Full Stack Development With MERN

Project Documentation

1. **Introduction**

   Project Title: [**Book a Doctor Using MERN**]

   Team Members: E. Sam Harrish., Dinesh kumar.R(TL), Swetha.R, Jagan.V

2. **Project Overview**

   Purpose

   The 'Book a Doctor Using MERN' project aims to streamline the healthcare appointment booking process by providing a responsive web application that connects patients and doctors on a single, easy-to-use platform. Users can search for doctors based on specialty, location, and availability, while doctors can manage their availability and view scheduled appointments in real time. This application serves as a bridge between patients and healthcare providers, enhancing accessibility to healthcare services.

   Features

   - User Registration and Login: Secure, role-based access for patients and doctors using JSON Web Tokens (JWT).
     - Profile Management: Allows both patients and doctors to manage their profiles, including personal information.
     - Doctor Search & Filtering: Enables users to search doctors by specialty, location, and other criteria.
     - Appointment Scheduling: Real-time booking to show availability and prevent double-booking.
     - Notifications: Confirmation notifications for booked appointments, with reminders as an optional feature.
     - Appointment History and Status Tracking: Allows users to track their appointments.
     - Admin Panel (Future Enhancement): Admins can manage roles, doctor verification, and system settings.

3. **Architecture**

   Frontend

   The frontend is built with React.js, creating a responsive UI through modular, reusable components. It uses React Context or Redux for state management and Axios for API requests. React Router provides seamless page navigation. The frontend includes pages for searching doctors, viewing profiles, booking appointments, and managing user accounts.

Backend
The backend, developed with Node.js and Express.js, handles server-side logic, including data processing, validation, and error handling. It uses middleware for JWT-based authentication, role-based access, and error handling, providing secure and RESTful API endpoints that connect the frontend and backend seamlessly.

Database
MongoDB is the primary database, chosen for its scalability and flexibility. Key collections include:
- Users: Stores patient and doctor profiles, roles, and authentication data.
- Doctors: Contains doctor profiles with details like specialties, experience, and availability.
- Appointments: Tracks bookings, linking patients with doctors, dates, and statuses (e.g., confirmed, completed, canceled).

4. Setup Instructions

Prerequisites
- Node.js and npm: Required for running both frontend and backend servers.
- MongoDB: For storing user, doctor, and appointment data, which can be set up locally or on MongoDB Atlas.

Installation
Clone the Repository:
```bash
git clone https://github.com/username/book-a-doctor.git
cd book-a-doctor
```

2. Backend Setup:
- Go to the `server` directory:
```bash
cd server
```
- Install dependencies:
```bash
npm install
```
- Create a `.env` file with these variables:
```
PORT=5000
MONGO_URI=your_mongodb_connection_string
JWT_SECRET=your_jwt_secret_key
```

```

3. Frontend Setup:
- Go to the `client` directory:
```bash
cd client
```
- Install dependencies:
```bash
npm install
```

5. Folder Structure

Client

- `src/`
  - `components/`: Reusable React components (e.g., DoctorCard, AppointmentForm).
  - `pages/`: Main pages like Home, DoctorProfile, Appointment, and UserDashboard.
  - `services/`: API calls using Axios.
  - `context/`: Holds the global state management setup (React Context or Redux).

Server

- `routes/`: Defines Express routes for resources (e.g., `/auth`, `/doctors`, `/appointments`).
  - `controllers/`: Business logic for processing requests.
  - `models/`: Mongoose schemas and models for MongoDB.
  - `middleware/`: Middleware for handling authentication and authorization.

6. Running the Application

To run the app locally:

Frontend:
```bash
cd client
npm start
```

Backend:
```bash

```
cd server
npm start
```

The app will now run on designated ports (`http://localhost:3000` for frontend and `http://localhost:5000` for backend).

7. **API Documentation**
   - **POST** `/auth/register`: Registers a new user.
   - **POST** `/auth/login`: Authenticates a user and returns a JWT.
   - **GET** `/doctors`: Fetches doctors based on search criteria.

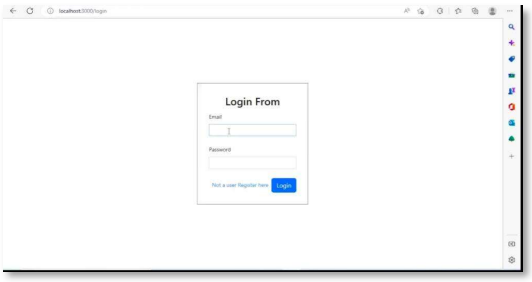
8. **Authentication**
   JWT-based authentication is used. After login, users receive a token stored client-side, which is included in headers for API requests. Backend middleware verifies the token, ensuring secure access to resources.

9. **User Interface**
   - **Home Page**: Allows users to search doctors by specialty or location.
   - **Doctor Profile**: Displays doctor details, availability, and ratings.
   - **Appointment Booking**: Interface for scheduling appointments.
   - **User Dashboard**: View and manage appointments.

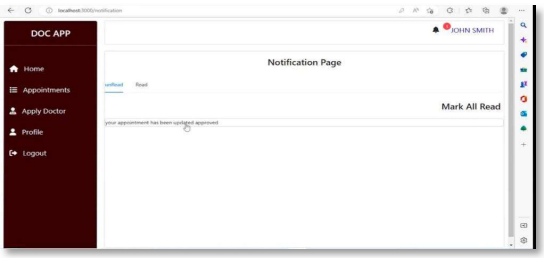
10. **Testing**
    - **Unit Testing**: Using Jest for React components and server-side functions.
    - **API Testing**: Using Supertest to verify API endpoints.
    - **End-to-End Testing**: Cypress is recommended for simulating user flows.
    -

11. **Screenshots or DEMO**

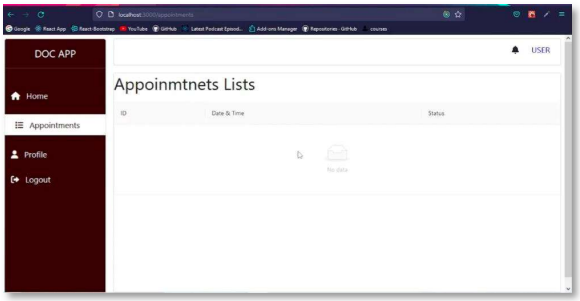
Login page:



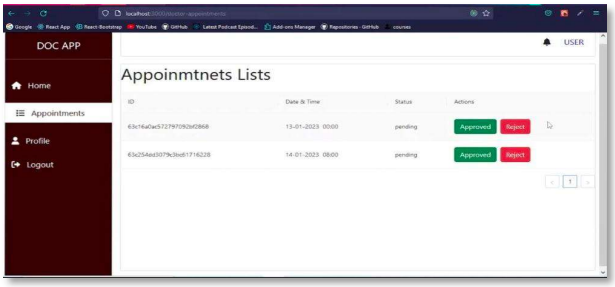
Notification page:



Appointment List:



Appointment List :



12. Known Issues

- **Availability Updates**: May have slight delays under heavy load.
- **Token Expiration**: Short session duration may need adjustment.

13. Future Enhancements

- **Push Notifications**: Real-time appointment reminders.
- **Ratings & Reviews**: Allows patients to rate doctors.
- **Admin Dashboard**: Centralized admin controls.
- **Enhanced UI**: Additional animations and a refined design.