ONLINE DOCTOR BOOKING USING MERN STACK

**ABSTRACT:**

The online doctor booking system provides a convenient, efficient platform for patients to book medical consultations with healthcare professionals. Leveraging the MERN stack— MongoDB, Express.js, React, and Node.js—this system facilitates seamless interactions between patients and doctors, optimizing both accessibility and convenience in healthcare services.

The system enables users to register as either patients or doctors, with tailored features for each role. Patients can browse doctor profiles, filter by specialization or location, and view available time slots for appointments. With an intuitive booking interface, patients can select their preferred slot, confirm appointments, and make payments if required, all through a secure, user-friendly process. Doctors, in turn, can create profiles, manage appointment schedules, and receive real-time notifications, improving their ability to connect with patients effectively.

This system ensures data security and privacy by implementing user authentication and authorization protocols, including JWT (JSON Web Token) for session management.

MongoDB is used for managing and storing user data, while Express.js and Node.js support robust backend operations, and React provides a responsive and dynamic user interface.

The online doctor booking platform ultimately addresses the growing demand for accessible healthcare by digitizing the process of booking medical appointments, making it easy for patients to access quality healthcare while helping doctors reach a broader audience.

1. **INTRODUCTION:**

In the digital age, healthcare accessibility and convenience are paramount. Traditional methods of booking doctor appointments, such as in-person visits or phone calls, often lead to inefficiencies and frustration for both patients and healthcare providers. To address these challenges, this online doctor booking system provides a modern solution that connects patients with doctors through a streamlined digital platform.

This system aims to enhance patient access to healthcare while allowing doctors to manage their appointments and schedules more efficiently. Built using the MERN stack—MongoDB,

Express.js, React, and Node.js—the platform is designed to deliver a high-performance and user-friendly experience for patients and healthcare professionals alike.

* 1. **PURPOSE:**

The primary purpose of this project is to develop a convenient, secure, and efficient platform for patients to book appointments with doctors. The system is intended to simplify the appointment process by eliminating the need for phone calls, reducing wait times, and allowing users to book, reschedule, or cancel appointments at their convenience. For healthcare providers, this system also helps in managing schedules, reducing no-shows, and expanding their reach to more patients**.**

* 1. **SCOPE:**

The online doctor booking system targets two main user types: patients and doctors. Patients will have access to features such as doctor search, profile viewing, real-time slot availability, and appointment booking. Doctors can manage their profiles, availability, and appointments through a dedicated interface. Additionally, the system will support secure authentication, real-time notifications, and feedback mechanisms to ensure a reliable and interactive experience.

The project scope also covers essential aspects of data privacy and security, ensuring that users’ personal and medical information is protected. The use of the MERN stack allows for flexibility, scalability, and responsiveness, creating a robust application that can cater to a growing number of users and features.

* 1. **OBJECTIVE:**

The online doctor booking system is developed with the following key objectives:

* + 1. **Enhance Patient Access to Healthcare Services**: Make it easier for patients to find and book appointments with healthcare providers quickly and conveniently.
    2. **Streamline Appointment Scheduling**: Allow patients to view available slots in real time and book or manage appointments without the need for direct contact with the clinic, thus reducing scheduling conflicts and wait times.
    3. **Support Doctor-Patient Communication**: Enable a seamless flow of information between patients and doctors, including appointment reminders and feedback options.
    4. **Ensure Security and Privacy**: Protect users' sensitive data by implementing secure authentication and data handling processes.
    5. **Create a Scalable and User-Friendly Platform**: Utilize the MERN stack to build an intuitive, responsive, and efficient application capable of handling a large user base.

## SYSTEM REQUIREMENT:

* 1. **HARDWARE:**
* Processor: Minimum Intel i3 or equivalent
* RAM: 8 GB (recommended for smooth development experience)
* Storage: 256 GB SSD (recommended), or minimum 500 GB HDD
* Graphics: Standard integrated graphics for local development

## SOFTWARE:

* Operating System: Windows 10, MacOS, or Linux
* Development Environment: Visual Studio Code, WebStorm, or similar IDEs
* Front-End: React.js
* Back-End: Node.js with Express.js framework
* Database: MongoDB (NoSQL Database)
* Version Control: Git (e.g., GitHub, GitLab)
* Other Tools: Postman (for API testing), Docker (optional for containerization)

## NETWORK:

* Internet Connectivity: Required for initial setup, database access (if using a cloud-based MongoDB), and package installations
* Server: Deployment on cloud servers (AWS, Heroku, or similar), or local server for testing.

## PRE-REQUISITES:

Here are the key prerequisites for developing a full-stack application usingNode.js, Express.js, MongoDB, React.js:

Vite: Vite is a new frontend build tool that aims to improve the developer experience for development with the local machine, and for the build of optimized assets for production(go live). Vite (or ViteJS) includes: a development server with ES \_native\_ support and Hot Module Replacement; a build command based on rollup.

Installation : npm create vite@latest  Node.js and npm: Node.js is a powerful JavaScript runtime environment that allows you to run JavaScriptcode on the server-side. It provides a scalable and efficient platform for building network applications. Install Node.js and npm on your development machine, as they are required to run JavaScript on the server-side.

Download: https://nodejs.org/en/download/ Installation instructions: https://nodejs.org/en/download/package-manager/ Run “npm init” to get default dependencies

Express.js: Express.js is a fast and minimalist web application framework for Node.js. It simplifies the process of creating robust APIs and web applications, offering features like routing, middleware support, and modular architecture. Install Express.js, a web application framework for Node.js, which handles server-side routing, middleware, and API development. Installation: Open your command prompt or terminal and run the following command: npm install express

MongoDB: MongoDB is a flexible and scalable NoSQL database that stores data in a JSON- like format. It provides high performance, horizontal scalability, and seamless integration withNode.js, making it ideal for handling large amounts of structured and unstructured data. Set up a MongoDB database to store your application's data. Download: https[://w](http://www.mongodb.com/try/download/community)ww[.m](http://www.mongodb.com/try/download/community)o[ngodb.com/try/download/community](http://www.mongodb.com/try/download/community) Installation instructions: https://docs.mongodb.com/manual/installation/

React.js: React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications. Install React.js, a JavaScript library for building user interfaces. Follow the installation guide: https://reactjs.org/docs/create-a-new-react-app.html

 HTML, CSS, and JavaScript: Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.

Database Connectivity: Use a MongoDB driver or an Object-Document Mapping (ODM) library like Mongo set connect your Node.js server with the MongoDB database and perform CRUD(Create, Read, Update, Delete) operations.  Front-end Framework: Utilize React.js to build the user-facing part of the application, including entering booking room, status of the booking, and user interfaces for the admin dashboard. For making better UI we have also used some libraries like material UI and bootstrap. Install the required dependencies by running the following commands:

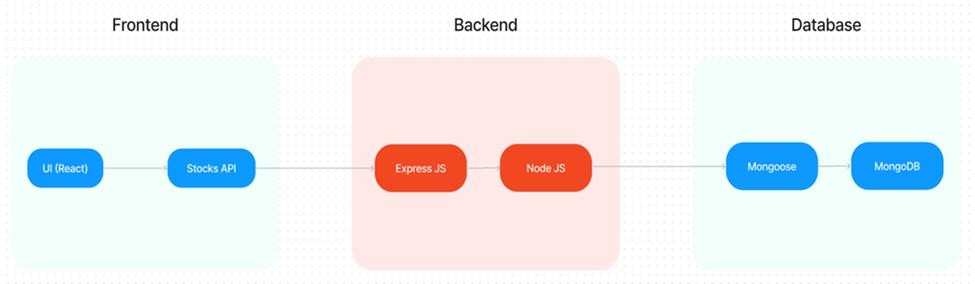
cd frontend || npm install cd ../backend || npm install

Start the Development Server: • To start the development server, execute the following command: npm start • The OLP app will be accessible at http://localhost:5172 The installation is successfully installed and set up the Online learning app on your local machine. You can now proceed with further customization, development, and testing as needed.

## ARCHITECTURE:

* Front-End: The React.js application is responsible for rendering the UI and interacting with the backend via API calls.
* Back-End: The server, built with Node.js and Express.js, manages client requests and performs business logic.
* Database: MongoDB stores user data, doctor profiles, appointment information, and feedback data.
* Deployment: The entire stack is hosted on a cloud platform, with options like AWS, Heroku, or DigitalOcean for scalability**.**

## TECHNICAL ARCHITECTURE:



Creating a technical architecture for a doctor booking system using the MERN stack involves setting up components for each part of the stack (MongoDB, Express, React, Node.js) to handle various parts of the system, including data management, API endpoints, and frontend UI. Here’s an overview of a typical architecture:

### Frontend: React

* + **Description:** The user-facing part of the system where patients can view doctor profiles, search for specialists, book appointments, and leave feedback.

### Components:

* + - **Patient Portal:** Contains pages for searching and booking doctors, viewing bookings, and providing feedback.
    - **Doctor Portal:** Allows doctors to view their bookings, manage availability, and respond to feedback.
    - **Admin Dashboard (optional):** For managing doctors, appointments, and reviews.
  + **State Management:** Use a state management library like Redux for managing user session data, booking states, and feedback.
  + **Authentication:** JWT tokens stored in cookies or local storage to manage user sessions.

### Backend: Node.js + Express

* + **Description:** Manages the application’s server-side logic, APIs, and communication between the frontend and the database.

### API Endpoints:

* + - **User Authentication:** Sign-up, login, and JWT-based authentication for security.
    - **Booking Management**: Endpoints for creating, viewing, updating, and deleting bookings.
    - **Doctor Management:** Allows creation and management of doctor profiles, including availability and specialization.
    - **Feedback Management:** Endpoints for patients to submit feedback and for viewing reviews.
  + **Middleware:** JWT authentication middleware to protect routes that require user login (e.g., booking or viewing appointments).
  + **Error Handling & Logging**: Implement robust error handling and logging mechanisms using libraries like Winston**.**

### Database: MongoDB

* + **Description:** Stores application data, such as patient and doctor information, appointment records, and feedback.

### Collections:

* + - **Users:** Stores patient and doctor information (e.g., name, contact details, password hash).
    - **Doctors:** Contains doctor details like specialization, experience, and availability.
    - **Appointments:** Stores appointment details, including patient ID, doctor ID, date, and time.
    - **Feedback:** Contains patient reviews and ratings for doctors.
  + **Indexes:** Add indexes on fields that will be frequently queried, such as doctor specialty or patient ID in the appointments collection.

### Authentication: JWT

* + **User Authentication**: JSON Web Tokens (JWT) for securely verifying users on each request.
  + **Access Control**: Different roles (patient, doctor, admin) to control access to specific routes and functionalities.

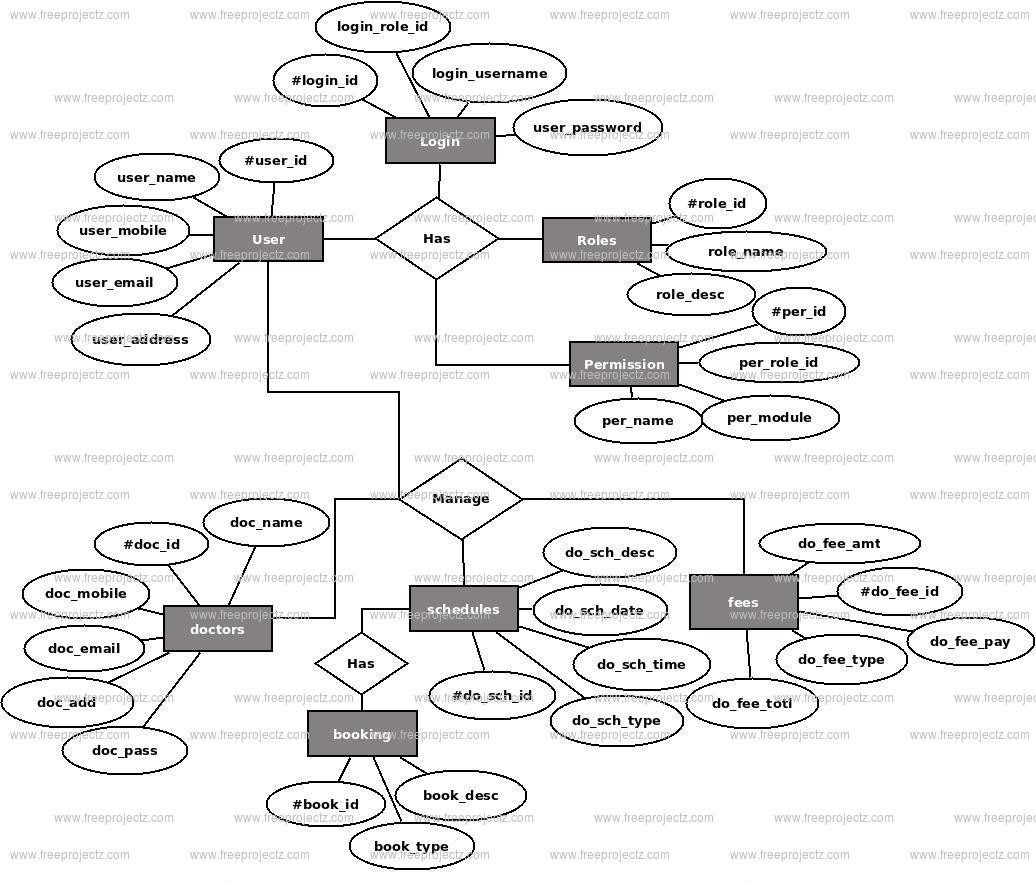
### Deployment & Hosting

* + **Frontend (React):** Deploy on platforms like Vercel or Netlify for scalability.
  + **Backend (Node.js + Express):** Deploy on Heroku or DigitalOcean for easy scaling.
  + **Database (MongoDB):** Use MongoDB Atlas for cloud-hosted, scalable MongoDB databases.
  + **Continuous Integration (CI):** Use GitHub Actions for automatic testing and deployment of updates.

### Additional Features:

* + **Real-Time Notifications:** Integrate WebSocket or Firebase for real-time updates, like new bookings or changes in appointment status.
  + **Email or SMS Notifications**: Trigger notifications for appointment confirmations, reminders, and updates.

## ER DIAGRAM:



An **ER (Entity-Relationship) Diagram** visually represents the relationships and structure of a database for a specific system—in this case, an **online doctor booking system**. Here’s a breakdown of the components and their roles within this system.

### Key Elements of the ER Diagram

1. **Entities**: Represented as rectangles, entities are the main objects within the system. Each entity holds attributes and has a unique identifier (Primary Key).
   * **Patient**: Represents users (patients) who can book appointments. Each patient has attributes like PatientID, Name, Contact Information, Gender, Age, and Address.
   * **Doctor**: Represents doctors who are available for booking. Each doctor has a DoctorID, Name, Contact Information, Specialization, and Availability.
   * **Specialization**: Represents the various areas of expertise a doctor may have, like "Cardiologist" or "Dermatologist". This allows patients to filter doctors based on the required expertise.
   * **Appointment**: Represents each booking or meeting between a patient and a doctor. Each appointment stores the PatientID, DoctorID, AppointmentDate, AppointmentTime, and Status.
   * **Feedback**: Represents feedback given by patients after their appointments. Feedback includes the Rating, Comments, and a timestamp (FeedbackDate).
2. **Relationships**: Shown as diamonds, relationships connect entities and describe how they interact. Each relationship has cardinality (1:1, 1, M), specifying how many entities participate.

### Patient - Appointment: A 1

relationship, where a single patient can book multiple appointments, but each appointment is associated with only one patient.

### Doctor - Appointment: A 1

relationship, where a single doctor can have multiple appointments (appointments with different patients), but each appointment is with only one doctor.

### Doctor - Specialization: A 1

relationship where each doctor has one specialization, but each specialization (e.g., Cardiologist) can have multiple doctors.

### Patient - Feedback: A 1

relationship, where each patient can leave multiple feedback entries for different appointments, but each feedback is tied to one patient.

### Doctor - Feedback: A 1

relationship, where each doctor can receive multiple feedback entries, each associated with a different patient.

1. **Attributes**: These are shown within each entity, often inside ovals, and they represent details that describe each entity. Key attributes include:
   * For **Patient**: PatientID (Primary Key), Name, Contact, Gender, etc.
   * For **Doctor**: DoctorID (Primary Key), Name, Contact, SpecializationID (Foreign Key), Availability, etc.
   * For **Appointment**: AppointmentID (Primary Key), PatientID, DoctorID, Date, Time, etc.
2. **Primary and Foreign Keys**: Primary keys (like PatientID) uniquely identify each instance of an entity. Foreign keys (like PatientID in the Appointment entity) connect one entity to another, indicating relationships.

### Purpose of the Diagram

This ER diagram helps to:

* **Design the Database Structure**: Ensures that all key components of the system are accounted for and linked properly.
* **Clarify Relationships**: Shows how entities (e.g., Patient and Doctor) interact, helping avoid redundant or conflicting relationships.
* **Optimize Database Queries**: Helps in deciding which fields to index and how to manage queries efficiently (e.g., searching appointments by DoctorID).

In summary, this ER diagram serves as a blueprint for structuring the doctor booking system’s database, ensuring that data is organized, easily accessible, and correctly interlinked. This foundation is essential for efficient data storage and retrieval, especially in a system involving multiple interactions between patients, doctors, and appointments.

## PROJECT STRUCTURE:

A typical project structure for a doctor booking system using the MERN (MongoDB, Express, React, Node.js) stack would look something like this. This structure organizes code by responsibility, ensuring scalability, easy maintenance, and separation of concerns.

|  |
| --- |
| doctor-booking-system/ |
| ├── client/ # React Frontend |
| │ ├── public/ # Static files (HTML, images, etc.) |
| │ │ ├── index.html # Main HTML file |
| │ │ └── favicon.ico |
| │ ├── src/ |
| │ │ ├── assets/ # Images, icons, fonts, etc. |
| │ │ ├── components/ # Reusable UI components (e.g., buttons, form inputs) |

|  |
| --- |
| │ │ ├── pages/ # Page components (e.g., Home, DoctorProfile, Booking) |
| │ │ ├── services/ # Functions to make API requests |
| │ │ ├── App.js # Main app component |
| │ │ ├── index.js # React entry point |
| │ │ ├── routes.js # Route definitions (React Router) |
| │ │ ├── store/ # State management files (e.g., Redux) |
| │ │ └── styles/ # CSS or Sass files |
| │ └── package.json # Client dependencies and scripts |
| ├── server/ # Node.js Backend |
| │ ├── config/ # Configuration files (e.g., database config) |
| │ │ └── db.js # MongoDB connection setup |
| │ ├── controllers/ # Request handlers for routes |
| │ │ ├── patientController.js |
| │ │ ├── doctorController.js |
| │ │ └── appointmentController.js |
| │ ├── models/ # Mongoose schemas for MongoDB collections |
| │ │ ├── Patient.js |
| │ │ ├── Doctor.js |
| │ │ ├── Appointment.js |
| │ │ └── Feedback.js |
| │ ├── routes/ # Route definitions |
| │ │ ├── patientRoutes.js |
| │ │ ├── doctorRoutes.js |
| │ │ └── appointmentRoutes.js |
| │ ├── middleware/ # Middleware (e.g., authentication, error handling) |
| │ │ ├── authMiddleware.js |
| │ │ └── errorMiddleware.js |
| │ ├── utils/ # Utility functions/helpers (e.g., token generation) |
| │ │ └── jwtUtils.js |
| │ ├── .env # Environment variables (e.g., database URI, secret keys) |
| │ ├── server.js # Entry point for Express server |
| │ └── package.json # Server dependencies and scripts |
| ├── .gitignore # Files to ignore in version control |
| ├── README.md # Project documentation |
| └── package.json # Root package file with common dependencies and scripts |

## APPLICATION FLOW:

* User Registration: Users sign up as either a patient or a doctor.
* Doctor Profile Creation: Doctors create profiles, set specializations, and update availability.
* Search and Book: Patients search for doctors, view availability, and book appointments.
* Appointment Management: Doctors and patients can view, reschedule, or cancel appointments.
* Notifications and Feedback: Patients receive appointment notifications, and feedback options are provided after consultation.

## PROJECT SETUP & CONFIGURATION:

* 1. **FRONT-END DEVELOPMENT:**
* Install Dependencies: Run npm install in the /client directory.
* Main Libraries: React, Axios (for API requests), Material-UI or Bootstrap (for styling).
* Component Structure: Design reusable components (e.g., Navbar, ProfileCard, Appointment Form).
* Routing: Use React Router for navigation.
* State Management: Context API or Redux for state sharing across components.

## 88.2 BACK-END DEVELOPMENT:

* Install Dependencies: Run npm install in the /server directory.
* Main Libraries: Express.js, Mongoose (for MongoDB interaction), JWT (for authentication).
* API Structure: Design RESTful APIs for managing users, appointments, and doctor profiles.
* Authentication: Use JWT to secure API routes and manage sessions.
* Middleware: Set up middleware for error handling, logging, and authentication**.**

## 8.3 DATABASE DEVELOPMENT:

* Database Setup: Use MongoDB Atlas or local MongoDB.
* Schema Design: Define schemas for User, Doctor, Appointment, and Feedback using Mongoose.
* Relationships: Connect users with appointments, doctors with profiles, and feedback with users.

1. **PROJECT IMPLEMENTATION:**

# Index.html:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1" />

<meta name="theme-color" content="#000000" />

<meta name="description"

content="Web site created using create-react-app"

/>

<title>Book a Doctor</title>

</head>

<body>

<noscript>You need to enable JavaScript to run this app.</noscript>

<div id="root"></div>

</body>

</html>

# Components:

Admin:

* 1. Admin appointment:

import React, { useEffect, useState } from 'react'; import Table from 'react-bootstrap/Table'; import Alert from 'react-bootstrap/Alert';

import { Container } from 'react-bootstrap'; import axios from 'axios';

const AdminAppointments = () => {

const [allAppointments, setAllAppointments] = useState([]) const getAppointments = async () => {

try {

const res = await axios.get('http://localhost:8001/api/admin/getallAppointmentsAdmin', {

headers: {

Authorization: `Bearer ${localStorage.getItem('token')}`

},

})

if (res.data.success) { setAllAppointments(res.data.data)

}

} catch (error) { console.log(error)

}

}

useEffect(() => { getAppointments();

}, [])

return (

<div>

<h2 className='p-3 text-center'>All Appointments for Admin Panel</h2>

<Container>

<Table className='my-3' striped bordered hover>

<thead>

<tr>

<th>Appointment ID</th>

<th>User Name</th>

<th>Doctor Name</th>

<th>Date</th>

<th>Status</th>

</tr>

</thead>

<tbody>

{allAppointments.length > 0 ? ( allAppointments.map((appointment) => { return (

<tr key={appointment.\_id}>

<td>{appointment.\_id}</td>

<td>{appointment.userInfo.fullName}</td>

<td>{appointment.doctorInfo.fullName}</td>

<td>{appointment.date}</td>

<td>{appointment.status}</td>

</tr>

)

})

) : (

<tr>

<td colSpan={6}>

<Alert variant="info">

<Alert.Heading>No Appointments to show</Alert.Heading>

</Alert>

</td>

</tr>

)}

</tbody>

</Table>

</Container>

</div>

)

}

export default AdminAppointments

# Admin Doctors:

import React, { useEffect, useState } from 'react'; import { Button } from 'react-bootstrap';

import Table from 'react-bootstrap/Table'; import Alert from 'react-bootstrap/Alert'; import { Container } from 'react-bootstrap'; import axios from 'axios';

import { message } from 'antd'; const AdminDoctors = () => {

const [doctors, setDoctors] = useState([])

const getDoctors = async () => { try {

const res = await axios.get('http://localhost:8001/api/admin/getalldoctors', { headers: {

Authorization: `Bearer ${localStorage.getItem("token")}`,

},

})

if (res.data.success) { setDoctors(res.data.data)

}

} catch (error) { console.log(error)

message.error('something went wrong')

}

}

const handleApprove = async (doctorId, status, userid) => { console.log(doctorId, status, userid)

try {

const res = await axios.post('http://localhost:8001/api/admin/getapprove', { doctorId, status, userid }, {

headers: {

Authorization: `Bearer ${localStorage.getItem("token")}`,

},

})

if (res.data.success) { message.success(res.data.message)

}

console.log(res)

} catch (error) { console.log(error)

message.error('something went wrong')

}

}

const handleReject = async (doctorId, status, userid) => { console.log(doctorId, status, userid)

try {

const res = await axios.post('http://localhost:8001/api/admin/getreject', { doctorId, status, userid }, {

headers: {

Authorization: `Bearer ${localStorage.getItem("token")}`,

},

})

if (res.data.success) { message.success(res.data.message)

}

console.log(res)

} catch (error) { console.log(error)

message.error('something went wrong')

}

}

useEffect(() => { getDoctors()

}, [])

return (

<div>

<h2 className='p-3 text-center'>All Doctors</h2>

<Container>

<Table striped bordered hover>

<thead>

<tr>

<th>Key</th>

<th>Name</th>

<th>Email</th>

<th>Phone</th>

<th>Action</th>

</tr>

</thead>

<tbody>

{doctors.length > 0 ? ( doctors.map((user) => { return (

<tr key={user.\_id}>

<td>{user.\_id}</td>

<td>{user.fullName}</td>

<td>{user.email}</td>

<td>{user.phone}</td>

<td>{user.status === 'pending' ?

<Button onClick={() => handleApprove(user.\_id, 'approved', user.userId)} className='mx-2' size='sm' variant="outline-success">

Approve

</Button>

:

<Button onClick={() => handleReject(user.\_id, 'rejected', user.userId)} className='mx-2' size='sm' variant="outline-danger">

Reject

</Button>}</td>

</tr>

)

})

) : (

<tr>

<td colSpan={5}>

<Alert variant="info">

<Alert.Heading>No Doctors to show</Alert.Heading>

</Alert>

</td>

</tr>

)}

</tbody>

</Table>

</Container>

</div>

)

}

export default AdminDoctors

* 1. AdminHome:

import React, { useEffect, useState } from 'react'

import axios from 'axios';

import { Link } from 'react-router-dom';

import CalendarMonthIcon from '@mui/icons-material/CalendarMonth'; import MedicationIcon from '@mui/icons-material/Medication';

import LogoutIcon from '@mui/icons-material/Logout';

import NotificationsIcon from '@mui/icons-material/Notifications'; import { Badge } from 'antd';

import Notification from '../common/Notification'; import AdminUsers from './AdminUsers';

import AdminDoctors from './AdminDoctors';

import AdminAppointments from './AdminAppointments';

const AdminHome = () => {

const [userdata, setUserData] = useState({})

const [activeMenuItem, setActiveMenuItem] = useState('');

const getUserData = async () => { try {

await axios.post('http://localhost:8001/api/user/getuserdata', {}, { headers: {

Authorization: "Bearer " + localStorage.getItem('token')

},

});

} catch (error) { console.log(error);

}

};

const getUser = () => {

const user = JSON.parse(localStorage.getItem('userData')) if (user) {

setUserData(user)

}

}

useEffect(() => { getUserData(); getUser()

}, []);

const logout = () => { localStorage.removeItem("token") localStorage.removeItem("userData") window.location.href = "/"

}

const handleMenuItemClick = (menuItem) => { setActiveMenuItem(menuItem);

};

return (

<>

<div className='main'>

<div className="layout">

<div className="sidebar">

<div className="logo">

<h2>MediCareBook</h2>

</div>

<div className="menu">

<div className={`menu-items ${activeMenuItem === 'adminusers' ? 'active'

: ''}`} onClick={() => handleMenuItemClick('adminusers')}>

<CalendarMonthIcon className='icon' /><Link>Users</Link>

</div>

<div className={`menu-items ${activeMenuItem === 'admindoctors' ? 'active' : ''}`} onClick={() => handleMenuItemClick('admindoctors')}>

<MedicationIcon className='icon' /><Link>Doctor</Link>

</div>

<div className="menu-items">

<LogoutIcon className='icon' /><Link onClick={logout}>Logout</Link>

</div>

</div>

</div>

<div className="content">

<div className="header">

<div className="header-content" style={{ cursor: 'pointer' }}>

<Badge className={`notify ${activeMenuItem === 'notification' ? 'active' : ''}`} onClick={() => handleMenuItemClick('notification')} count={userdata?.notification ? userdata.notification.length : 0}>

<NotificationsIcon className='icon' />

</Badge>

<h3>Hi..{userdata.fullName}</h3>

</div>

</div>

<div className="body">

{activeMenuItem === 'notification' && <Notification />}

{activeMenuItem === 'adminusers' && <AdminUsers />}

{activeMenuItem === 'admindoctors' && <AdminDoctors />}

{activeMenuItem !== 'notification' && activeMenuItem !== 'adminusers' && activeMenuItem !== 'admindoctors' && <AdminAppointments />}

</div>

</div>

</div>

</div>

</>

);

};

export default AdminHome;

1.4 : AdminUsers:

import React, { useEffect, useState } from 'react'; import Table from 'react-bootstrap/Table'; import Alert from 'react-bootstrap/Alert'; import { Container } from 'react-bootstrap'; import axios from 'axios';

const AdminUsers = () => {

const [users, setUsers] = useState([]) const getUsers = async()=>{

try {

const res = await axios.get('http://localhost:8001/api/admin/getallusers', { headers: {

Authorization : `Bearer ${localStorage.getItem("token")}`,

},

})

if(res.data.success){ setUsers(res.data.data) console.log(users)

}

} catch (error) { console.log(error)

}

}

useEffect(()=>{ getUsers()

},[])

return (

<div>

<h4 className='p-3 text-center'>All Users</h4>

<Container>

<Table className='my-3' striped bordered hover>

<thead>

<tr>

<th>Name</th>

<th>Email</th>

<th>Phone</th>

<th>isAdmin</th>

<th>isDoctor</th>

</tr>

</thead>

<tbody>

{users.length > 0 ? ( users.map((user) => {

return (

<tr key={user.\_id}>

<td>{user.fullName}</td>

<td>{user.email}</td>

<td>{user.phone}</td>

<td>{user.type}</td>

<td>{user.isdoctor === true ? 'Yes' : 'No'}</td>

</tr>

)

})

) : (

<Alert variant="info">

<Alert.Heading>No Users to show</Alert.Heading>

</Alert>

)}

</tbody>

</Table>

</Container>

</div>

)

}

export default AdminUsers

App.css:

@import "../node\_modules/bootstrap/dist/css/bootstrap.min.css";

\* {

padding: 0;

margin: 0;

box-sizing: border-box;

}

body {

background-color: rgb(190, 203, 203);

}

a {

color: black;

text-decoration: none;

margin-right: 20px;

}

.App { display: flex;

flex-direction: column;

min-height: 100vh;

}

.content { flex: 1;

}

.home-container { max-width: 100vw; height: 100vh; display: flex;

}

.left-side,

.right-side { flex: 1;

height: 100%; display: flex;

flex-direction: column; justify-content: center; align-items: center;

}

.left-side img { max-width: 100%;

height: 70%;

}

.right-side { display: flex;

align-items: center;

justify-content: center;

}

span {

font-family: Arial, Helvetica, sans-serif;

}

.f-letter {

font-size: 3em;

}

.s-letter {

font-size: 2.2em;

}

.t-letter {

font-size: 1.5em;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

.main { padding: 10px; height: 92.1vh;

/\* background-color: aqua; \*/

}

.layout { display: flex;

}

.sidebar {

min-height: 100%; height: 88vh; width: 17vw; border-radius: 5px;

background-color: rgb(153, 145, 145);

box-shadow: 0 0 2px grey; margin-right: 20px;

color: white;

}

.content { width: 100%;

height: 100%;

}

.header { height: 10vh;

margin-bottom: 20px; box-shadow: 0 0 2px grey;

background-color: whitesmoke; border-radius: 5px;

}

.body { height: 75vh;

margin-bottom: 20px; box-shadow: 0 0 2px grey;

background-color: whitesmoke; border-radius: 5px;

}

.logo h2 {

text-align: center; margin: 20px 0px;

}

.menu {

margin-top: 100px; display: flex;

flex-direction: column;

align-items: center;

}

.menu-items { margin-top: 20px;

}

.menu-items p { text-align: center; color: white;

text-decoration: none; font-size: 1.3rem; cursor: pointer;

}

.menu-items .icons {

font-size: 1.2rem; margin: 0 15px;

}

.header-content { display: flex;

align-items: center; height: 70px;

justify-content: flex-start; margin: auto 15px;

}

.notify {

position: relative; display: inline-block;

cursor: pointer;

}

sup{

margin-right: 15px;

}

.notify .icon { margin-right: 10px; font-size: 1.5rem;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

.box { display: flex;

flex-direction: column; justify-content: center; align-items: center; padding: 20px;

border-radius: 10px;

box-shadow: 0 0 2px grey; background-color: whitesmoke;

max-width: 700px; width: 100%;

}

.component { display: flex;

flex-direction: column; justify-content: center; align-items: center; margin: 20px; padding: 20px;

border: 2px dashed grey; border-radius: 10px; width: 100%;

height: 200px; text-align: center;

cursor: pointer;

}

.component p { margin: 10px 0; color: grey;

}

.btn-progress { display: flex;

justify-content: center; align-items: center; margin-top: 20px;

width: 100%;

}

.btn-progress button { color: grey;

padding: 10px 20px;

margin-right: 10px;

}

.btn-progress button:hover { box-shadow: 0 0 10px grey;

}

progress { width: 100%; height: 10px;

appearance: none; background-color: #f0f0f0; border-radius: 10px;

}

progress::-webkit-progress-bar { background-color: #f0f0f0;

border-radius: 10px;

}

progress::-webkit-progress-value { background-color: grey;

border-radius: 10px;

}

App.js:

import { BrowserRouter as Router, Routes, Route } from "react-router-dom"; import "./App.css";

import Home from "./components/common/Home"; import Login from "./components/common/Login"; import Register from "./components/common/Register";

import UserHome from "./components/user/UserHome"; import AdminHome from "./components/admin/AdminHome";

import UserAppointments from "./components/user/UserAppointments";

function App() {

const userLoggedIn = !!localStorage.getItem("userData"); return (

<div className="App">

<Router>

<div className="content">

<Routes>

<Route exact path="/" element={<Home/>} />

<Route path="/login" element={<Login/>} />

<Route path="/register" element={<Register/>} />

{userLoggedIn ? (

<>

<Route path="/adminhome" element={<AdminHome />} />

<Route path="/userhome" element={<UserHome />} />

<Route path="/userhome/userappointments/:doctorId" element={<UserAppointments />} />

</>

) : (

<Route path="/login" element={<Login />} />

)}

</Routes>

</div>

<footer className="bg-light text-center text-lg-start">

<div className="text-center p-3">© 2023 Copyright: MediCareBook</div>

</footer>

</Router>

</div>

);

}

export default App;

# Index.js:

import React from 'react';

import ReactDOM from 'react-dom/client'; import App from './App';

const root = ReactDOM.createRoot(document.getElementById('root')); root.render(

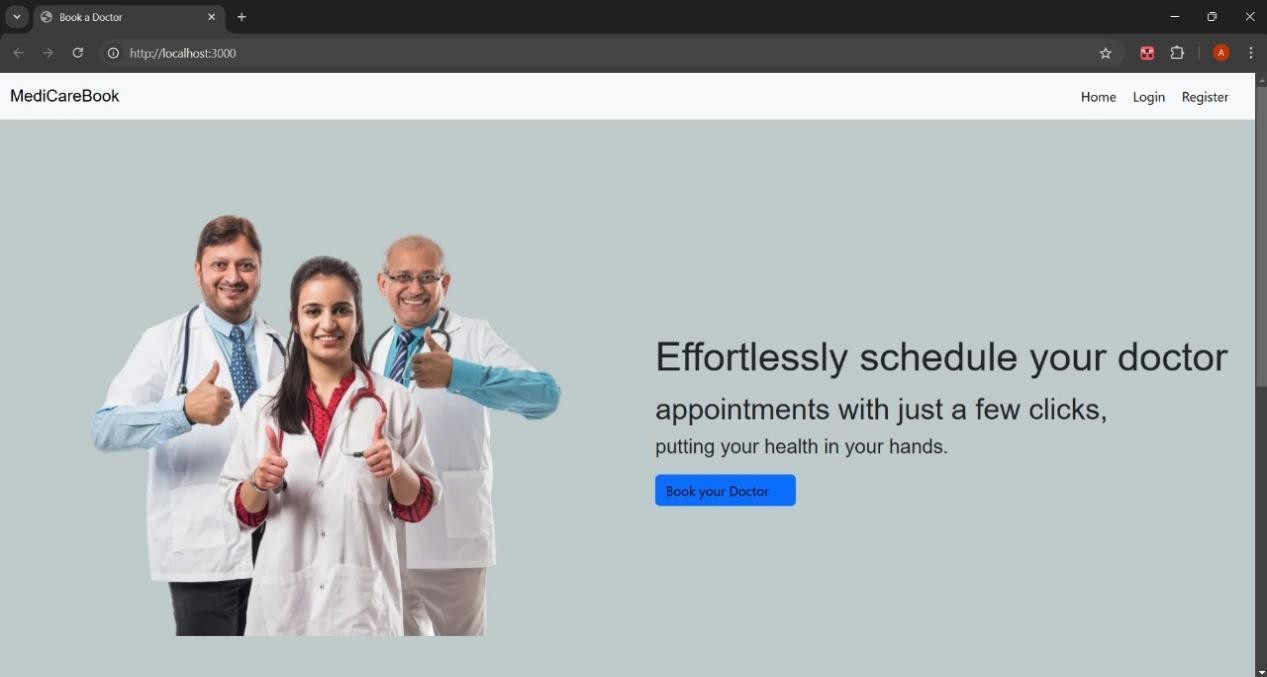
<React.StrictMode>

<App />

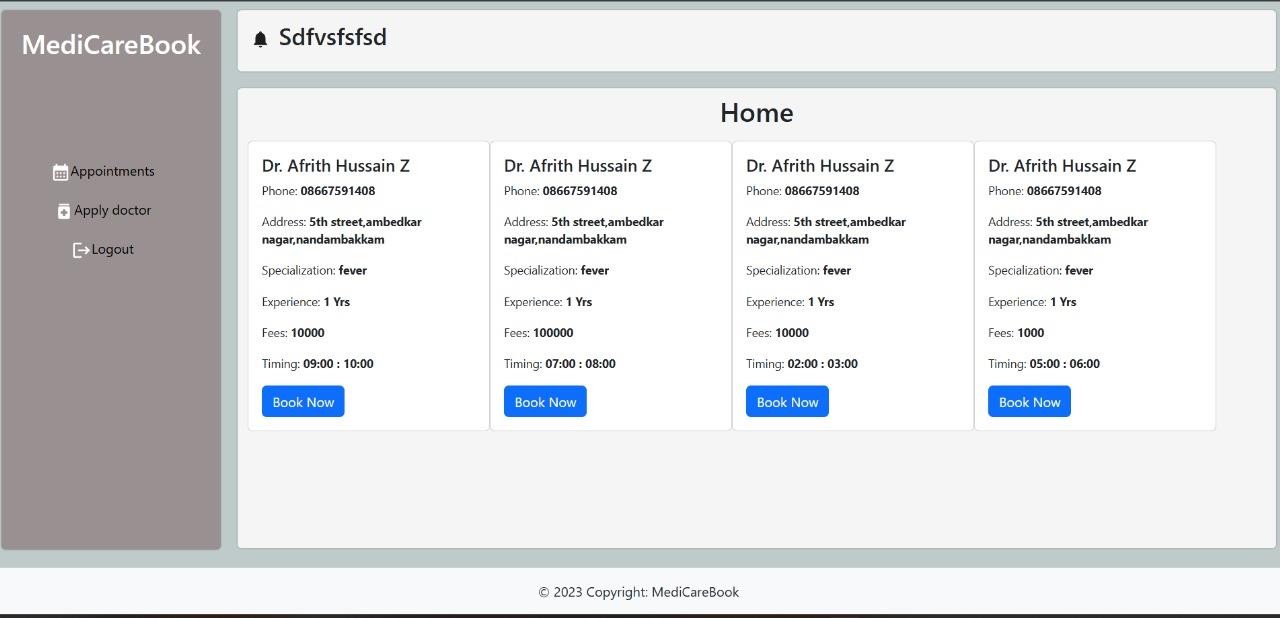
</React.StrictMode>

);

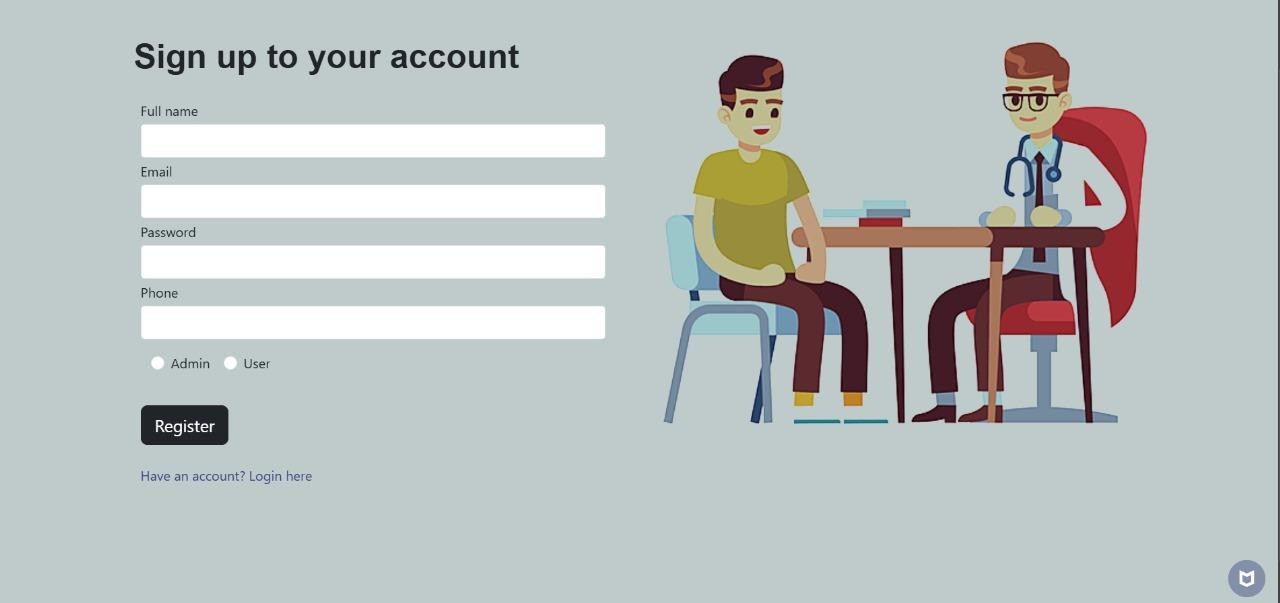
## OUTPUT

**Landing Page**

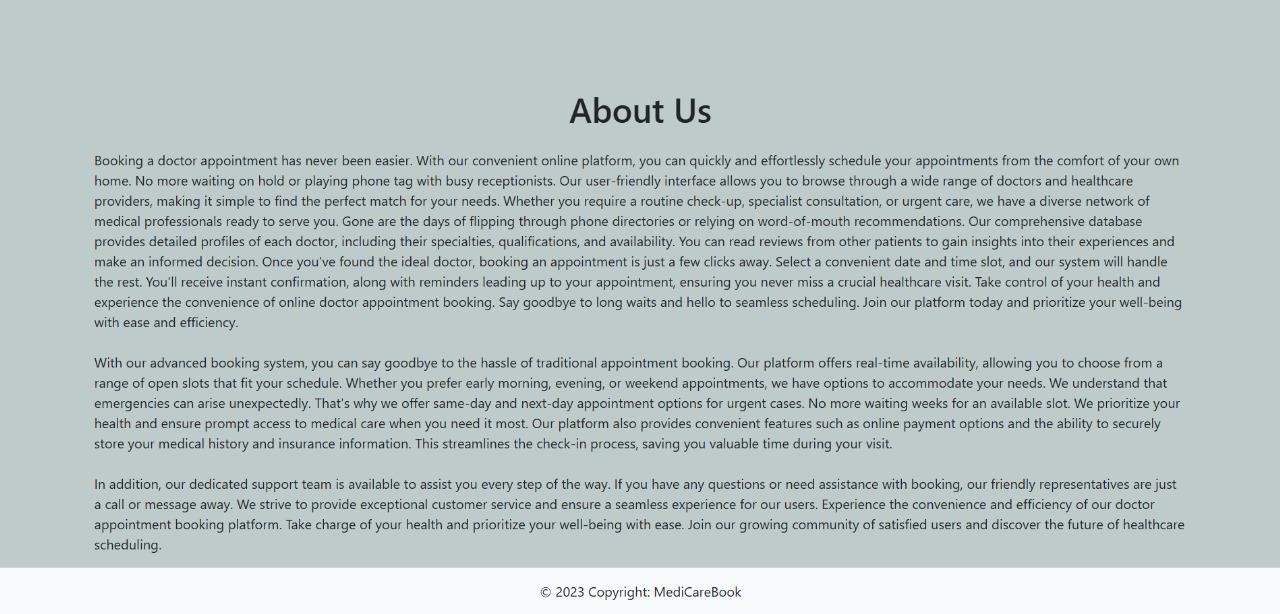
**User Doctor Booking-Page**



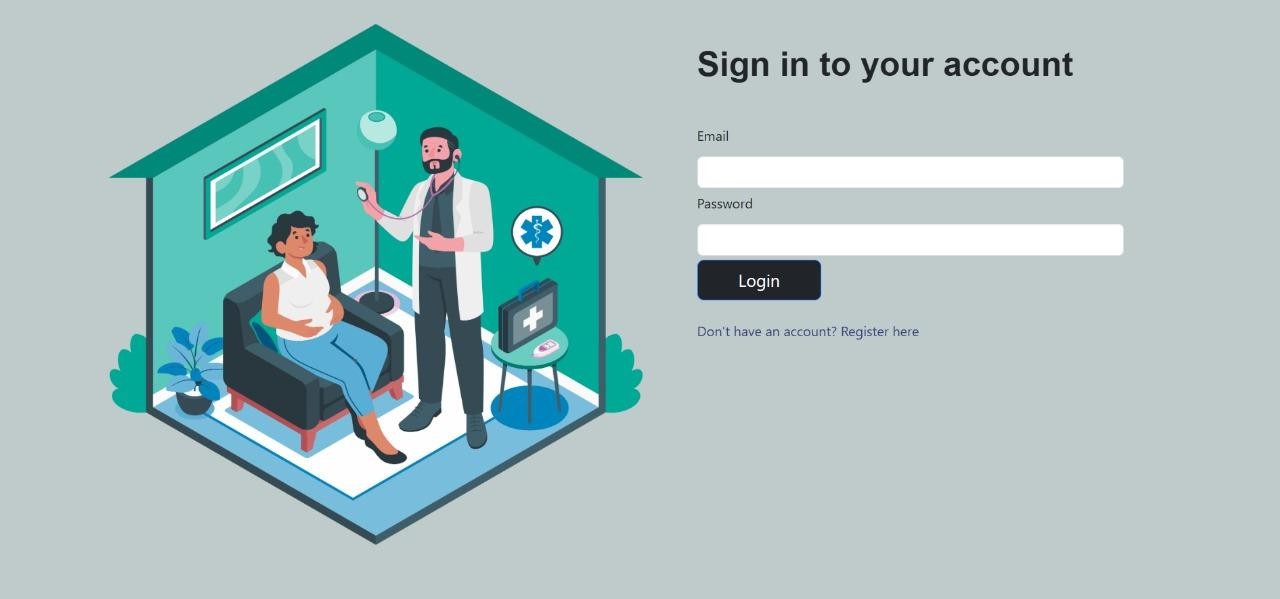
**Sign up Page**



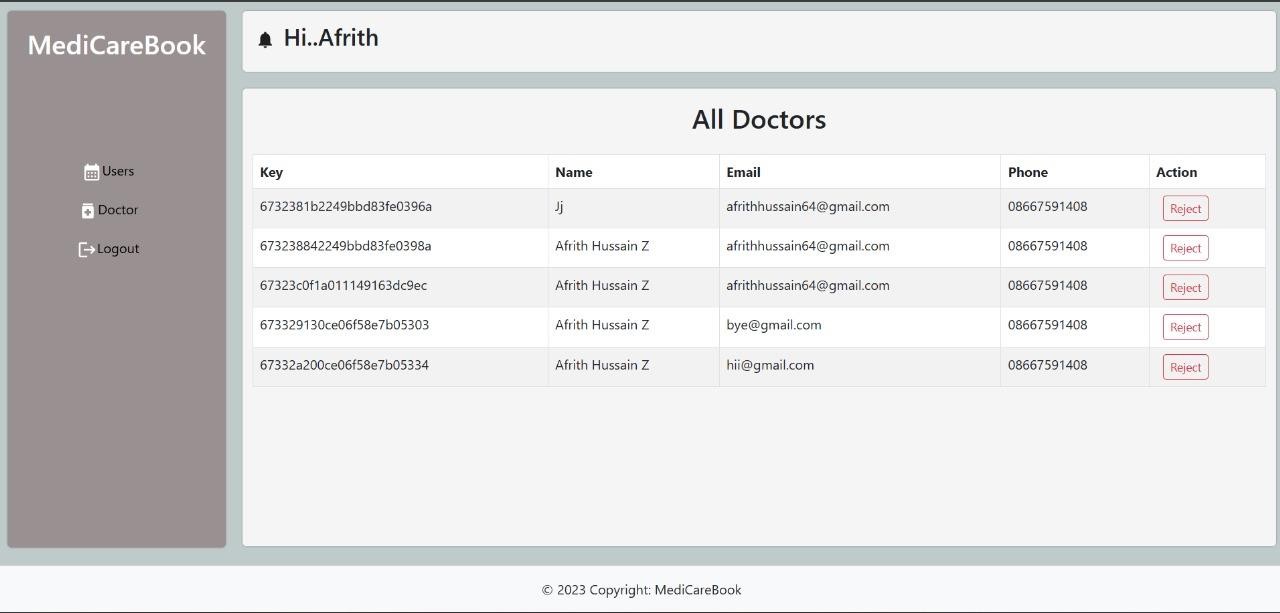
**About Us Page**



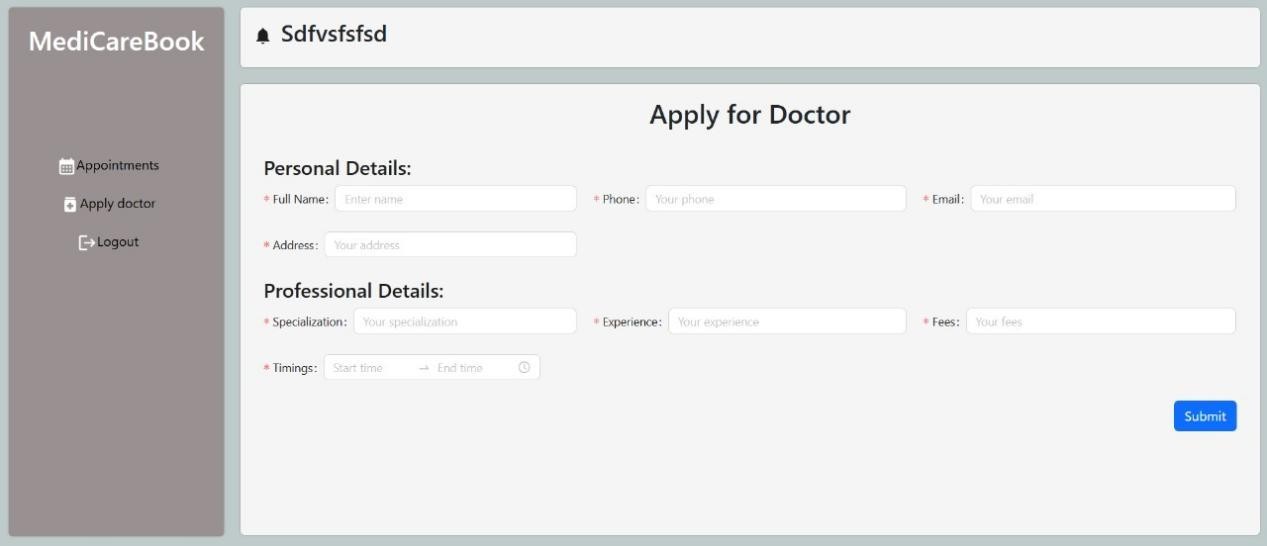
**Login Page**



**Admin portal page**

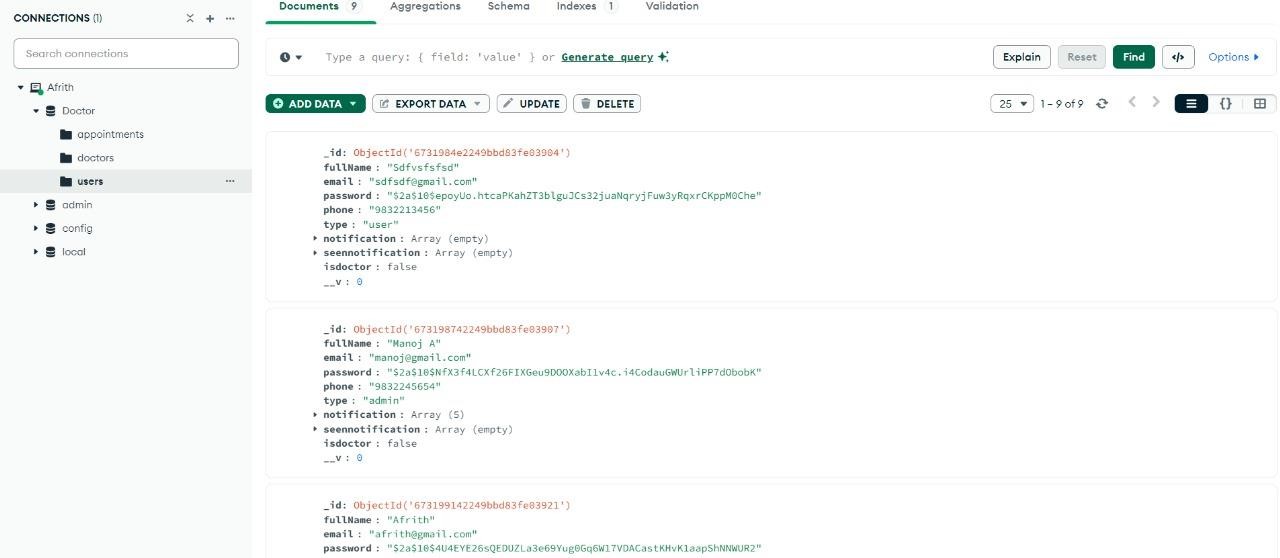


**Doctor Booking Dashboard Page**

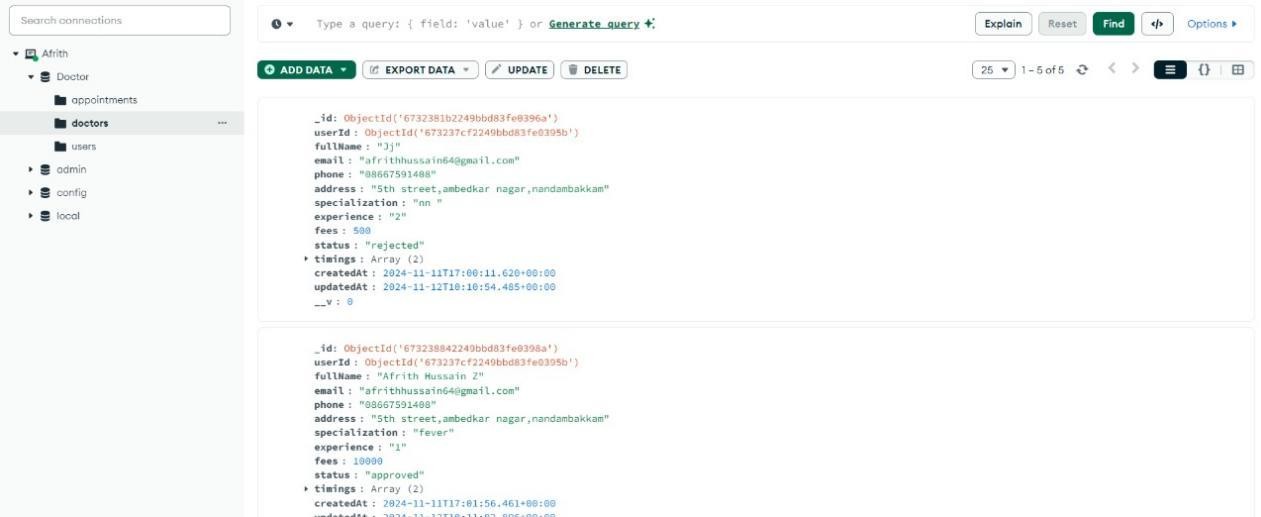


**MongoDB :**

**User sign in details page**



**Doctors login data-page**



## CONCLUSION

This project outlines the development of a comprehensive appointment booking system with user roles for customers, doctors, and admin. Each milestone—setup, backend, database, frontend, and final implementation—forms a structured foundation for a scalable application. The integrated functionalities allow for seamless appointment management, user authentication, and role-specific operations.

Team Members:

**Manoj**

**Afrith Hussain.Z Janani S Bhuvaneshwari E**