Full Stack Development with MERN

Project Documentation

Introduction

- **Project Title**: Book A Doctor using MERN
- **Team Members**:
  - **Julfath Sahania N** : Frontend Developer
  - **Jebasti** : Backend Developer
  - **Bhuvanesh**: Database Architect
  - **Akula Sumanth** : Quality Assurance Engineer

Project Overview

- **Purpose** :

  The "Book A Doctor" application was developed to simplify the process of booking medical appointments. By connecting patients with doctors through a digital platform, it aims to improve accessibility, reduce waiting times, and streamline the appointment booking experience. This application is ideal for patients looking for medical consultations and doctors seeking a platform to manage their appointments.

- **Key Features**
  - **User Authentication**: Patients and doctors can sign up, log in, and manage their profiles.
  - **Doctor Search**: Users can search for doctors based on specialization, location, and availability.
  - **Appointment Booking**: Patients can book appointments, check available time slots, and cancel appointments if necessary.
  - **Doctor Profiles**: Doctors can create and update their profiles with information about their specialization, education, and available times.
  - **Admin Panel**: Admins can manage users, doctors, and appointments.
  - **Notifications**: Appointment reminders sent to users to ensure timely visits.

Architecture

- **Frontend**:
  The frontend is built using **React.js**, which provides a dynamic, responsive, and fast user interface. The application uses **React Router** for smooth navigation between pages and **React Context API** for state management across components. API requests are made using **Axios**.

- **Backend**:
  The backend is powered by **Node.js** and **Express.js**, which handle server-side logic, API routes, and communication with the database. The application is designed around RESTful principles, with endpoints for handling users, doctors, appointments, and more. The backend also includes middleware for tasks like authentication and error handling.

- **Database**:
  MongoDB is used to store data. It is a NoSQL database that is flexible and scalable, making it ideal for this application. Collections in the database include:

    - **Users** (patients, doctors, admins)
    - **Appointments** (appointments between patients and doctors)
    - **Doctors** (doctor profiles with information about their specialties and availability)

Example schema for **appointments**:

json

```
{
  "patient_id": ObjectId,
  "doctor_id": ObjectId,
  "appointment_time": Date,
  "status": String  // e.g. "booked", "completed", "cancelled"
}
```

Setup Instructions

- **Prerequisites**:
    - **Node.js** (version 14.0 or higher)
    - **MongoDB** (either local or MongoDB Atlas)
    - **npm** (Node Package Manager)
- **Installation**:

1.Clone the repository:

git clone https://github.com/your-repository/book-a-doctor.git

2.Navigate into the project directory:

cd book-a-doctor

3.Install dependencies for both the client and server:

cd client

npm install

cd ../server

npm install

4.Set up environment variables: Create a .env file in both the root and server directories:

env

MONGO_URI=mongodb://your-database-uri

JWT_SECRET=your-secret-key

5.Run the frontend and backend:

Frontend:
Navigate to the client directory and run:

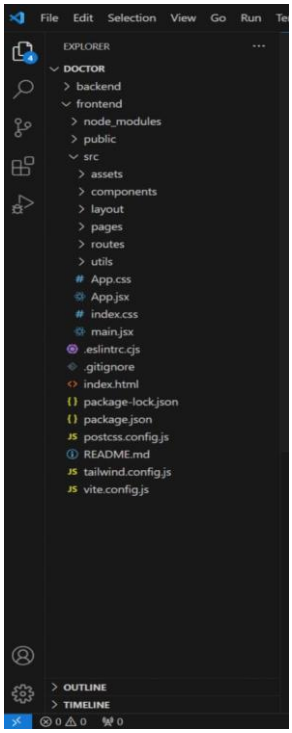
npm start

Backend:
Navigate to the server directory and run:

npm start

Now the application should be running locally. The frontend will be accessible at http://localhost:3000, and the backend API will be at http://localhost:5000.

Folder Structure



Running the Application

To run the application locally, you'll need to start both the frontend and backend servers:

Frontend:
In the client directory, run:

npm start

Backend:
In the server directory, run:

npm start

The frontend should be available at http://localhost:3000, and the backend API at http://localhost:5000.

API Documentation

Here is an overview of the main API endpoints:

POST /api/users/register:

- Register a new user (patient or doctor).
- Request body:

json

```
{
  "username": "JohnDoe",
  "email": "johndoe@example.com",
  "password": "securepassword",
  "role": "patient/doctor"
}
```

Response:

json

```
{
  "message": "User registered successfully",
  "user": { "id": "user_id", "username": "JohnDoe" }
}
```

POST /api/users/login:

- Log in a user and receive a JWT token.
- Request body:

json

```
{
  "email": "abcd@example.com",
  "password": "securepassword"
}
```

Response:

json

```
{
  "token": "jwt_token_here"
}
```

GET /api/doctors:

- Fetch a list of doctors, optionally filtered by specialization or location.
- Query parameters: specialization, location
- Response:

```
json
[
  {
    "id": "doctor_id",
    "name": "Dr. John Doe",
    "specialization": "Cardiology",
    "available_times": ["2024-11-20T10:00", "2024-11-20T14:00"]
  }
]
```

POST /api/appointments:

- Book a new appointment with a doctor.
- Request body:

```
json
{
  "doctor_id": "doctor_id",
  "appointment_time": "2024-11-20T10:00"
}
```

Response:

```
json
{
  "message": "Appointment booked successfully",
  "appointment": { "id": "appointment_id", "status": "booked" }
}
```

Authentication

- **JWT Authentication**:

- Users log in and receive a **JWT token** that must be included in the header of any protected API requests:

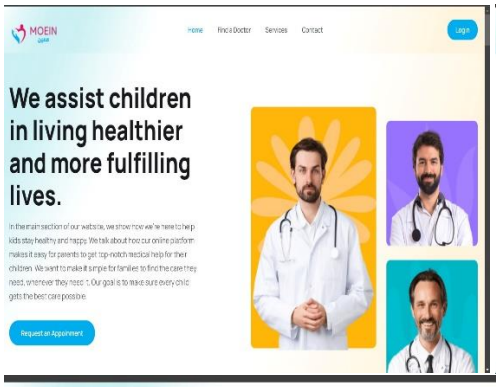
Authorization: Bearer <token>

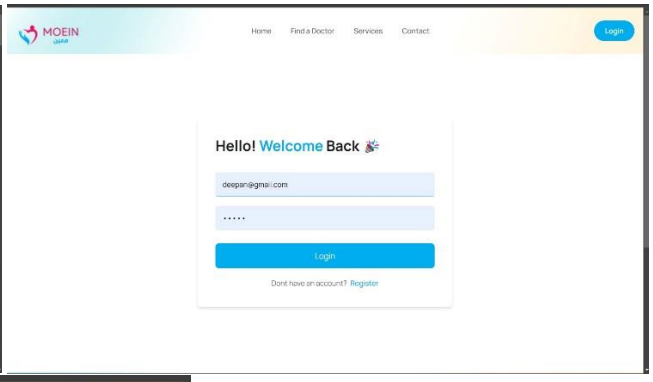
- The JWT is stored either in **localStorage** or **cookies** on the frontend to persist the user session.

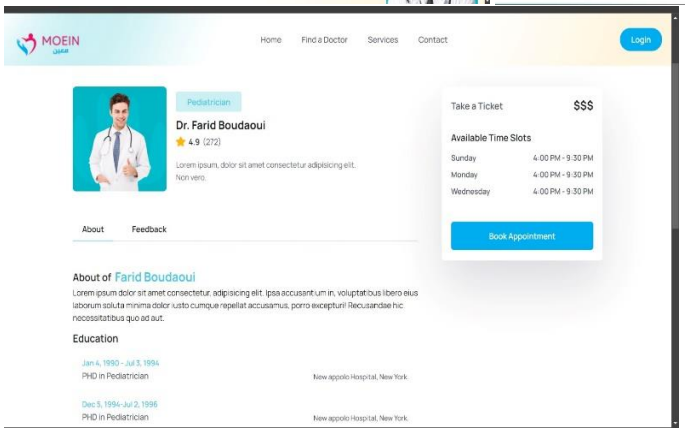
User Interface

The UI is designed to be clean, intuitive, and responsive. Key screens include:

- **Home Page**: A search interface for finding doctors based on specialty, location, and availability.
- **Doctor Profile**: Displays detailed information about the doctor, including their specialties, availability, and consultation options.
- **Booking Appointment**: Users can view available slots and book an appointment directly through an easy-to-use calendar.







Testing

Testing is a critical part of the development process. We used the following strategies:

- **Unit Testing**: Components and functions are tested using **Jest** and **React Testing Library** for the frontend, and **Mocha/Chai** for backend logic.
- **Integration Testing**: We use tools like **Postman** to test API endpoints and verify their functionality.

- **End-to-End Testing**: **Cypress** is used for simulating user behavior to test the full application flow from login to booking an appointment.

Screenshots or Demo

- **Demo Link**: https://drive.google.com/file/d/1WcZuNW6Tm8SZDMKa3aPrVJzryXOb7sj6/view?usp=drivesdk

Known Issues

- **Bug 1**: Users have reported occasional slow load times when fetching doctor availability. This will be investigated and optimized in a future update.
- **Bug 2**: Sometimes, the appointment cancellation status does not update immediately in the UI. This is being addressed with a fix.

Future Enhancements

- **Doctor Ratings**: Allow patients to rate and leave feedback on doctors after consultations