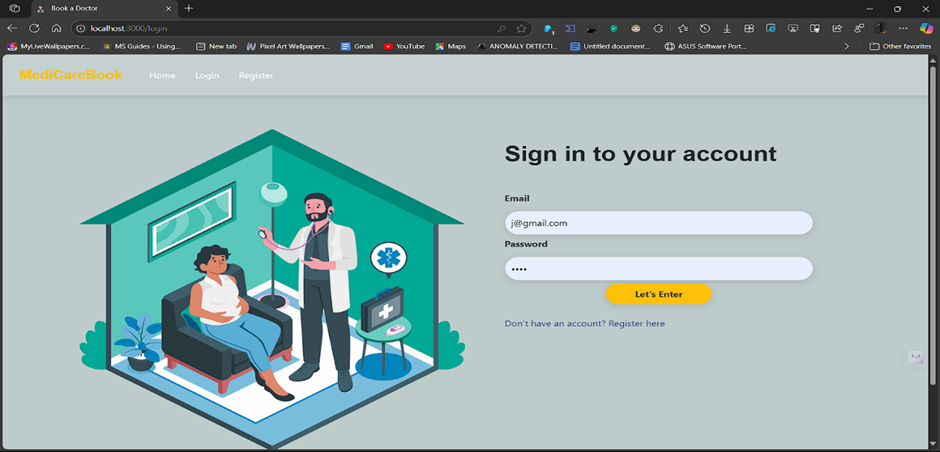
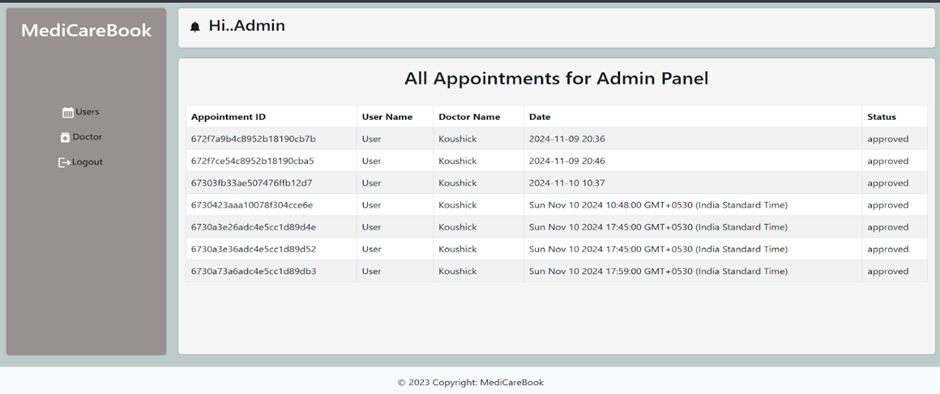
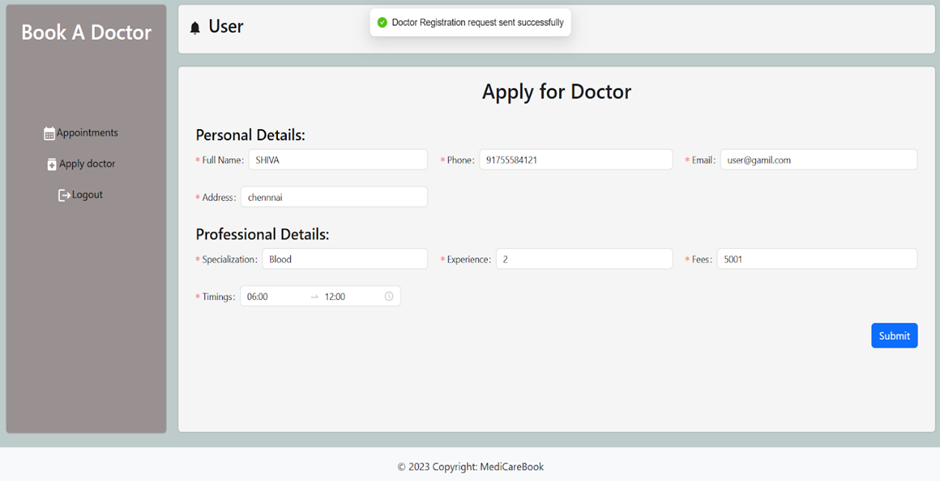
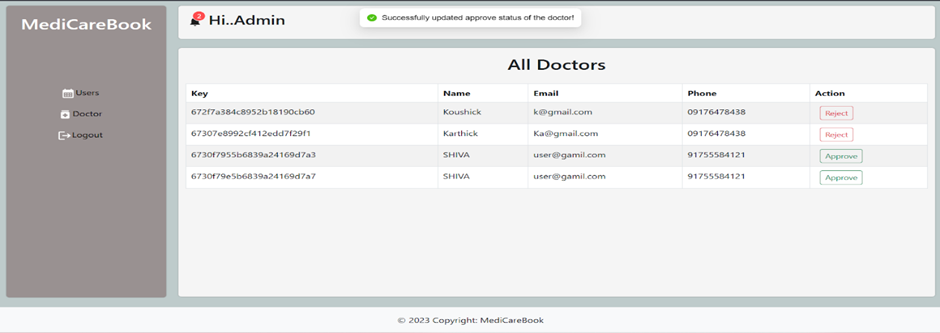
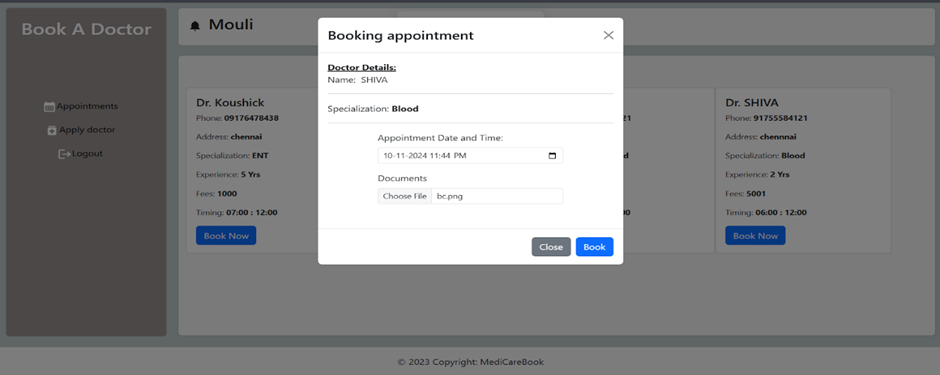
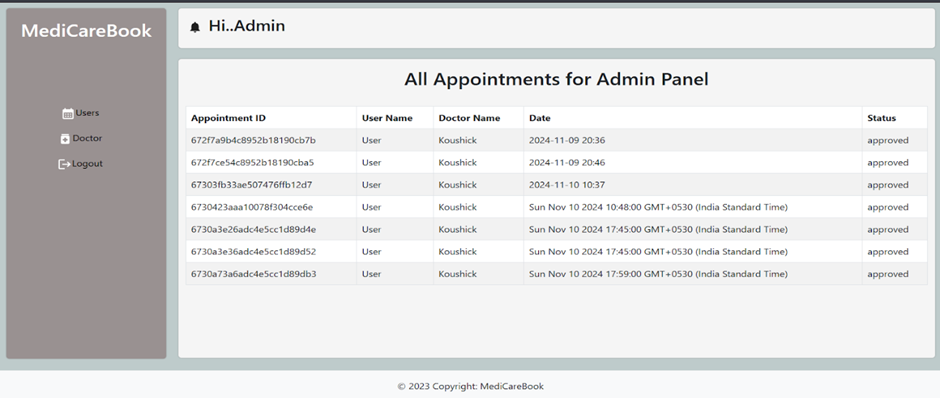
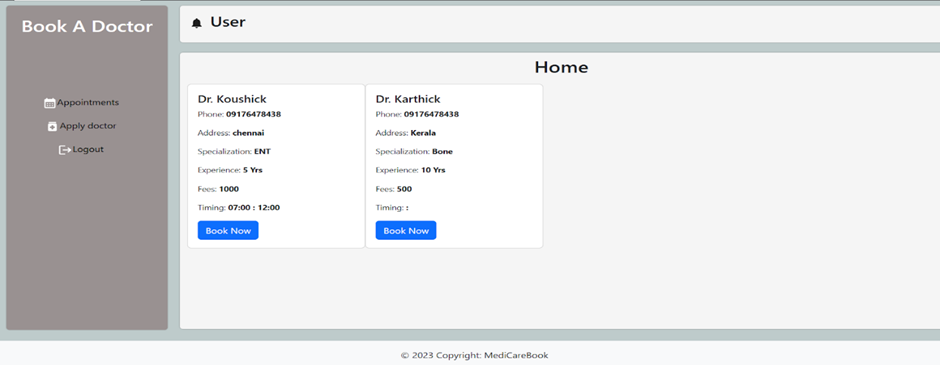
1. **User Registration**: John, needing a routine check-up, visits DocSpot and signs up as a patient with his email and password.
2. **Browsing Doctors**: Upon logging in, John sees a dashboard with available doctors, filtering by specialty or location.
3. **Booking an Appointment**: John selects a doctor, clicks “Book Now,” chooses a date, uploads documents (e.g., medical records), and submits the request.
4. **Appointment Confirmation**: The doctor reviews and confirms the appointment, updating the status to “scheduled.” John receives a confirmation with details.
5. **Appointment Management**: John views or cancels appointments in the booking history section.
6. **Admin Approval (Background)**: Admins review and approve new doctor registrations, ensuring only legitimate doctors join the platform.
7. **Doctor’s Appointment Management**: Dr. Smith manages appointments, confirms or reschedules them, and updates statuses.
8. **Appointment Consultation**: John visits the doctor’s office for the check-up, where Dr. Smith provides care.
9. **Post-Appointment Follow-up**: Dr. Smith updates John’s records and sends a visit summary with follow-up instructions via the app.

## 11. Screenshots or Demo

Screenshots showcasing key features:

* **Landing Page**: Welcoming interface for users.



* **Register Page**: User-friendly registration form.  
  
* **Login Page**: Secure login interface.  
  
* **Admin Dashboard**: Overview for platform management.  
  
* **Doctor Dashboard**: Appointment and patient management for doctors.  
  
* **Admin Approve Doctor**: Interface for approving doctor registrations.  
  
* **Book Doctor**: Appointment booking form.  
  
* **All History**: Booking history for users.  
  
* **User Dashboard**: Patient interface for browsing and booking.  
  

**Demo Video**: <https://drive.google.com/file/d/1_9C5GZQvkcefAiArfnPdQFmcIsvIqkpQ/view?usp=sharing>

## 12. Known Issues

* **Appointment Rescheduling**: No direct option for patients to reschedule; they must cancel and rebook.
* **Notification Delays**: Occasional delays in email/SMS notifications under high server load.
* **File Upload Limits**: Size restrictions on medical document uploads may lack clear warnings.
* **Mobile Responsiveness**: Minor UI elements need optimization for smaller screens.
* **Role Permissions**: Edge cases in role-based access control may show restricted features (though not accessible).

## 13. Future Enhancements

* **Appointment Rescheduling Feature**: Enable direct rescheduling through the interface.
* **Real-Time Chat**: Add secure messaging for patient-doctor communication.
* **Multi-Language Support**: Include localization for multiple languages.
* **Advanced Analytics**: Provide doctors and admins with appointment and patient trend insights.
* **Payment Integration**: Enable online payments for consultation fees.
* **Mobile App**: Develop native Android and iOS apps for broader accessibility.
* **AI-based Doctor Recommendations**: Suggest doctors based on patient history and preferences.

## 14. Testing

A comprehensive testing strategy ensures reliability and performance:

* **Unit Testing**: Tested React components and API controllers using **Jest**, **React Testing Library**, **Mocha**, and **Chai**.
* **Integration Testing**: Verified API endpoints and database interactions using **Postman** and automated scripts.
* **User Acceptance Testing (UAT)**: Validated user experience and functional requirements with stakeholders.
* **Performance Testing**: Conducted load testing to ensure responsiveness under concurrent users.
* **Security Testing**: Audited password encryption, authentication flows, and data transmission for vulnerabilities.

## 15. Project Flow and References

**Application Flow**:

* **Customer**: Registers, logs in, browses doctors, books appointments, uploads documents, and manages bookings in the history section.
* **Doctor**: Gets admin approval, manages appointments, and updates patient records.
* **Admin**: Approves doctor registrations, monitors platform operations, and ensures compliance with policies.

**Sample Credentials**:

* **Admin**: [admin@docspot.com](mailto:admin@docspot.com) / admin123
* **Sample Patient**: [john.doe@example.com](mailto:john.doe@example.com) / patient123
* **Sample Doctor**: [sarah.johnson@docspot.com](mailto:sarah.johnson@docspot.com) / doctor123

**Credentials used for Doctor logins:**

<https://docs.google.com/spreadsheets/d/1QiHjEuzk07JBgVMDPorK6Z81q6SNZ6SDmUZxhNjQ-mA/edit?usp=sharing>