**Project Title:**

Book a Doctor - Online Appointment Booking System

**Team Members:**

* GANESH ARVIND P - 311621205012
* SHYAM A - 311621205049
* TIRU VIKRAM S - 311621205056
* VIGNESH A - 311621205057

**Project Overview:**

Book a doctor is a web-based platform designed to simplify the process of booking doctor appointments online. It offers a user-friendly interface, a robust backend, and a seamless user experience. Key features include:

* User Registration and Login
* Doctor Profile Search and Filtering
* Appointment Scheduling
* Appointment Confirmation and Reminders
* Doctor Dashboard for Appointment Management
* Admin Dashboard for User and Doctor Management

**Architecture**

**Frontend:**

* **React:** A powerful JavaScript library for building user interfaces.
* **Redux:** A state management library for managing complex application states.
* **Material-UI:** A popular UI component library for creating beautiful and responsive user interfaces.
* **Axios:** A library for making HTTP requests to the backend API.

**Backend:**

* **Node.js:** A JavaScript runtime environment for building server-side applications.
* **Express.js:** A minimalist web framework for Node.js.
* **MongoDB:** A NoSQL database for storing user data, doctor profiles, and appointment information.
* **Moment.js:** A JavaScript library for manipulating dates and times.

**Database:**

* **Users:** Stores user information, including name, email, password, and contact details.
* **Doctors:** Stores doctor profiles, including name, specialization, availability, and contact information.
* **Appointments:** Stores appointment details, including patient ID, doctor ID, date, time, and status.

**Setup Instructions**

**Prerequisites:**

* Node.js and npm (or yarn) installed
* MongoDB database

**Installation:**

1. Clone the repository: git clone [https://github.com/SIMTASK/Doctor\_appoinment.git]
2. Install dependencies:
   * cd client
   * npm install
   * cd server
   * npm install
3. Set up environment variables:
   * Create a .env file in both the client and server directories.
   * Add the following variables:
     + REACT\_APP\_API\_URL: PSD:\naanmudhalvan\PROJECT\Shyam\server> node index.js
     + MONGODB\_URI: mongosh:SHYAM
4. Start the development server:
   * **Frontend:** **npm start** in the client directory
   * **Backend:** **node index.js** in the server directory

**Folder Structure**

**Client:**

* **src:** 
  + components: Reusable UI components
  + pages: Individual page components
  + store: Redux store and reducers
  + utils: Helper functions

**Server:**

* routes: API routes
* models: Mongoose models
* controllers: Controller functions
* config: Configuration files

**API Documentation**

**Endpoint:** /api/doctors

* **Method:** GET
* **Description:** Retrieves a list of all doctors.

**Endpoint:** /api/doctors/:id

* **Method:** GET
* **Description:** Retrieves a specific doctor's profile by ID.

**Endpoint:** /api/appointments/book

* **Method:** POST
* **Description:** Books a new appointment.

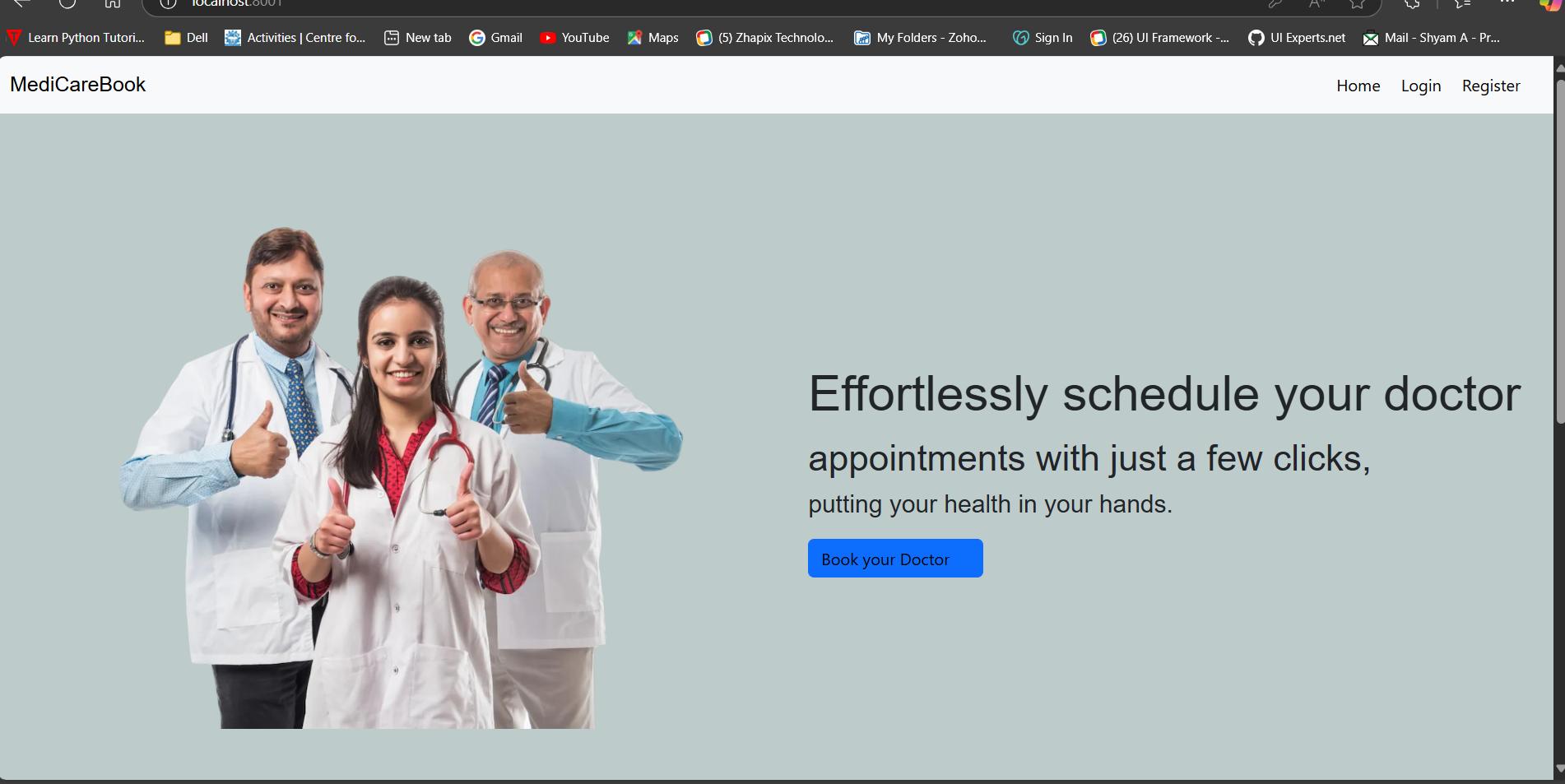
**Endpoint:** /api/appointments/cancel

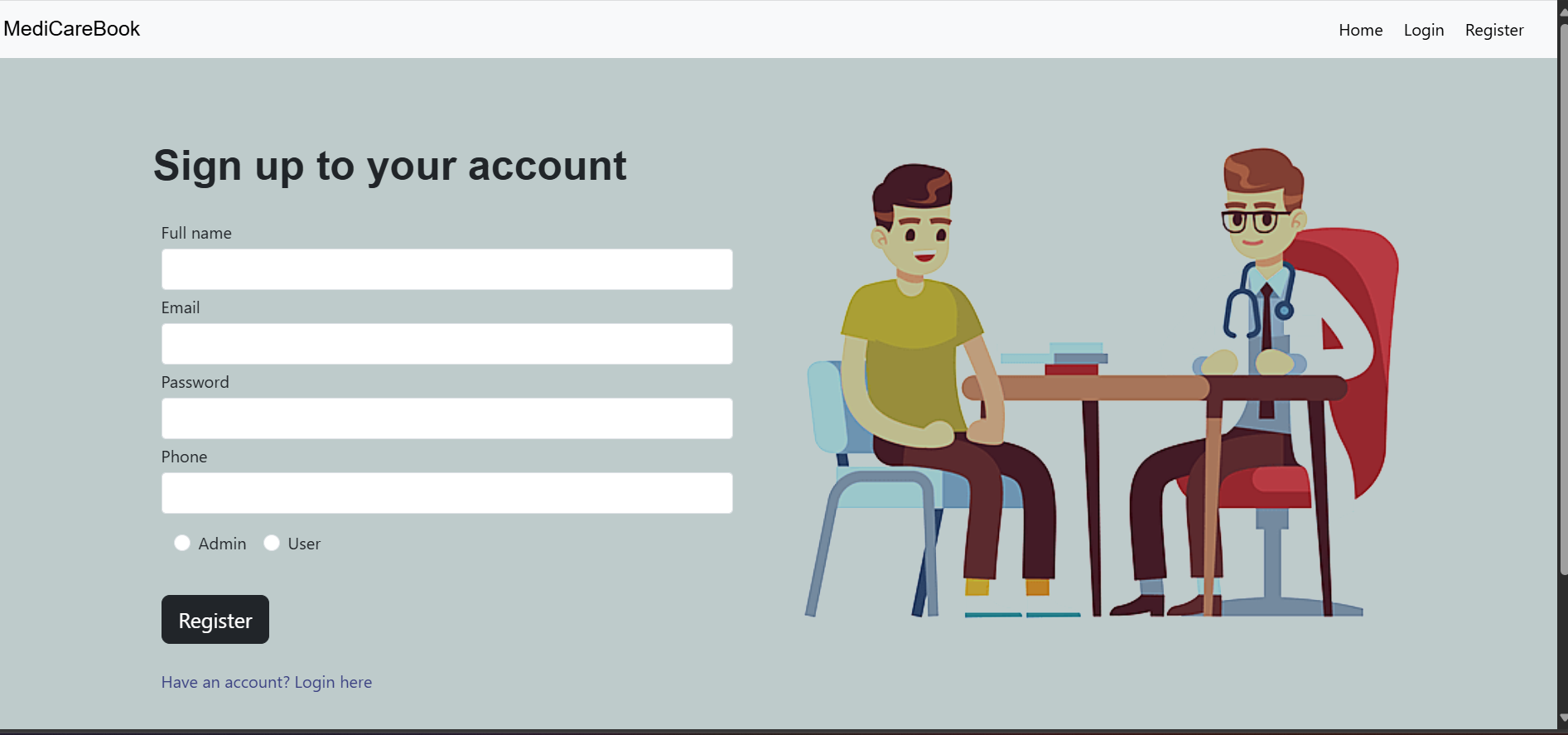
* **Method:** DELETE
* **Description:** Cancels an existing appointment.

**Authentication**

JWT-based authentication is used to protect API endpoints. Users receive a JWT token upon successful login, which is then included in the Authorization header of subsequent requests.

**User Interface**

****



**Testing:**

Unit tests are written using Jest for both frontend and backend components. Integration tests are conducted using tools like Postman to ensure proper API interactions.

**Known Issues:**

* There are no Such issues occur while creating but I faced a connection issue to data base then it’s been rectified.

**Future Enhancements:**

* Implement a rating and review system for doctors.
* Integrate with a payment gateway for online consultation fees.
* Enable video conferencing for remote consultations.
* Implement a notification system for appointment reminders and updates.
* Explore the use of AI-powered chatbots for initial patient inquiries.

**Additional Considerations:**

* **Security:** Implement measures to protect user data, such as strong password hashing, secure data transmission, and regular security audits.
* **Performance:** Optimize database queries, minimize network requests, and leverage caching techniques to improve application performance.
* **Scalability:** Design the application to handle increasing user loads and future growth.
* **Accessibility:** Ensure the application is accessible to users with disabilities by following web accessibility standards.
* **User Experience:** Continuously gather user feedback and iterate on the design and functionality to improve the user experience.