Full Stack Development with MERN

Project Document

1. Introduction

- **Project Title:** Book a Doctor Using Mern
- **Team Members:** Nirupama (2021503031)
  Krithika Ravi (2021503517)
  Rachitaa Sai J(2021503541)
  Rakshavihasini A S(2021503543)
  Shruthika L (2021503557)

2. Project Overview

- **Purpose:**
  The Book a Doctor system is designed to simplify and enhance the user experience for scheduling medical appointments. It caters to three different user roles: Admin, Patient, and Doctor. The primary goal of the project is to create a seamless and efficient platform for managing appointments, doctor availability, and patient records. The system ensures a smooth end-to-end experience, from booking appointments to consulting with doctors.

- **Features:**

  - **Streamlined Appointment Booking:** Provides patients with an easy-to-navigate platform where they can search for doctors, view their availability, and book appointments.
  - **Efficient Doctor Management:** Enables doctors to manage their schedules, view patient details, and keep records of consultation history in real time.
  - **Comprehensive Admin Control:** Allows admins to oversee and manage the entire platform, including user accounts, appointments, and system analytics.
  - **Enhanced User Experience:** Offers a user-friendly interface with smooth interaction flows for patients, doctors, and administrators.
  - **Secure Transactions:** Ensures secure and reliable payment processing for online consultations or appointment fees.

3. Architecture

Frontend: React Architecture
The frontend of the Book a Doctor system is built using React, following a component-based design approach to create reusable and scalable UI components.

- **Component-Based Structure**: The application is divided into modular components, such as:
  - Landing Page: Serves as the homepage with an overview of the service.
  - Register Page: Allows new users (patients and doctors) to register.
  - Login Page: Enables registered users to log into the platform.
  - Admin Dashboard: Provides administrative tools to manage platform operations.
  - Doctor Dashboard: Allows doctors to manage their schedules, view appointments, and update their profiles.
  - User Dashboard: Allows patients to view their booked appointments, update personal information, and manage bookings.

- **State Management**:
    - Uses React Context API or Redux for managing global state, ensuring consistency across components, such as appointment statuses and user profiles.
- **Routing**:
    - Employs React Router to manage navigation between pages (e.g., Home, Doctor List, Appointment Booking, User Dashboard), ensuring a seamless single-page application (SPA) experience.
- **API Integration**:
    - Utilizes Axios or Fetch API for asynchronous communication with backend services, allowing users to fetch doctor data, book appointments, and process payments.
- **Styling**:
    - Implements responsive design using CSS-in-JS libraries (e.g., Styled Components) or frameworks like Bootstrap or TailwindCSS to provide an engaging user interface.

Backend: Node.js and Express.js Architecture
The backend is developed using Node.js and Express.js, providing a robust, scalable, and efficient server-side environment.

- **RESTful API**:
    - Exposes a set of RESTful APIs to handle CRUD operations on users, appointments, and doctor data.
- **Authentication & Authorization**:
    - Implements JWT (JSON Web Token) for secure authentication and role-based access control, ensuring that patients, doctors, and admins have appropriate permissions.
- **Middleware**:
    - Custom middleware for error handling, logging, and validation of incoming requests is implemented using Express.js.
- **Security**:
    - Security best practices are implemented with CORS, helmet.js, and rate limiting to protect against common vulnerabilities.

3Database: MongoDB Schema and Interactions
The system uses MongoDB as the database due to its flexibility and scalability, with schemas defined using Mongoose.

- **User Schema**:
    - Includes essential details such as username, email, hashed password, role (patient, doctor, admin), and contact information.
- **Doctor Schema**:
    - Contains doctor-specific data like name, specialty, qualifications, availability schedule, consultation fees, and ratings.
- **Appointment Schema**:
    - Captures appointment details, including patient ID, doctor ID, appointment date and time, payment status, and consultation notes.
- **Admin Schema**:
    - Manages administrative data, including platform statistics, user logs, and system settings.

4. Setup Instructions

- **Prerequisites:**
  Node.js (v18+)
  MongoDB (v6+)
  Git for version control

  NPM for package management

- **Installation:**
  1. Install Dependencies
     Frontend:
     cd client
     npm install

     Backend:
     cd server
     npm install

5. Folder Structure

Client:

```
client/
├── public/
│   └── index.html
├── src/
│   ├── components/
│   │   ├── admin/
│   │   │   ├── AdminAppointments.jsx
│   │   │   ├── AdminDashboard.jsx
│   │   │   ├── AdminDoctors.jsx
│   │   │   ├── AdminHome.jsx
│   │   │   ├── AdminNotifications.jsx
│   │   │   └── AdminUsers.jsx
│   │   ├── common/
│   │   │   ├── Header.jsx
│   │   │   ├── Footer.jsx
│   │   │   ├── Home.jsx
│   │   │   ├── Login.jsx
│   │   │   ├── Notification.jsx
│   │   │   └── Register.jsx
│   │   └── user/
│   │       ├── AddDoctor.jsx
│   │       ├── ApplyDoctor.jsx
│   │       ├── DoctorList.jsx
│   │       ├── Notifications.jsx
│   │       ├── UserAppointments.jsx
│   │       └── UserHome.jsx
│   ├── images/
│   │   ├── p2.png
│   │   ├── p3.webp
│   │   └── photo1.png
│   ├── context/
│   │   └── AuthContext.jsx
│   ├── utils/
│   │   └── api.js
│   ├── App.jsx
```

```
    ├── App.css
    └── index.js
├── .gitignore
├── package-lock.json
└── package.json
```

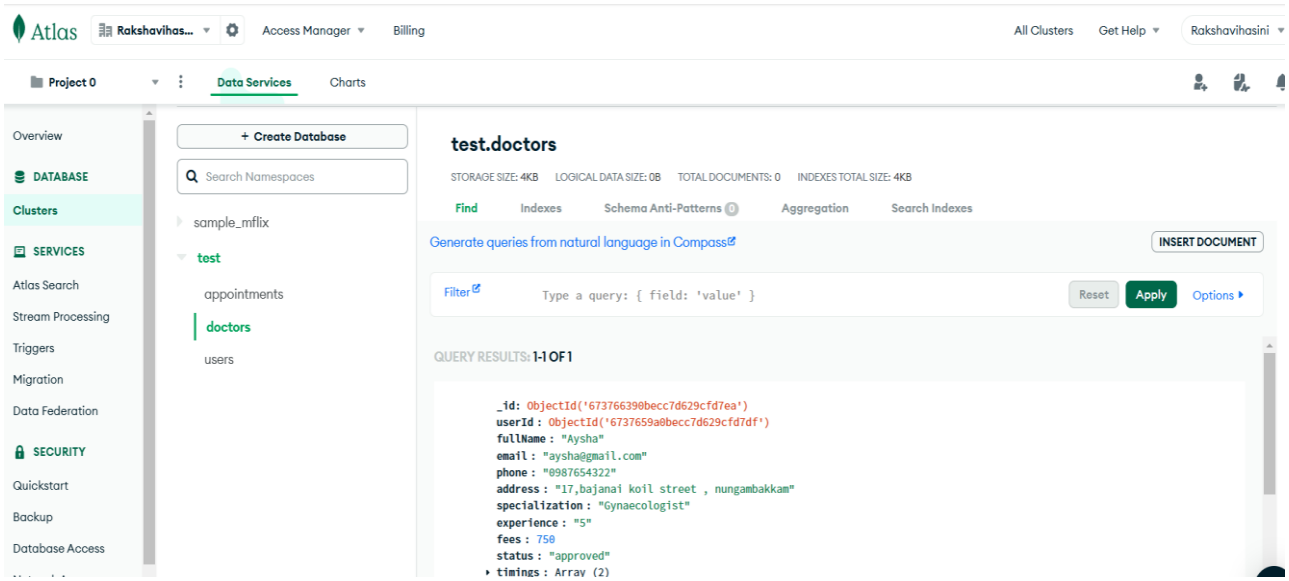
Server:

```
server/
├── config/
│   └── connectToDB.js
├── controllers/
│   ├── adminController.js
│   ├── doctorController.js
│   └── userController.js
├── middleware/
│   ├── authMiddleware.js
│   └── errorHandler.js
├── routes/
│   ├── adminRoutes.js
│   ├── doctorRoutes.js
│   └── userRoutes.js
├── schemas/
│   ├── appointmentModel.js
│   ├── docModel.js
│   └── userModel.js
├── uploads/
│   ├── <file1>
│   └── <file2>
├── .env
├── .gitignore
├── index.js
├── package-lock.json
└── package.json
```

6. Running the Application

- Commands to start the frontend and backend servers locally.
    - **Frontend:**
      cd client
      npm start
    - **Backend:**
      cd server
      node index.js

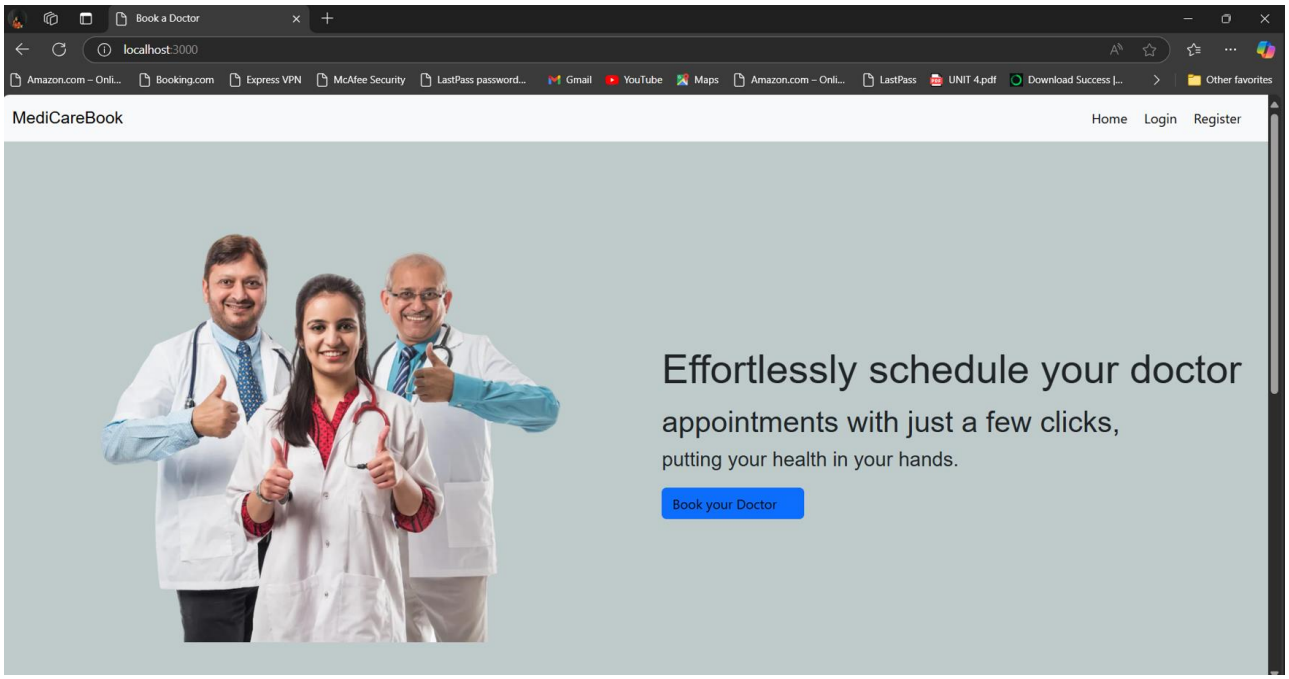
7. API Documentation



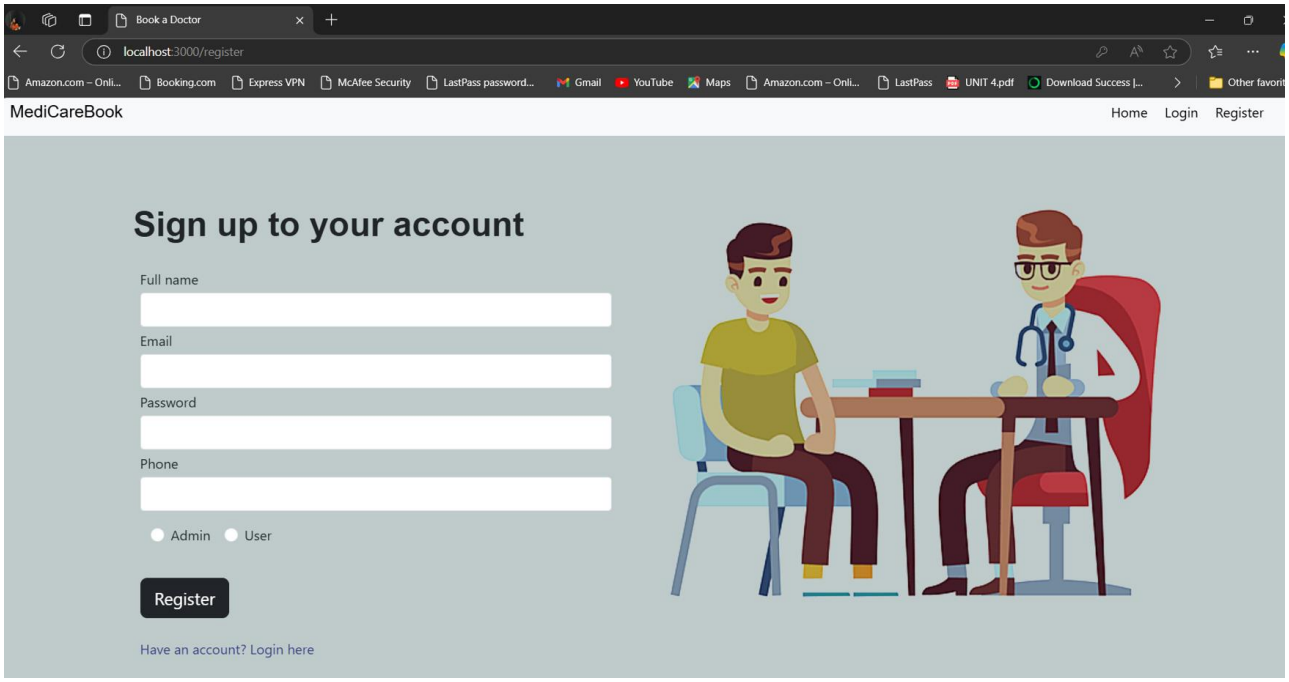
Endpoint	Method	Description
`/api/users/register`	POST	Register a new user
`/api/users/login`	POST	User login
`/api/doctors`	GET	Fetch list of available doctors
`/api/doctors/:id`	GET	Get doctor details by ID
`/api/appointments`	POST	Book a new appointment
`/api/appointments/:id`	GET	Get appointment details by ID
`/api/doctors/:specialty`	GET	Fetch doctors by specialty
`/api/doctors/:id/available`	GET	Fetch availability schedule for a doctor
`/api/users/:id`	GET	Get user profile details
`/api/appointments/user/:id`	GET	Fetch all appointments for a specific user

8. Screenshots or Demo

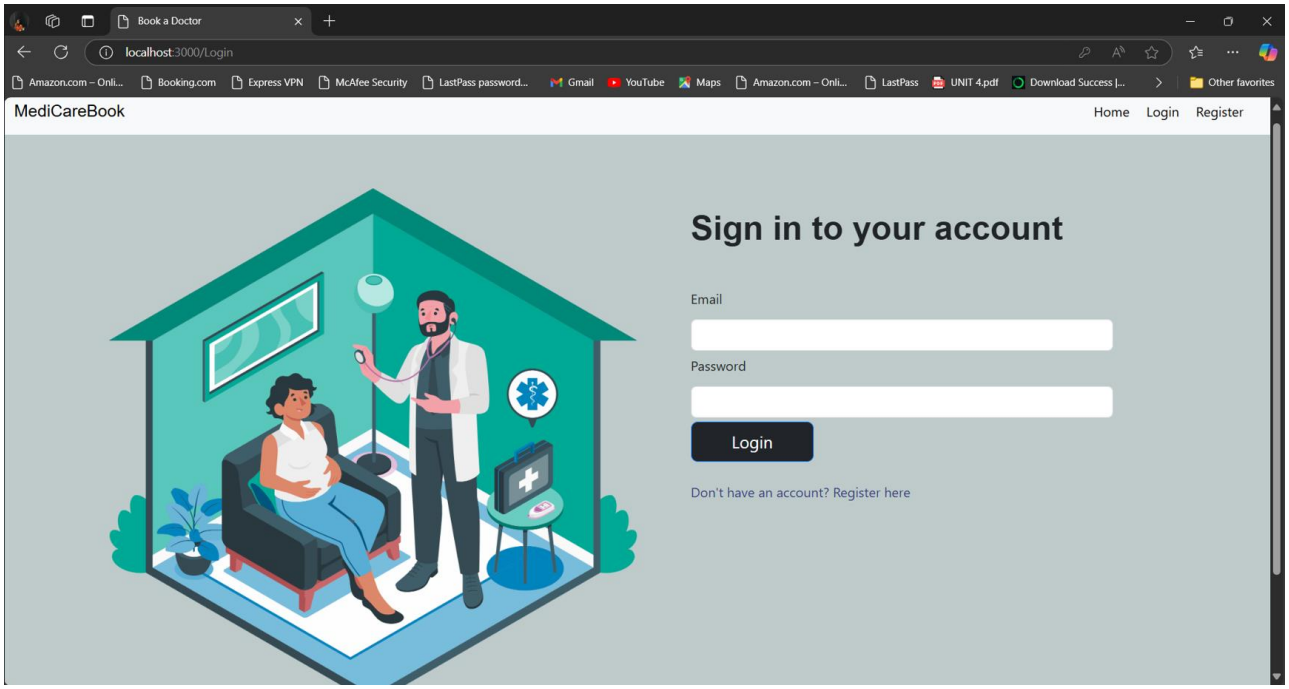
Landing page :



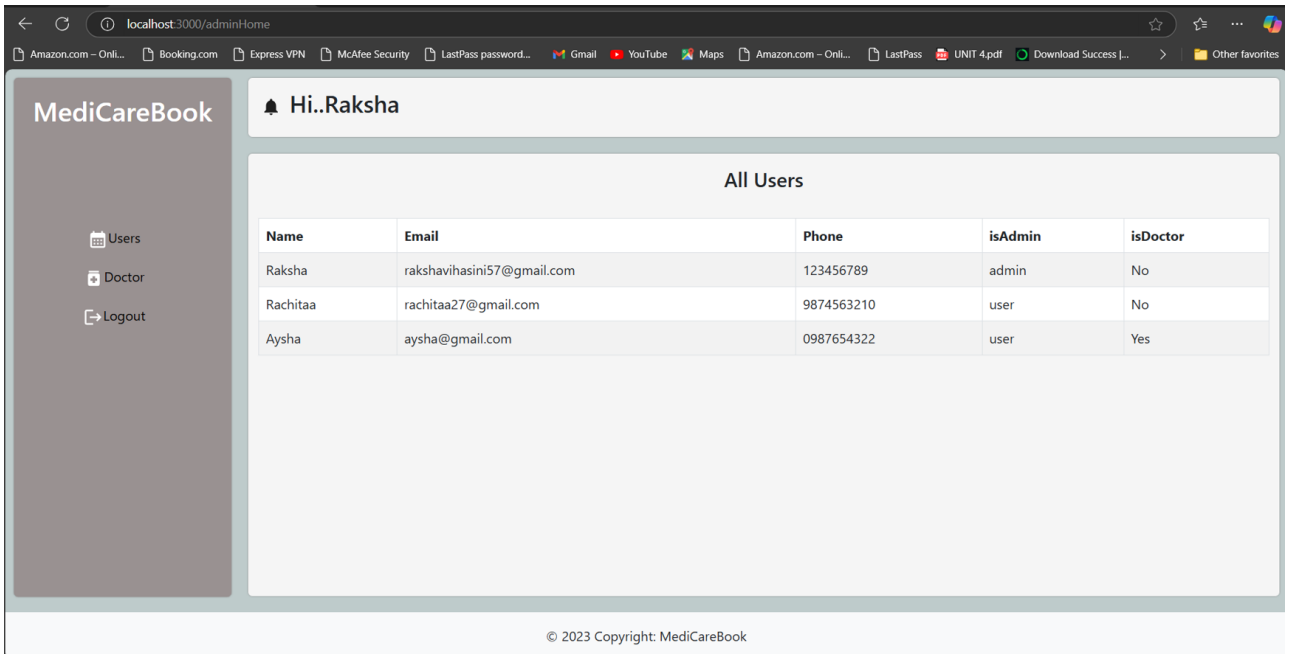
Register page:



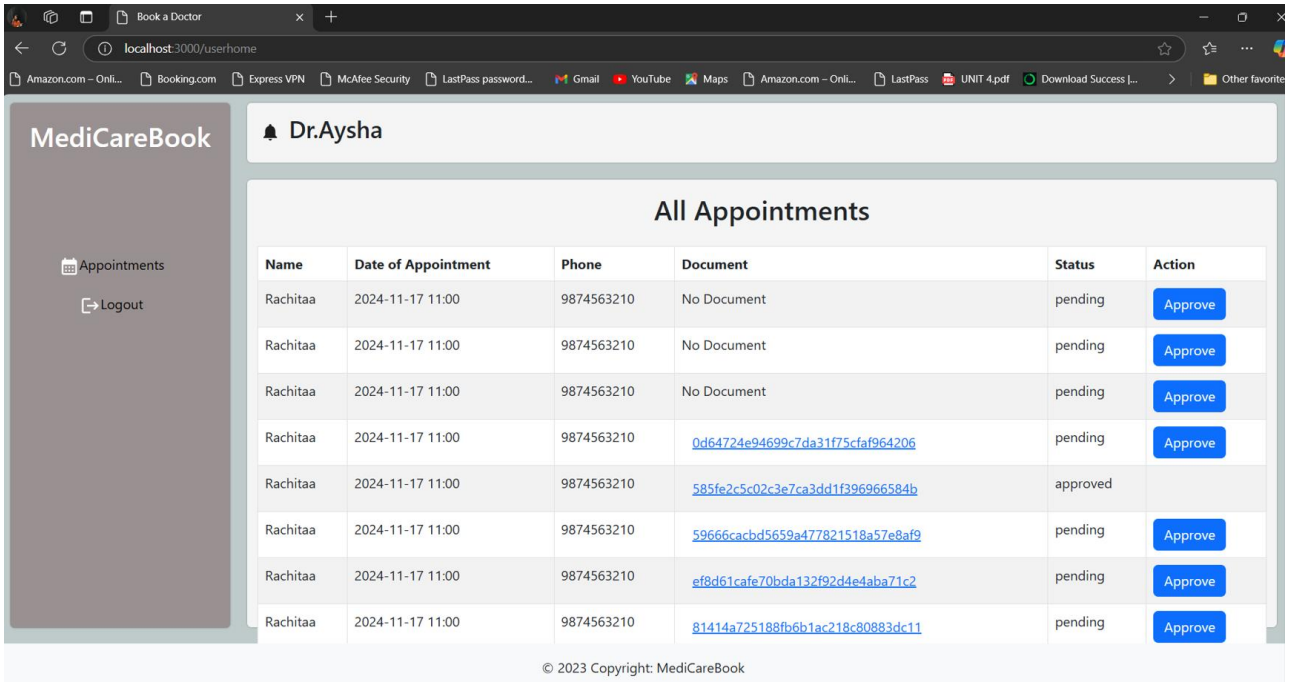
Login page:



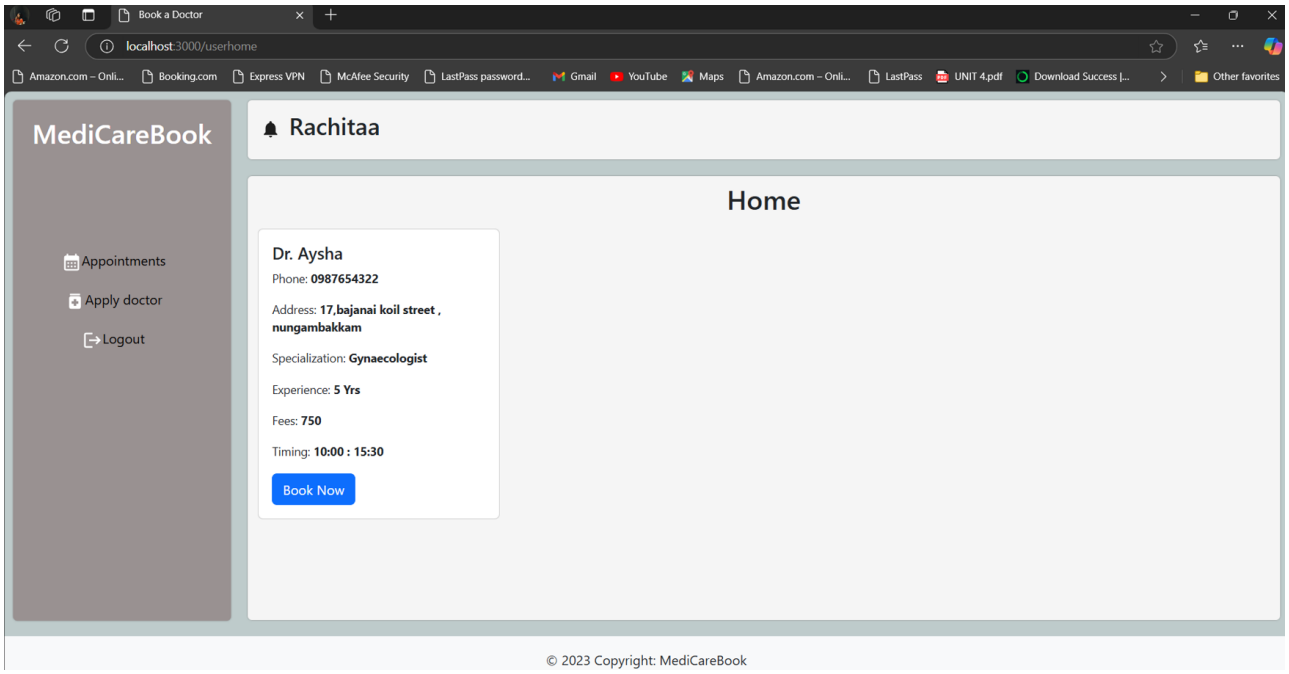
Admin Dashboard:



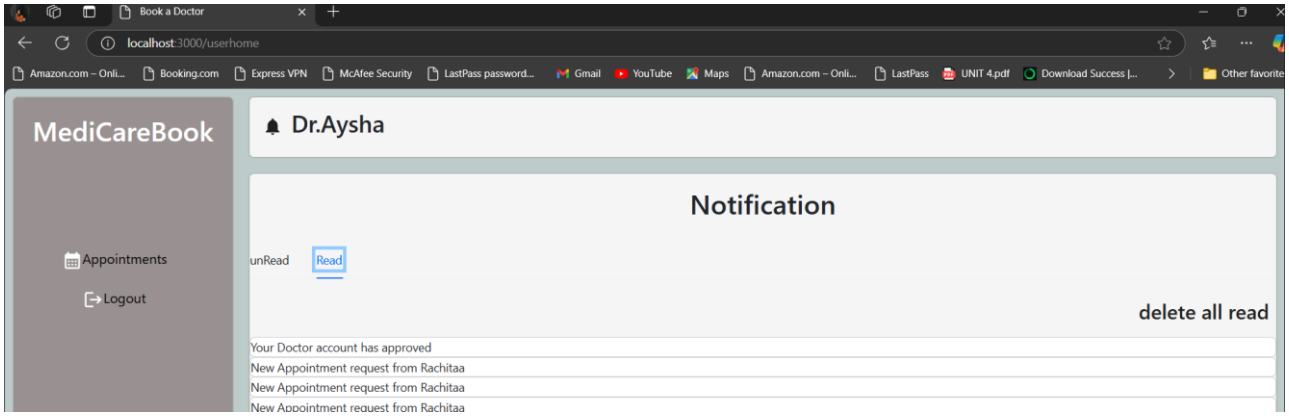
Doctor Dashboard:



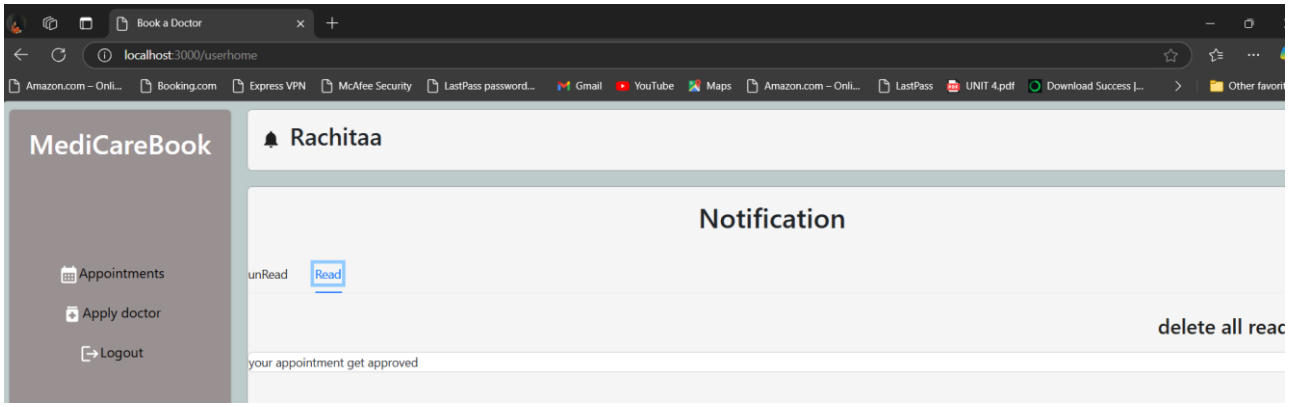
User Dashboard:



Doctor appointment request:



Doctor appointment approved notification:



9. Authentication

- **Login**: Users receive a JWT token upon login, which is stored on the client side for future requests.
- **Protected Routes**: Middleware verifies the JWT token to ensure users are authorized to access specific endpoints, like booking an appointment or managing appointments.
- **Role-Based Access Control (RBAC)**:
  - Admin: Has control over managing users, doctors, and appointments.
  - Patient: Can view doctors, book appointments, and check their booking status.
  - Doctor: Can manage their availability, view appointments, and mark them as completed.

10. User Interface

- Role-Based Access: UI adapts based on user roles (Admin, Customer, Doctor) with different permissions and views.
- Admin Dashboard: Provides analytics and management of doctors, users, and appointments.
- Doctor & Appointment Management: Users can view doctor availability, apply, and book appointments.
- Appointment Tracking: Customers and doctors can view and manage appointment status.
- Notification System: Keeps users updated with real-time appointment notifications.

11. Testing

Tools:

- Jest: Used for unit testing functions, React components, and hooks. For example, testing form validation logic in Login.jsx and Register.jsx, or testing the behavior of the AuthContext.jsx functions.
- Supertest: Used for API testing, ensuring that routes such as login, doctor management, and appointment booking return the correct response, especially verifying JWT token handling and role-based access control.
- React Testing Library: Used to test React components like Login.jsx, Register.jsx, AdminDashboard.jsx, and others, ensuring they render correctly and handle user interactions.
- Cypress: For end-to-end testing, simulating user flows such as signing in, booking an appointment, and checking status updates.

Testing Strategy:

- Unit Testing: Test individual components such as form validation in Login.jsx, token management in AuthContext.jsx, and CRUD operations for appointments and doctors.
- Integration Testing: Test integration between backend API and frontend components like DoctorList.jsx, ensuring that API calls for fetching doctors and booking appointments work seamlessly.
- End-to-End Testing: Using Cypress to simulate real-world interactions, such as logging in as a customer, selecting a doctor, booking an appointment, and managing appointment status.

12. Known Issues

- Session Expiry: Users may need to re-login after being inactive for long periods due to JWT token expiration.
- Order Synchronization: Occasionally, there may be delays in updating appointment statuses in the patient's UI.

13. Future Enhancements

- Real-Time Notifications: Implement Socket.io to provide real-time updates for appointment status changes and reminders.
- Mobile App: Develop a React Native mobile app to extend the platform for users on the go.
- Payment Gateway Integration: Integrate additional payment methods like Apple Pay and Google Pay for seamless appointment booking.
- AI-based Recommendations: Use AI to offer personalized doctor recommendations based on the user's past appointments and preferences.