



**TO DEVELOP A PROJECT ON
BOOK A DOCTOR USING MERN STACK POWERED BY MANGODB**

A PROJECT REPORT

Submitted by

AAFRIN NISHA .M - 110521104001

ANANTHA RAJA .M - 110521104002

ANVAR THASEEM .A - 110521104003

ARAVINDHAN .R - 110521104004

In partial fulfilment for the award of the degree

Of

BACHELOR OF ENGINEERING

IN

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GOJAN SCHOOL OF BUSINESS AND TECHNOLOGY

ANNA UNIVERSITY, CHENNAI-600 025.

ABSTRACT

- ❖ The healthcare industry increasingly relies on digital solutions to enhance patient care and streamline administrative processes. A Doctor Booking System built using the MERN (MongoDB, Express.js, React, Node.js) stack is an efficient, scalable, and responsive platform that simplifies the process of scheduling medical appointments. This system enables patients to easily search for doctors based on specialty, location, and availability, while providing doctors with an intuitive interface to manage their appointments and availability.
- ❖ In this system, MongoDB serves as the NoSQL database, storing patient and doctor details, appointment records, and other relevant data. Express.js and Node.js handle server-side operations, ensuring a secure and fast backend environment. React.js, as the frontend framework, delivers a dynamic and user-friendly interface that facilitates real-time booking and notifications.
- ❖ The Doctor Booking System supports features like user authentication, appointment management, search filters, and real-time updates. It also integrates calendar management, allowing both patients and doctors to view and update their schedules in real time. The application aims to reduce the complexity of manual booking processes, improve accessibility to healthcare professionals, and enhance user satisfaction by providing a seamless digital experience.
- ❖ This platform represents a modern solution to the challenges faced in healthcare appointment scheduling, leveraging the MERN stack to provide a responsive, scalable, and maintainable solution suitable for both small practices and large medical institutions.

OVER VIEW

Purpose:

The purpose of the Doctor Booking System developed using the MERN (MongoDB, Express.js, React, Node.js) stack is to modernize and simplify the process of scheduling medical appointments for both patients and healthcare providers. Traditional appointment booking methods—whether over the phone or through paper-based systems—often lead to inefficiencies, scheduling conflicts, and limited access to healthcare services. This system seeks to overcome those challenges by providing a digital platform that is accessible, user-friendly, and efficient.

1. Convenience for Patients: Allow patients to easily search for doctors based on specialty, location, availability, and ratings. By offering a seamless online booking experience, the platform reduces the barriers to accessing healthcare services, especially for those in remote areas or with busy schedules.

2. Efficiency for Healthcare Providers: Give doctors and medical practitioners a centralized platform to manage their schedules, availability, and patient appointments. This reduces administrative overhead and scheduling conflicts, allowing healthcare professionals to focus more on patient care.

3. Real-Time Updates: Ensure that both patients and doctors can receive immediate updates on appointment confirmations, cancellations, and modifications. This real-time communication helps to minimize no-shows and improves operational efficiency.

4. Improved User Experience: Provide an intuitive and responsive interface that is easy to navigate for users of all technical skill levels. The React-based frontend ensures smooth interactions across different devices, while the backend infrastructure powered by Node.js and Express.js ensures stability and scalability.

5. Data Security: Safeguard sensitive patient information and appointment details through secure user authentication and encryption, aligning with healthcare privacy standards (such as HIPAA).

Features:

1. User Authentication and Role Management

Patient Registration and Login: Patients can create accounts, sign in, and securely manage their profiles, including personal details and appointment history.

Doctor Registration and Login: Doctors can register on the platform, create their profiles, and manage their availability and consultation preferences.

Admin Panel: Admins can manage user accounts (patients, doctors), oversee appointments, and monitor the system's operations.

2. Doctor Search and Filtering

Search by Specialty, Location, and Availability: Patients can search for doctors based on specific medical specialties (e.g., cardiology, dermatology), location (e.g., city, zip code), and available appointment slots.

Doctor Profiles: Detailed doctor profiles include qualifications, specialties, consultation fees, availability, and patient reviews and ratings.

Doctor Ratings and Reviews: Patients can leave reviews and rate doctors based on their experience, helping other users make informed decisions.

3. Appointment Scheduling and Management

Real-Time Booking: Patients can select an available doctor and book an appointment in real time, with immediate confirmation or notifications of any changes.

Appointment Calendar: Doctors can manage their availability through an interactive calendar, updating free time slots and adjusting their schedule as needed.

Appointment Reminders and Notifications: Both patients and doctors receive automated reminders and notifications via email/SMS or app alerts about upcoming appointments or any schedule changes.

Appointment History: Patients and doctors can access a history of past appointments, helping track consultation records.

4. Doctor Availability Management

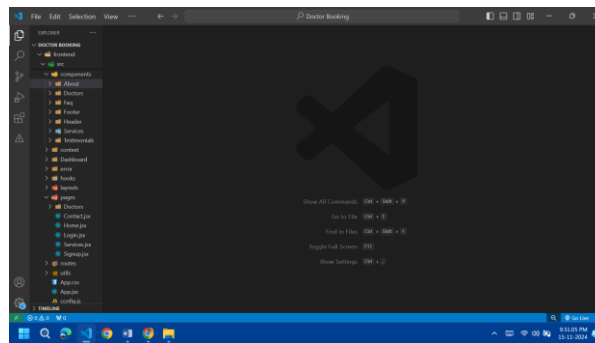
Availability Scheduling: Doctors can set their working hours, define their availability for appointments, and make adjustments for holidays, vacations, or other time off.

Dynamic Schedule Updates: Availability updates are reflected in real time, ensuring that patients can only book appointments during the doctor's available times.

ARCHITECTURE

FRONT-END:

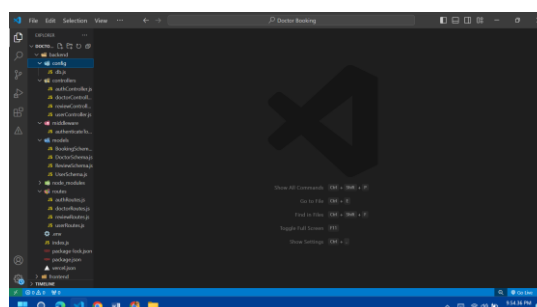
The frontend of the Doctor Booking System is built using **React.js**, a popular JavaScript library for building dynamic and interactive user interfaces. The frontend architecture is designed to be modular, responsive, and easy to maintain, while offering a smooth user experience for both patients and healthcare providers.



- ❖ **Header & Navbar:** A navigation bar containing links to various sections like home, doctor search, profile, etc.
- ❖ **Footer:** Contains links to privacy policy, terms of service, and contact information.
- ❖ **DoctorList:** Displays a list of doctors based on search criteria.
- ❖ **DoctorCard:** Displays individual doctor details (name, specialty, rating, availability).
- ❖ **AppointmentModal:** A modal dialog for confirming or managing appointments.
- ❖ **SearchBar:** A search input that allows patients to filter doctors by specialty, location, or availability.

BACK-END:

The backend of the **Doctor Booking System** is built using the **Node.js** runtime and **Express.js** framework, which serve as the foundation for handling API requests, business logic, and interactions with the database. The system utilizes **MongoDB** as its database for flexible, schema-less data storage. The backend architecture is designed to be scalable, secure, and efficient, providing an optimal environment for managing user authentication, appointment scheduling, and real-time updates.



Node.js

- **Runtime Environment:** Node.js is used for the backend runtime, enabling the server-side logic to be written in JavaScript. It is efficient and well-suited for handling numerous concurrent requests, which is critical in a real-time system like a Doctor Booking platform.

Express.js

- **Web Framework:** Express.js is a lightweight, flexible framework built on top of Node.js that simplifies routing, middleware integration, and request handling. It provides a set of tools to build RESTful APIs and manage HTTP requests with minimal boilerplate code.

MongoDB

- **NoSQL Database:** MongoDB is a NoSQL database, which allows the storage of structured and unstructured data (e.g., user details, appointment records, doctor profiles). Its flexibility and scalability make it ideal for applications that need to handle large volumes of data with varying structures.

Mongoose

- **ODM (Object Data Modeling):** Mongoose is used to interact with MongoDB, providing a schema-based solution to model application data and perform database operations with built-in validation, queries, and middleware support.

DATABASE:

The Doctor Booking System uses **MongoDB** for scalable and flexible data storage. Below is a concise outline of the key collections and their structure:

1. Users Collection

Stores user information for both patients and doctors.

Fields:

``_id``: ObjectId (auto-generated by MongoDB)

``role``: String (either "patient" or "doctor")

``name``: String

``email``: String (unique)

``passwordHash``: String (hashed password)

`phone`: String
`profilePicture`: String (URL or path to image)
`createdAt`: Date
`updatedAt`: Date

2. Doctors Collection:

Stores doctor-specific details such as specialty, availability, and consultation fee.

Fields:

`_id`: ObjectId (references a user)
`specialty`: String
`consultationFee`: Number
`availableSlots`: Array of objects (each object stores day and time slots)
`rating`: Number
`experienceYears`: Number
`bio`: String
`languagesSpoken`: Array of strings
`createdAt`: Date
`updatedAt`: Date

3. Appointments Collection:

Stores information about patient appointments with doctors.

Fields:

`_id`: ObjectId
`patientId`: ObjectId (references a user who is a patient)
`doctorId`: ObjectId (references a user who is a doctor)
`appointmentDate`: Date
`status`: String (e.g., "booked", "completed", "cancelled")
`createdAt`: Date
`updatedAt`: Date

4. Reviews Collection:

Stores patient reviews for doctors after appointments.

Fields:

``_id``: ObjectId
``doctorId``: ObjectId (references a doctor)
``patientId``: ObjectId (references a patient)
``rating``: Number (1 to 5)
``comment``: String
``createdAt``: Date

5. Availability Collection

Stores the available time slots for doctors to manage appointments.

Fields:

``_id``: ObjectId
``doctorId``: ObjectId (references a doctor)
``availableDates``: Array of objects (each object stores a date with time slots)
``createdAt``: Date
``updatedAt``: Date

Relationships:

Users: Can be patients or doctors based on the ``role`` field.

Appointments: Linked to both ``patients`` and ``doctors`` through ``patientId`` and ``doctorId``.

Reviews: Linked to both ``patients`` and ``doctors`` via ``patientId`` and ``doctorId``.

Availability: Stores doctor availability for scheduling purposes, linked to ``doctorId``.

This structure ensures efficient management of user data, doctor schedules, appointments, and reviews within the MongoDB database.

SETUP INSTRUCTIONS & RUNNING THE APPLICATION:

Prerequisites

Ensure you have the following installed on your machine:

1. **Node.js** (with npm)
 - Download and install [Node.js](#).
 - Ensure both `node` and `npm` are installed:


```
Copy code
node -v
npm -v
```

2. **MongoDB** (Local or Cloud)
 - **Local MongoDB:** Install [MongoDB](#) on your machine and run it locally.
 - **Cloud MongoDB:** Create a MongoDB Atlas account at [MongoDB Atlas](#) and create a cluster to get a connection string.
3. **Git** (Optional, for version control)
 - Install [Git](#) to clone repositories and manage versions.
 - Verify with `git --version`.
4. **Text Editor** (Recommended: Visual Studio Code)
 - Download and install [Visual Studio Code](#).

Installation Steps

1. Clone the Repository

If the project is hosted on GitHub (or other Git repositories), clone it to your local machine:

```
git clone https://github.com/your-repo/doctor-booking-system.git
cd doctor-booking-system
```

2. Backend Setup (Node.js and Express)

1. **Navigate to the Backend Directory** (if the project is separated into frontend and backend directories):

```
cd backend
```

2. **Install Backend Dependencies:** Install required Node.js packages using npm:

```
npm install
```

3. **Create .env File for Environment Variables:** Create a `.env` file in the root of the backend folder to store sensitive information like the MongoDB URI and JWT secret.

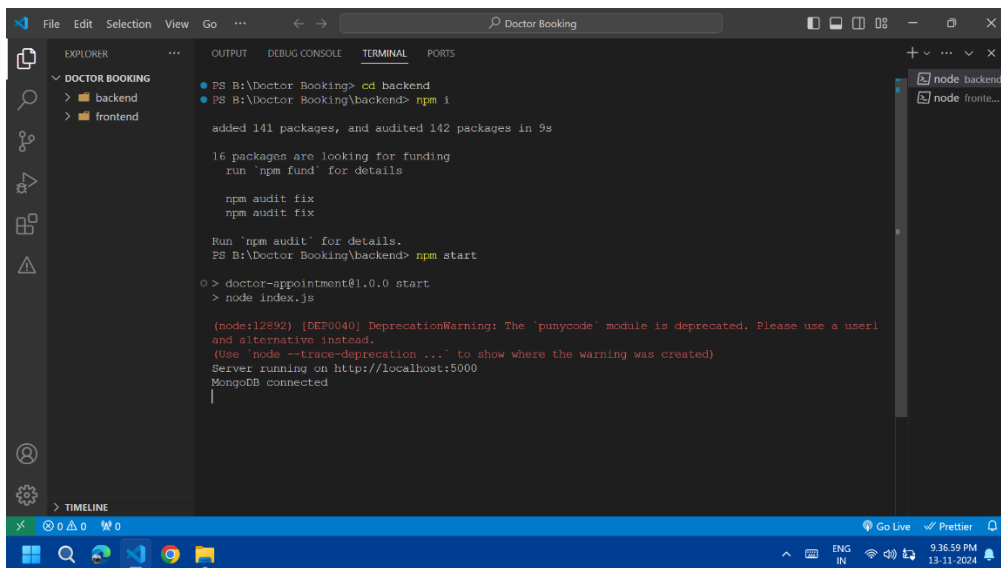
Example `.env`:

```
MONGO_URI=mongodb://localhost:27017/doctor_booking
JWT_SECRET=your_jwt_secret_key
PORT=5000
```

- If using MongoDB Atlas, replace the `MONGO_URI` with your connection string from MongoDB Atlas.
4. **Start the Backend Server:** Start the backend server using the following command:

```
npm start
```

- The server should now be running on <http://localhost:5000>.



3. Frontend Setup (React.js)

1. Navigate to the Frontend Directory:

```
cd frontend
```

2. Install Frontend Dependencies: Install React.js dependencies using npm:

```
npm install
```

3. Set up Environment Variables (for frontend API calls): If required, create a `.env` file in the `frontend` directory to store the backend API URL or other settings.

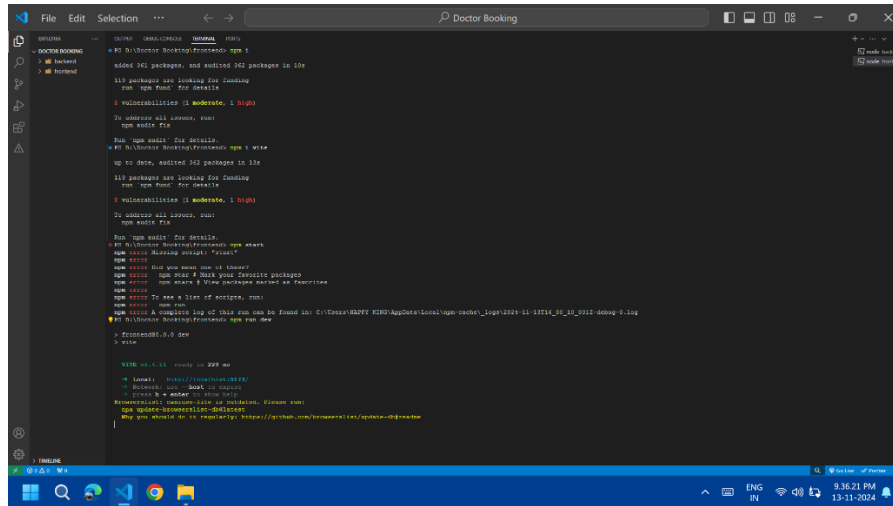
Example `.env`:

```
REACT_APP_API_URL=http://localhost:5000/api
```

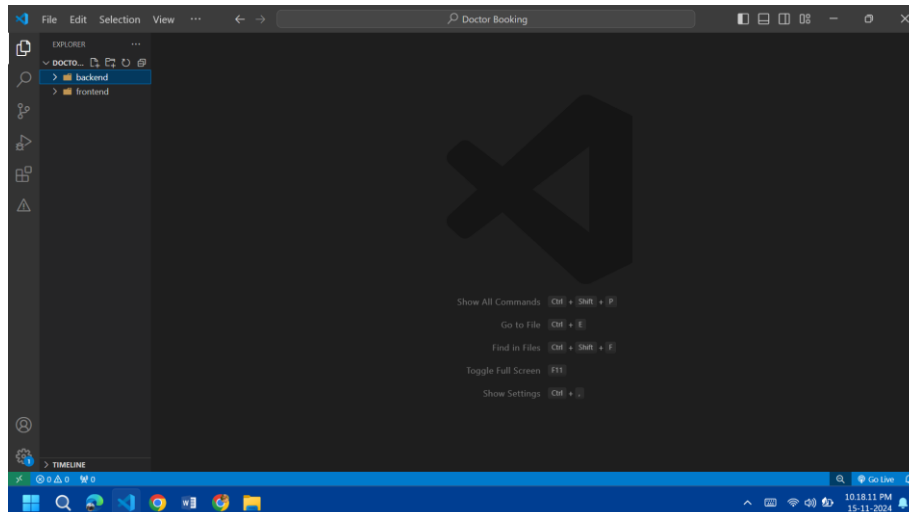
4. Start the Frontend Development Server: Run the React development server using:

```
npm start
```

- The frontend should now be running on <http://localhost:3000>.



FOLDER STRUCTURE : (Frontend & Backend)



API DOCUMENTATION

Creating an API documentation for booking a doctor using the MERN (MongoDB, Express.js, React.js, and Node.js) stack involves several key endpoints to handle features like doctor listing, booking appointments, and user management. Here's a basic structure for the API documentation:

Doctor Booking API:

Base URL:

<http://<your-domain>/api>

Authentication

1. User Login

Endpoint: /auth/login

Method: POST

Description: Authenticate a user and return a JWT token.

Request Body:

json

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

Response:

json

```
{  
  "success": true,  
  "token": "JWT_TOKEN",  
  "user": {  
    "id": "user123",  
    "name": "John Doe"  
  }  
}
```

2. User Registration

Endpoint: /auth/register

Method: POST

Description: Register a new user.

Request Body:

json

```
{  
  "name": "John Doe",  
  "email": "user@example.com",  
  "password": "password123"  
}
```

Response:

json

```
{  
  "success": true,  
  "message": "User registered successfully"  
}
```

Doctors

1. Get All Doctors

Endpoint: /doctors

Method: GET

Description: Retrieve a list of all available doctors.

Response:

json

```
{  
  "success": true,  
  "doctors": [  
    {  
      "id": "doc123",  
      "name": "Dr. Smith",  
      "specialization": "Cardiology",  
    }  
  ]  
}
```

```
    "availableSlots": ["2024-11-18T10:00:00", "2024-11-18T11:00:00"]
  }
]
}
```

2. Get Doctor by ID

Endpoint: /doctors/:id

Method: GET

Description: Retrieve details for a specific doctor.

Response:

json

```
{
  "success": true,
  "doctor": {
    "id": "doc123",
    "name": "Dr. Smith",
    "specialization": "Cardiology",
    "availableSlots": ["2024-11-18T10:00:00", "2024-11-18T11:00:00"]
  }
}
```

Bookings

1. Book an Appointment

Endpoint: /appointments

Method: POST

Description: Book an appointment with a doctor.

Request Body:

json

```
{
  "userId": "user123",
  "doctorId": "doc123",
  "appointmentDate": "2024-11-18T10:00:00"
}
```

Response:

json

```
{
  "success": true,
  "message": "Appointment booked successfully",
  "appointment": {
    "id": "appointment123",
    "userId": "user123",
    "doctorId": "doc123",
    "appointmentDate": "2024-11-18T10:00:00"
  }
}
```

2. Get Appointments

Endpoint: /appointments

Method: GET

Description: Retrieve all appointments for the logged-in user.

Headers:

Authorization: Bearer JWT_TOKEN

Response:

json

```
{
  "success": true,
  "appointments": [
    {
      "id": "appointment123",
      "doctorId": "doc123",
      "doctorName": "Dr. Smith",
      "appointmentDate": "2024-11-18T10:00:00"
    }
  ]
}
```

3. Cancel Appointment

Endpoint: /appointments/:id

Method: DELETE

Description: Cancel a specific appointment.

Response:

json

```
{
  "success": true,
  "message": "Appointment canceled successfully"
}
```


Middleware

JWT Authentication

Protect routes to ensure only authenticated users can access them.

Error Handling

Responses should include an appropriate status code and error message:

- 400: Bad Request
- 401: Unauthorized
- 404: Not Found
- 500: Internal Server Error

USER INTERFACE

Designing the *user interface (UI)* for a doctor booking system involves creating a seamless, intuitive experience for users to interact with features like registration, login, viewing doctors, and booking appointments. Here's an outline of the key components and design considerations:

1. UI Design Overview

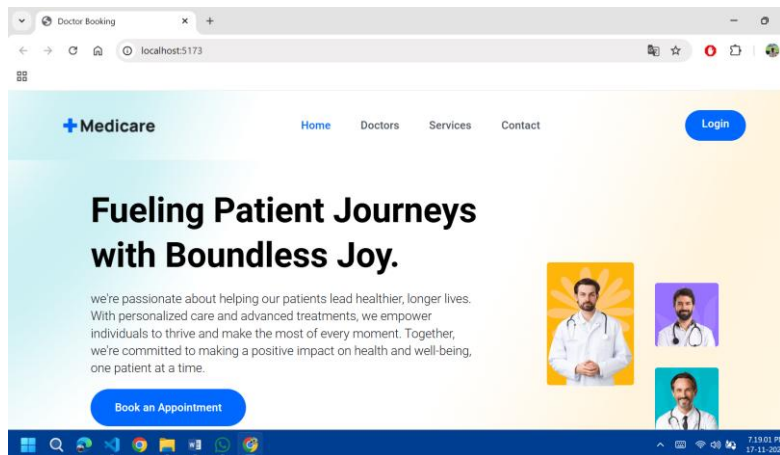
The UI will have three primary roles:

- Patients: Register, login, view doctors, and book appointments.
- Doctors: Manage availability and view appointments.
- Admin (optional): Manage users and doctors.

2. Core Pages for Patients

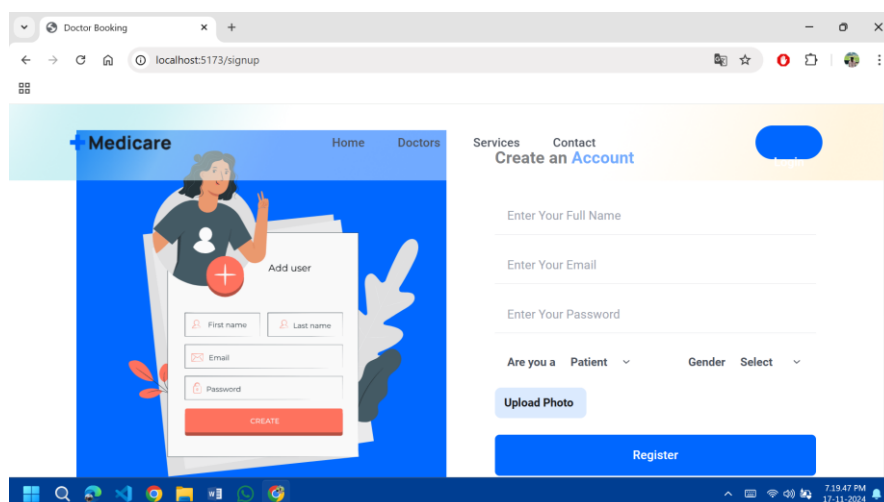
A. Home Page

- Purpose: Welcome users and provide quick navigation options.
- Features:
 - Navigation menu (e.g., Home, Doctors, Appointments, Profile, Logout).
 - Call-to-action buttons: "Book a Doctor" or "Sign In/Register."



B. Registration Page

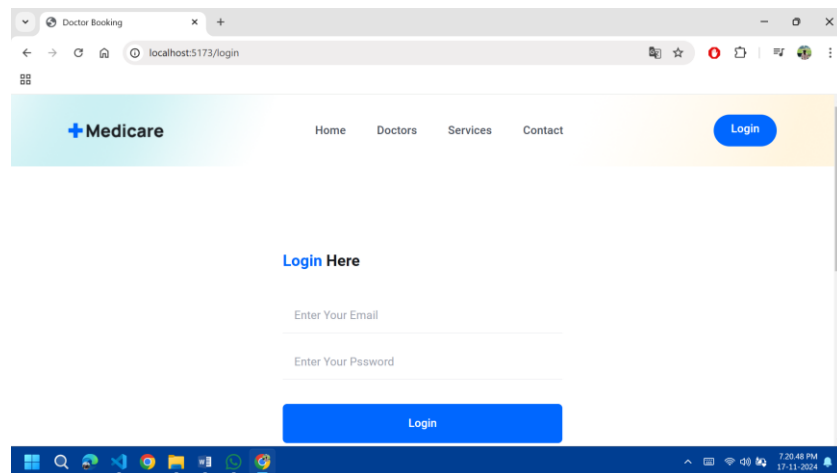
- Purpose: Allow new users to sign up.
- Elements:
 - Form with fields:
 - Full Name
 - Email
 - Password (with password strength indicator)
 - Confirm Password
 - "Register" button.
 - Link to "Already have an account? Login."



C. Login Page

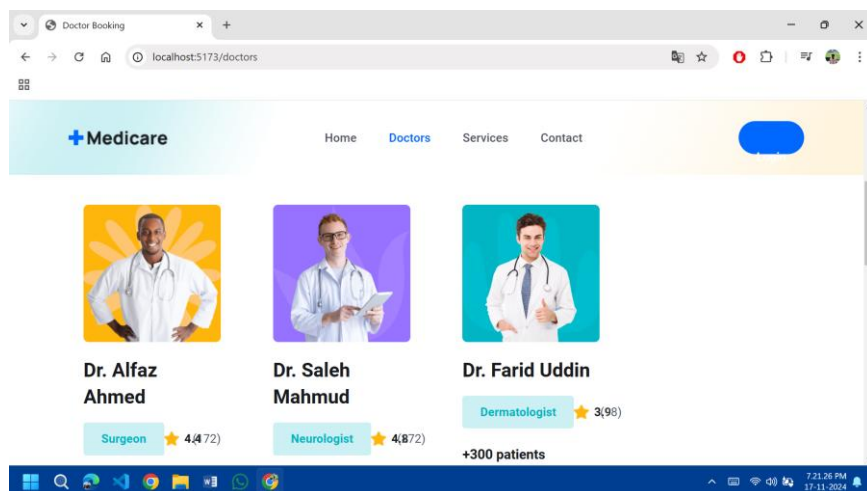
- Purpose: Authenticate existing users.
- Elements:

- Email and Password fields.
- "Login" button.
- Link to "Forgot Password?" and "New here? Register."



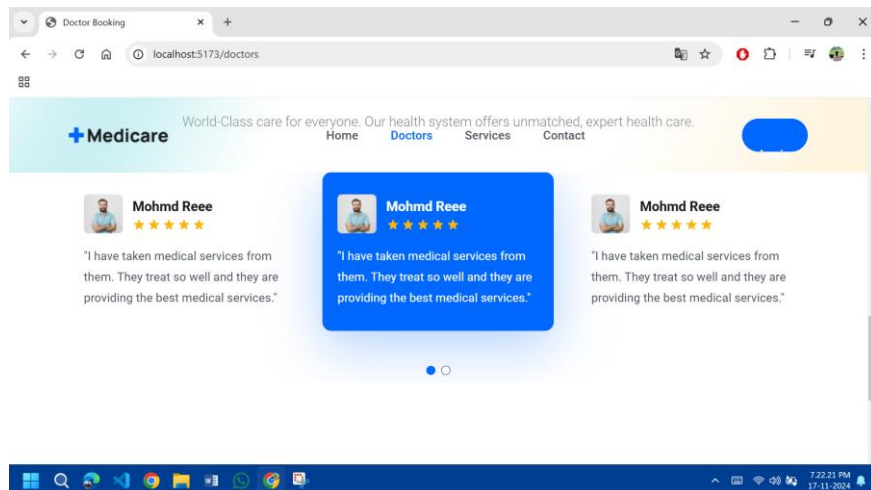
D. Doctors Listing Page

- Purpose: Display a list of available doctors for booking.
- Features:
 - Search bar to find doctors by name, specialization, or location.
 - Filters for specialization, availability, and ratings.
 - Card-based layout for each doctor:
 - Doctor's photo, name, specialization, and rating.
 - "View Details" and "Book Appointment" buttons.



E. Doctor Details Page

- Purpose: Provide detailed information about a specific doctor.
- Features:
 - Doctor's profile:
 - Name, photo, qualifications, specialization, experience.
 - Ratings and reviews.
 - Available time slots (selectable).
 - "Book Now" button.



F. Appointment Booking Page

- Purpose: Allow users to book an appointment with a doctor.
- Features:
 - Selected doctor's details.
 - Date picker to select a preferred date.
 - Time slots (based on the doctor's availability).
 - Confirmation button.

G. Appointment History Page

- Purpose: Show upcoming and past appointments.
- Features:
 - List of booked appointments (upcoming at the top).
 - Details: Doctor's name, date, time, and status (confirmed, canceled).
 - Cancel option for upcoming appointments.

H. User Profile Page

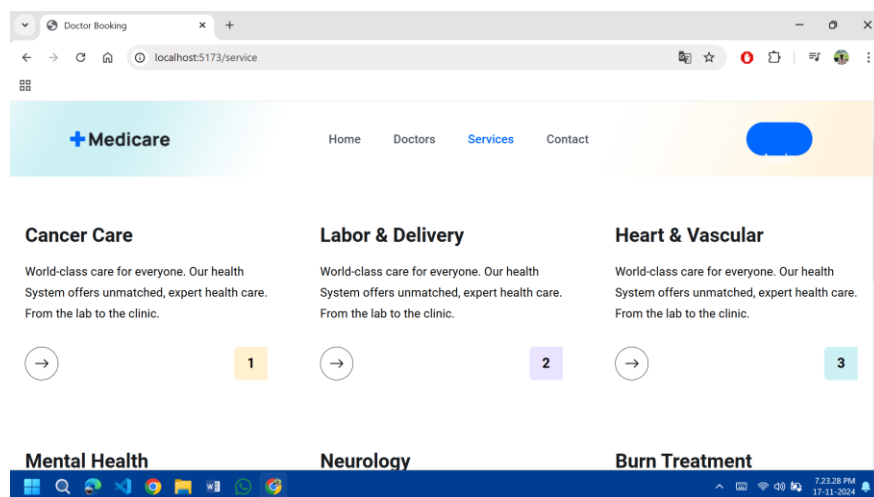
- Purpose: Allow users to manage their account.
- Features:
 - Display personal information (name, email).
 - Edit button to update details (except email).
 - Change password option.

3. Core Pages for Doctors

- Doctor Dashboard: View booked appointments.
- Availability Management: Set available slots for booking.
- Profile Management: Update personal and professional details.

4. Core Pages for Admin

- Admin Dashboard: High-level stats on users, doctors, and appointments.
- User Management: Add, remove, or block users.
- Doctor Management: Add or update doctor profiles.



5. UI Design Best Practices

1. Responsive Design

- Ensure the app works seamlessly on mobile, tablet, and desktop devices.

2. Clear Navigation

- Use a navigation bar or side menu for easy access to major sections.

3. Consistent Design

- Maintain consistency in fonts, colors, and button styles.

4. Feedback and Confirmation

- Show success/error messages after actions like booking or login.
- Use modals for confirmation (e.g., "Are you sure you want to cancel this appointment?").

5. Accessibility

- Use proper labels, high contrast, and keyboard-friendly navigation.

6. Tools for UI Design


- Figma or Adobe XD: For creating wireframes and prototypes.
- React.js Frameworks:
 - Material-UI: Prebuilt components for a modern design.
 - Ant Design: Comprehensive design system with professional-grade components.
 - Bootstrap: For responsive layouts and styling.

7. Example Navigation Flow

1. Home Page
2. Login/Register
3. Doctors List
4. Doctor Details
5. Book Appointment
6. Confirmation and History

TESTING

- **Test Case ID:** Unique identifier for each test case.
- **Feature:** The module or functionality being tested (e.g., Authentication, Appointment Booking).
- **Test Scenario:** High-level description of the test objective.
- **Test Steps:** Step-by-step actions to perform the test.
- **Test Data:** Sample input data used for the test.
- **Expected Result:** The anticipated system behavior or output.

Test Case ID	Feature	Test Scenario	Test Steps	Test Data	Expected Result
TC001	User Authentication	Login with valid credentials	1. Enter valid email and password. 2. Click "Login".	Email: <code>user@example.com</code> Password: <code>password123</code>	User is logged in, and JWT token is returned.
TC002	Doctor Management	Fetch all doctors	1. Send a GET request to <code>/doctors</code> .	N/A	List of all doctors is returned in JSON format.
TC003	Appointment Booking	Book an appointment with valid details	1. Select doctor, date, and time. 2. Click "Book".	Doctor ID: <code>doc001</code> Date: <code>2024-11-18</code> Time: <code>10:00 AM</code>	Appointment is successfully booked.
TC004	Frontend - Appointment	Cancel an appointment	1. Navigate to "My Appointments". 2. Click "Cancel" for a specific appointment. 	Appointment ID: <code>appt123</code>	Appointment is canceled, and confirmation is shown.

-THANK YOU-