BOOK A DOCTOR USING MERN

1. Introduction

Project Title: Book a Doctor with MERN

TEAM MEMBERS:

S.NO	NAME	ROLE	RESPONSIBILITIES
1.	**Jaswanth V**	Full-Stack Developer	Responsible for overall development, Including front-end, back-end, server-side logic, and database design
2.	**Harini**	Frontend Developer	Responsible for UI/UX design using React, Material UI, and Bootstrap.
3.	**Jairus**	Database Administrator	Responsible for MongoDB setup and ensuring data integrity.
4.	**Harish Kumar**	Backend Developer	Responsible for Express.js setup and API development.

2. Project Overview

Purpose: The goal of the "Book a Doctor" application is to create a seamless online platform for booking doctor appointments. By connecting patients with healthcare professionals, the system allows users to find, book, and manage medical appointments with ease. It is intended to reduce the manual effort involved in scheduling appointments, making healthcare more accessible and efficient. This platform can cater to both individuals looking for a healthcare provider and doctors who want to manage their appointment schedules.

Features:

- **User Registration & Login:**
  - Patients and doctors can register with their personal information (name, email, and password) and log in securely using JWT-based authentication.
- **Doctor Directory:**
  - The app provides a list of doctors that users can filter based on specialties, location, and availability.
- **Appointment Booking:**
  - Patients can book an appointment with doctors, select preferred times, and upload documents like medical reports or prescriptions.
- **Admin Dashboard:**
  - Admins have access to manage doctor registrations, approve/reject doctor profiles, and monitor user activity.
- **Doctor Dashboard:**
  - Doctors can manage their appointments, view patient details, approve or reject booking requests, and update availability.
- **Notifications:**
  - Notifications are sent to users when their appointments are confirmed, canceled, or rescheduled.

2. Architecture

Frontend:

- The frontend is built with React.js, providing a component-based architecture that allows for reusable UI elements.
- The React Router handles navigation between pages such as the login page, dashboard, and appointment booking page.
- Material UI and Bootstrap are used for styling, ensuring the app has a modern, responsive Material UI provides ready-to-use components for consistent UI elements like buttons, text fields, and dialogs.
- Axios is used for making HTTP requests from the frontend to the backend, enabling communication with RESTful APIs.

Backend:

- The backend is powered by Node.js and Express.js. Express.js simplifies routing and middleware management.
- The MongoDB database is used to store user and doctor data, as well as appointment records. Mongoose ODM is used to define models and interact with MongoDB, allowing for easy schema-based data validation and manipulation.
- The backend is also responsible for handling authentication via JWT (JSON Web Token) and managing user sessions.

Database:

- The application uses MongoDB, a NoSQL database, to store and manage the data. MongoDB is chosen for its flexibility and scalability.
- Mongoose is used to define the application's schemas and models. The key models are:
- User Model: Stores patient and doctor information, including email, hashed password, and role (patient/doctor/admin).
- Doctor Model: Contains doctor details such as name, specialty, availability, and location.
- Appointment Model: Tracks appointment data, including patient ID, doctor ID, time, and status.

4. Setup Instructions

Prerequisites:

- **Node.js:** Make sure Node.js is installed. This is needed to run the application locally and install necessary dependencies.
- **MongoDB:** Ensure that you have a running instance of MongoDB. You can use either a local MongoDB setup or a cloud-based solution like MongoDB Atlas.
- **npm:** The Node Package Manager (npm) is required to install dependencies.

Installation:

1. Clone the Repository:

   git clone https://github.com/your-repo/book-a-doctor.git
2. Install Frontend Dependencies: Navigate to the client folder and run:

   cd client

   npm install
3. Install Backend Dependencies: Navigate to the server folder and run:

   cd server

   npm install
4. Set up Environment Variables: In the server folder, create a .env file to store sensitive information such as:

   MONGODB_URI: The connection string for your MongoDB database.

   JWT_SECRET: A secret key used for signing JWT tokens.

   PORT: Port on which the backend server will run (default is 5000).
5. Start the Backend Server: Run the following command in the server directory:

   npm start
6. Start the Frontend Server: Run the following command in the client directory:

   npm start

The application will be available at http://localhost:3000 (frontend) and http://localhost:5000 (backend).

5. Folder Structure

Client:

- **src/:** Contains all React components, hooks, and utilities.
- **components/:** Reusable React components such as Header, DoctorList, AppointmentForm, and LoginForm.
- **App.js:** Main entry point for the React application, managing routing and rendering components.
- **services/:** Contains files for Axios API calls to interact with the backend.

Server:

- **controllers/:** Contains the logic for handling requests, such as registering users, booking appointments, etc.
- **models/:** Defines Mongoose models for the MongoDB database (e.g., User, Doctor, Appointment).
- **routes/:** Defines the API endpoints (e.g., /api/doctors, /api/users).
- **middleware/:** Contains middleware for handling authentication, token verification, and error handling.
- **config/:** Stores configuration files like the database connection and environment variables.

6. Running the Application

Frontend: To start the frontend server, run the following command in the client folder:

```
npm start
```

Backend: To start the backend server, run the following command in the server folder:

```
npm start
```

7. API Documentation

1. **POST /api/auth/register:** Register a new user (either a patient or a doctor).

   - **Request Body:**

     {
     "email": "user@example.com",
     "password": "password123",
     "name": "John Doe",
     "type": "customer/doctor"
     }

   - **Response:**

     {
     "message": "Registration successful"
     }

2. POST /api/auth/login: Authenticate a user and receive a JWT token.

   - **Request Body:**

     {
     "email": "user@example.com",
     "password": "password123"
     }

   - **Response:**

     {
     "token": "jwt_token_here"
     }

3. GET /api/doctors: Fetch a list of doctors available for appointments.

- **Response:**

```
[
 {
"id": 1,
 "name": "Dr. John Doe",
"specialty": "Cardiologist",
 "location": "New York"
 }
]
```

4. POST /api/appointments/book: Book an appointment with a selected doctor.

- **Request Body:**

```
{
  "doctorId": 1,
"appointmentDate": "2024-12-01T10:00:00Z",
"documents": ["file1.pdf", "file2.pdf"]
}
```

- **Response:**

```
{
  "message": "Appointment booked successfully",
  "appointmentId": "12345"
}
```

8. Authentication

JWT Authentication: In this system, users (patients and doctors) authenticate by logging in and receiving a **JWT (JSON Web Token).** This token must be included in the authorization header of all future requests to protected routes. The token serves as proof that the user is authenticated and ensures that only authorized users can access specific resources, like booking appointments or accessing the doctor's dashboard.

Authorization Middleware:

- **Authentication Middleware:** This middleware verifies whether a user is logged in by checking the validity of the JWT token. If the token is valid, the user is allowed to proceed; otherwise, access is denied.

- **Authorization Middleware:** Once the user's identity is verified through authentication, this middleware checks if the user has the appropriate role to access a certain route. For example, only doctors should be able to access doctor-specific routes (e.g., managing their schedule), while patients should be restricted to booking appointments and viewing their own data.

9. User Interface

The User Interface (UI) of the Book a Doctor with MERN application is designed to provide a smooth and intuitive experience for the users. Below are the main UI components:

Login Page:

- Users can log in using their email and password.
- Features a clean design with clear input fields and a login button.
- Responsive design for mobile and desktop screens.

Dashboard (Customer):

- Displays a list of available doctors with basic information like name, specialty, and availability.
- Option to book an appointment by selecting a doctor, date, and uploading necessary documents.

Booking Form:

- After selecting a doctor, the user is directed to a form to book the appointment.
- The form includes fields for appointment date and document upload.

Doctor Dashboard:

- Doctors can view their upcoming appointments.
- Option to confirm or cancel appointments based on user actions.

Admin Dashboard:

- Admins can manage doctor applications, monitor all user bookings, and enforce platform policies.

GIFs showing the user flow and interactions (e.g., login, booking process, dashboard navigation) can also be provided.

10. Testing

Testing Strategy:

Unit Testing:

- Used for testing individual components and backend functions.
- Jest and Mocha were used for unit testing backend routes, validation, and middleware.

Integration Testing:

- Focused on ensuring that the communication between the frontend, backend, and database worked as expected.
- Tools like Postman were used to test API endpoints.

End-to-End Testing:

- Simulated the user journey from login to booking an appointment and managing it.
- Cypress was used for front-end testing, ensuring that all user interactions are properly handled.

Testing Tools:

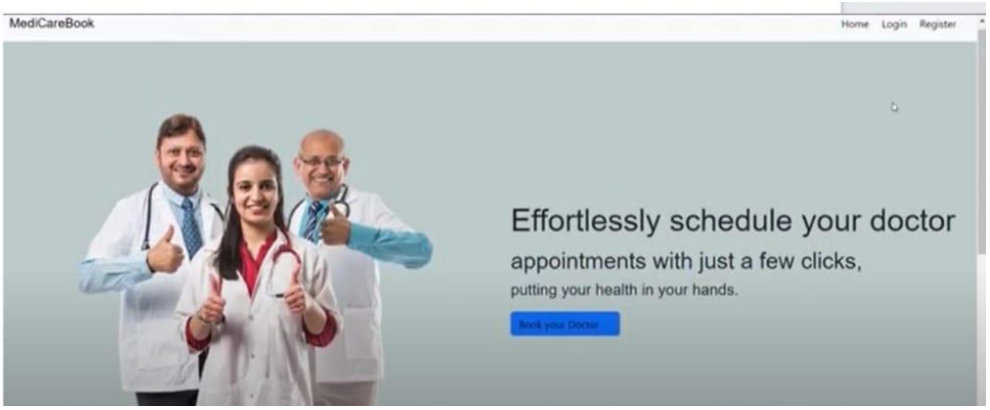
Frontend: Jest, Enzyme, React Testing Library for unit and integration testing of React components.

Backend: Mocha, Chai, Supertest for testing Express.js API routes.

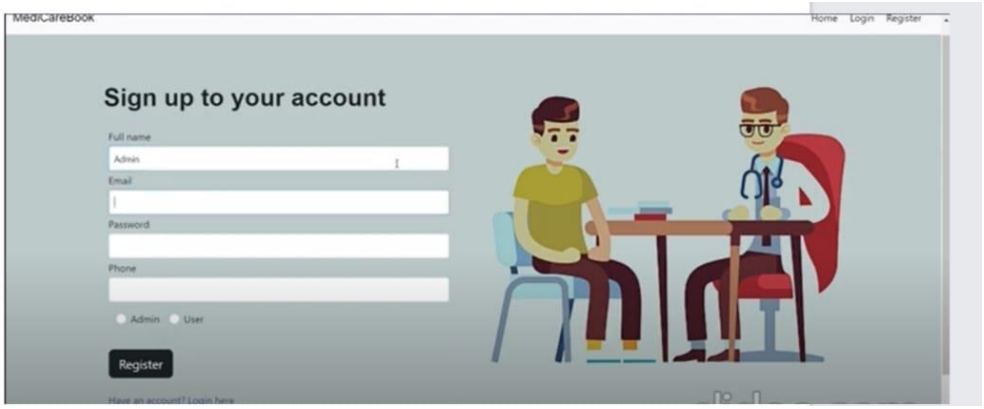
API Testing: Postman for manually testing backend endpoints.

11. Screenshots

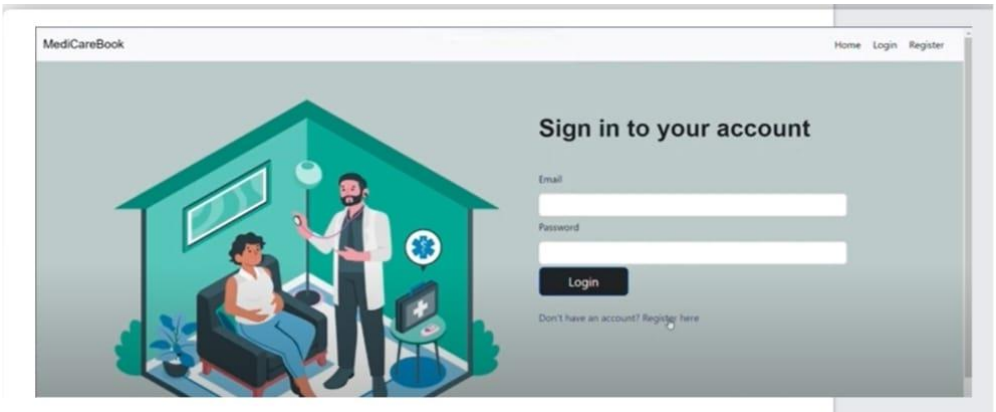
Landing page:



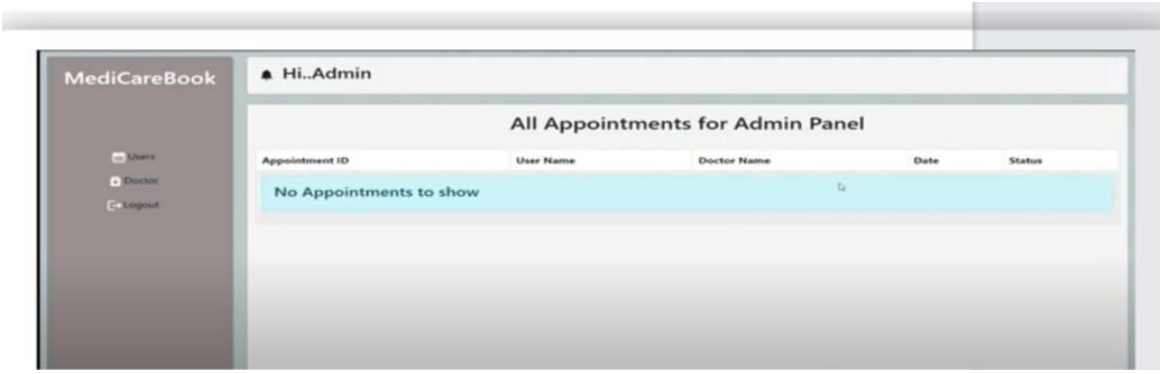
Register page:



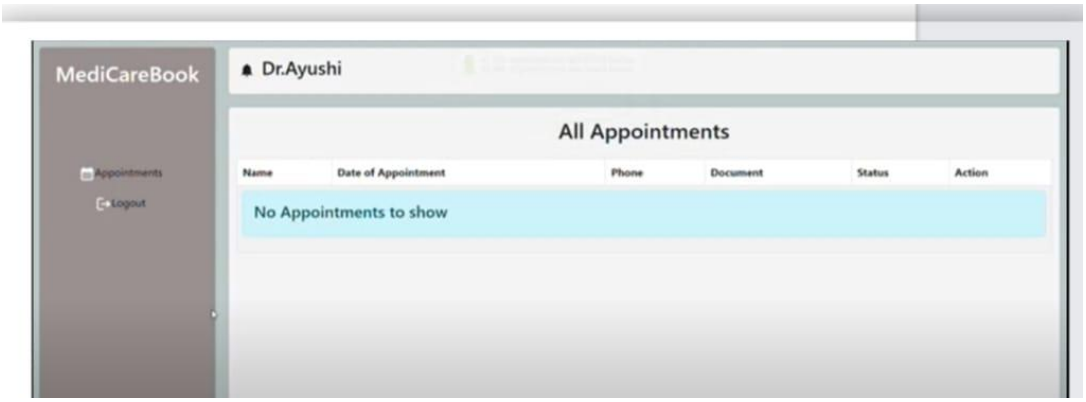
Login Page:



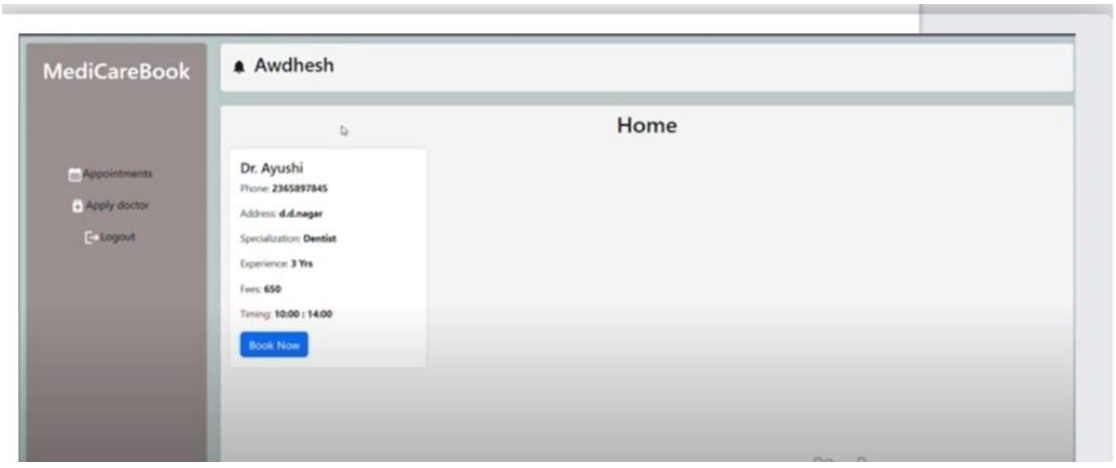
Admin Dashboard:



Doctor Dashboard:



User Dashboard:



12. Known Issues

While most of the application is fully functional, there are a few issues that developers or users should be aware of:

Issue with Appointment Confirmation:

- Occasionally, the appointment confirmation status may take a few moments to update due to delays in backend processing.

Responsive Layout on Mobile:

- Some UI components may not fully adjust for extremely small screen sizes (e.g., older smartphones).

File Upload Issue:

- The file upload feature for appointment documents may fail intermittently with larger files (greater than 10MB).

Workarounds:

- For appointment confirmation delays, refreshing the page can help load the updated status.
- Mobile users can zoom out to view the full content if it's cut off on smaller screens.
- For file uploads, users are encouraged to compress files before uploading.

13. Future Enhancements

There are several potential improvements and features that could be added to the Book a Doctor with MERN application:

Mobile App Integration:

- Developing native mobile applications for iOS and Android to enhance the user experience on mobile devices.

Telemedicine (Video Consultations):

- Adding a real-time video consultation feature, where users can have virtual appointments with doctors.

Payment Integration:

- Integrating payment gateways (e.g., Stripe or PayPal) to enable users to pay for consultations or book appointments for a fee.

Real-Time Chat:

- Implementing a messaging system that allows users and doctors to communicate directly before, during, or after an appointment.

AI-powered Doctor Recommendations:

- Using AI and machine learning algorithms to recommend doctors to users based on their health conditions, preferences, and history.

Advanced Search Filters:

- Enhancing the doctor search functionality with filters based on specialties, location, availability, and ratings.