**BOOK A DOCTOR USING MERN**

# PROJECT REPORT

**Submitted by**

**NAME: ANUJA.M REG NO: 312621205004**

**NM\_ID:** **3D08333DB70EF9FAC3345EEE4C14BF38**

**In partial fulfilment for the award of the degree of**

**BACHELOR OF TECHNOLOGY**

**IN**

**INFORMATION TECHNOLOGY**



**THANGAVELU ENGINEERING COLLEGE**

**KARAPAKKAM, CHENNAI - 600097**

**ANNA UNIVERSITY**

**CHENNAI - 600025**

OCTOBER 2023

|  |  |
| --- | --- |
| S.NO | TABLE OF CONTENTS |
| 1 | Introduction |
| 2 | Project Objectives |
| 3 | Technology Stack |
| 4 | System Architecture |
| 5 | Backend Development |
| 6 | Frontend Development |
| 7 | Authentication and Authorization |
| 8 | Error Handling and Logging |
| 9 | User Interface Design |
| 10 | Conclusion and References |

INTRODUCTION

The **Book a Doctor** web application is an innovative and user-friendly platform designed to simplify the process of booking medical appointments. Built using the **MERN stack** (MongoDB, Express.js, React, Node.js), the application provides a seamless interface for users to find doctors based on their specialties, book appointments, and manage medical schedules. It also includes dashboards for doctors and administrators, ensuring efficient management of appointments and doctor information.

**Purpose**

In today’s fast-paced world, access to healthcare is crucial, and the traditional method of scheduling appointments by phone or in-person can often be time-consuming and inefficient. The **Book a Doctor** system aims to address this challenge by offering an online platform where users can quickly and easily search for doctors, check their availability, and schedule appointments at their convenience. Similarly, doctors can manage appointments, view patient details, and update appointment statuses through their dedicated dashboard.

**Scope of User Roles**

* **Users** can register, log in, view a list of available doctors, book appointments, and track their appointments.
* **Doctors** can manage and view appointments, accept or decline patient requests, and reschedule appointments when necessary**.**
* **Admins** have full control over the platform, with the ability to manage doctor profiles, view and delete appointments, and ensure the system operates smoothly**.**

**Technologies Used**

* **MongoDB:** NoSQL database for storing user, doctor, and appointment data.
* **Express.js:** Web application framework for building the backend API and handlingHTTP requests.
* **React.js**: JavaScript library for building the user interface with dynamic, reusable components.
* **Node.js**: JavaScript runtime for building the server-side logic of the application.

PRIMARY OBJECTIVES

1. **Simplify Appointment Booking**:

Provide a user-friendly platform for individuals to search for doctors, view available time slots, and book appointments online, making healthcare more accessible and efficient.

1. **Doctor Appointment Management**:

Enable doctors to manage their appointments effectively, including accepting, declining, and rescheduling appointments, while maintaining a clear overview of their schedule.

1. **Admin Control and Management**:

Allow administrators to manage the entire system by adding or removing doctors, editing doctor profiles, and overseeing user and appointment data, ensuring smooth operation of the platform.

1. **Role-Based Access Control**:

Implement role-based access for users, doctors, and admins to ensure appropriate functionality and security for each user type.

1. **Scalability and Flexibility**:

Build the application on a scalable architecture using the MERN stack (MongoDB, Express, React, Node.js) to accommodate future growth, additional features, and large numbers of users and appointments.

1. **User Authentication and Security**:

Implement secure authentication (via JWT) to protect user data and prevent unauthorized access to sensitive information.

1. **Responsive and Intuitive UI**:

Create an intuitive, responsive, and mobile-friendly user interface to ensure a seamless experience across devices.

TECHNOLOGY STACK

|  |  |
| --- | --- |
| Category | Technology |
| Frontend | React.js, React Router, Axios, Bootstrap/CSS3 |
| Backend | Node.js, Express.js, JWT Authentication |
| Database | MongoDB, Mongoose |
| Deployment & Hosting | MongoDB Atlas |
| Version Control | Git, GitHub |
| Security | Bcrypt.js, CORS |
| Testing | Postman |

**Frontend Dependencies:**

* **react**: Core library for building the user interface.
* **react-router-dom**: For routing and navigation between pages.
* **axios**: For making HTTP requests to interact with the backend API.

**Backend Dependencies:**

* **express**: Web framework for Node.js to handle routing and API requests.
* **jsonwebtoken (JWT)**: For generating and verifying JSON Web Tokens for user authentication.
* **bcryptjs**: For securely hashing and comparing user passwords.
* **mongoose**: ODM library for interacting with MongoDB and managing data models and validation.

**Development Tools:**

* **nodemon**: Automatically restarts the server during development when code changes are made.
* **dotenv**: For managing environment variables (e.g., API keys, database credentials).
* **postman**: For testing API endpoints during development.

SYSTEM ARCHITECTURE





BACKEND DEVELOPMENT

1. **Authentication & Authorization**:

* **JWT (JSON Web Token)** is used for **user authentication**.
* Passwords are securely hashed using **bcryptjs**.
* Role-based access control ensures that only authorized users (Admin, Doctor, User) can access specific routes.

1. **Models**:

* **User Model**: Stores user details such as name, email, password, and role (user, doctor, admin).
* **Doctor Model**: Stores doctor information, including name, specialty, and available days.
* **Appointment Model**: Stores appointment details, including patient name, doctor, date, and status (pending, confirmed, canceled).

1. **Middleware**:

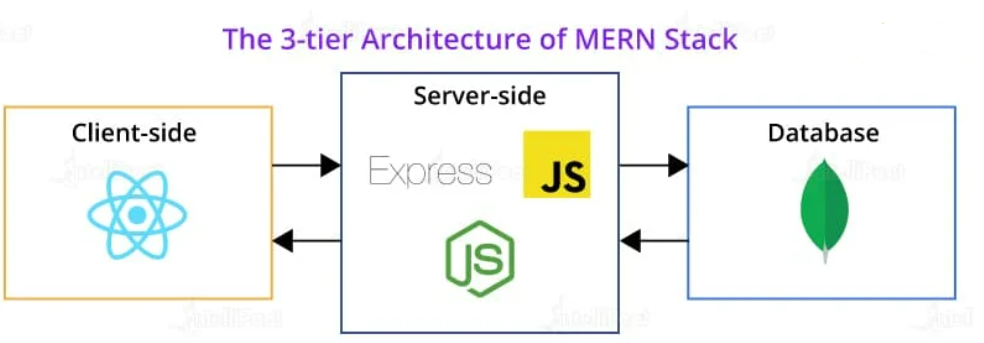
* **auth.js**: Verifies JWT and authenticates requests.
* **checkAdmin.js**: Restricts access to admin-specific routes.
* **checkDoctor.js**: Restricts access to doctor-specific routes.

1. **Routes**:

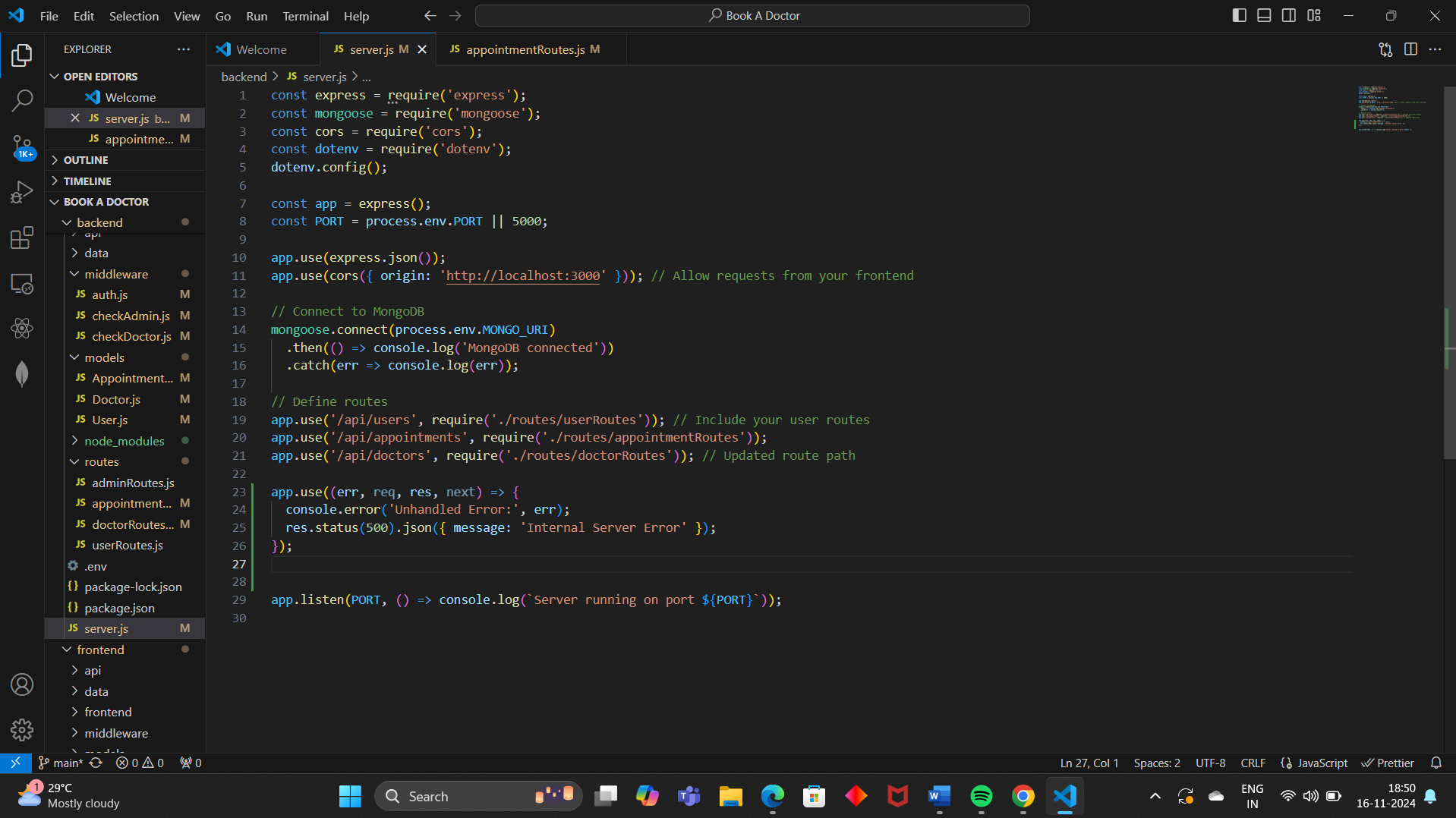
* **User Routes**: For registration, login, and JWT token generation.
* **Doctor Routes**: For viewing and adding doctors (admin access).
* **Appointment Routes**: For booking, accepting, or rescheduling appointments.
* **Admin Routes**: For managing doctor and patient data.

1. **Security:**

* **JWT Authentication** protects sensitive routes.
* **Role-based access control** ensures users can only access routes relevant to their role (admin, doctor, user).
* **Password hashing** with **bcryptjs** ensures user passwords are stored securely.



Server.js



1. **Dependencies**:

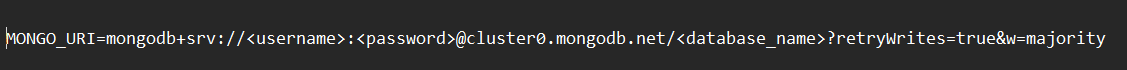
* **express:** Used to create the server and handle HTTP requests.
* **mongoose:** Used for interacting with the **MongoDB** database.
* **cors:** Middleware to allow cross-origin requests, enabling the frontend to communicate with the backend.
* **dotenv:** Loads environment variables from a .env file (e.g., database URL, port).

1. **Middleware**:

* **express.json():** Parses incoming requests with JSON payloads.
* **cors():** Allows the frontend (running on port 3000) to make requests to the backend.

1. **MongoDB Connection**:

* The application connects to MongoDB using the URI stored in an environment variable **(MONGO\_URI).** If successful, it logs a connection message; otherwise, it catches and logs any errors.



1. **Routes**:

* /api/users: Manages user-related routes (e.g., registration, login).
* /api/appointments: Manages appointment-related routes.
* /api/doctors: Manages doctor-related routes.

1. **Error Handling**:

* A global error handler catches unhandled errors and responds with a 500 status code and a generic error message.

1. **Server**:

* The application listens on the specified port (5000 by default) and logs a success message when the server is running.

FRONTEND DEVELOPMENT

**Technologies Used**

* **React.js**: A JavaScript library for building dynamic user interfaces with a component-based structure.
* **React Router**: A library for routing and navigation within the React application.
* **Axios**: A promise-based HTTP client for making requests to the backend API.
* **CSS**: For styling the application and creating a responsive design.
* **React Hooks**: For managing state and lifecycle methods in function components.
* **JWT**: For storing and managing user authentication tokens on the client-side.

**Key Features Implemented**

1. **Landing Page**:
   * Displays options to **Register** or **Login**.
   * User can click on the respective button to navigate to the corresponding page.
2. **Registration and Login**:
   * **User Registration**: Collects user details like name, email, password, and role (Admin, Doctor, User) and communicates with the backend API to register the user.
   * **User Login**: Users log in by providing their credentials, and if successful, a JWT token is stored for authenticated requests.
3. **Dashboard**:
   * **User Dashboard**: Displays a list of doctors and their specialties. Users can select a doctor, choose an available date/time, and book an appointment.
   * **Doctor Dashboard**: Shows the list of patients who have booked appointments. Doctors can accept, decline, or reschedule appointments.
   * **Admin Dashboard**: Allows admins to view and manage the list of doctors, appointments, and other users.

4. **Role-Based Access**:

* Users, doctors, and admins see different content based on their role. For example, only admins can add or delete doctors, while only doctors can manage appointments.

5. **Appointment Management**:

* **User** can book, view, and cancel appointments.
* **Doctor** can accept, decline, or reschedule appointments.
* **Admin** can view and delete appointments.

1. **User Registration**

The **User Registration** process allows a user to sign up and create an account. Once the registration is successful, the user can log in and access the **User Dashboard**.

**Core Steps:**

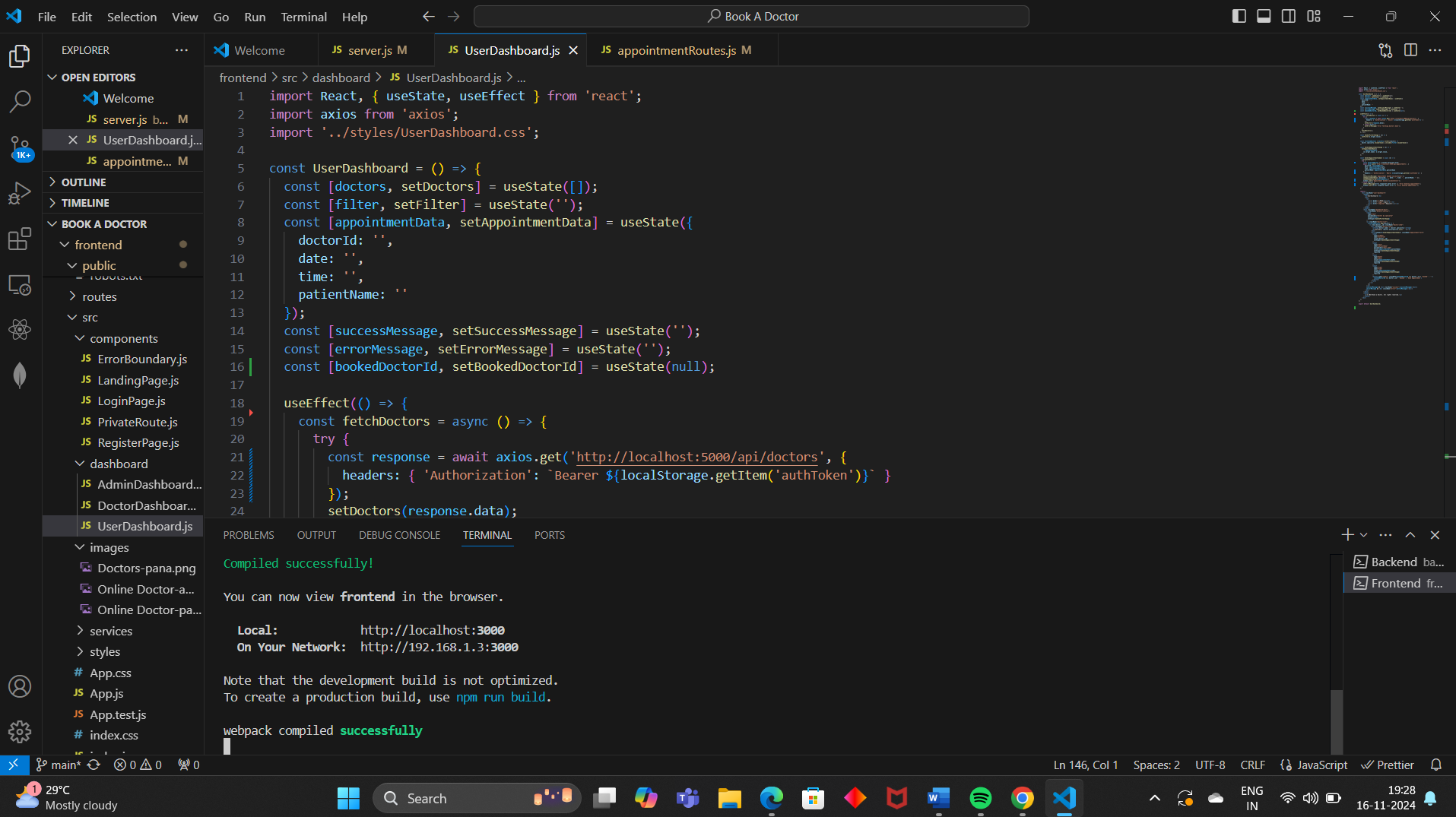
1. **Input Fields**:
   * **Name**: Full name of the user.
   * **Email**: Unique email address.
   * **Password**: Strong password.
   * **Role**: By default, users are assigned the role of a **user** (can be updated to doctor or admin later if needed).
2. **Validation**:
   * Ensure that all required fields are filled.
   * Validate that the email is not already taken by another user.
3. **JWT Token Generation**:
   * After successful registration, a **JWT** token is generated, which is used for authenticating the user in subsequent requests.
4. **Response**:
   * After successful registration, the server returns a success message, the **JWT token**, and basic user details like name, email, and role.

**B. User Login**

The **User Login** process allows registered users to log in with their email and password. Upon successful authentication, a **JWT token** is returned, which grants access to protected routes (such as the User Dashboard).

**Core Steps:**

1. **Input Fields**:
   * **Email**: The email used during registration.
   * **Password**: The password entered by the user during registration.
2. **Validation**:
   * Check that the email exists in the database.
   * Validate the password using **bcrypt** to compare the entered password with the hashed password in the database.
3. **JWT Token**:
   * On successful login, a **JWT** token is generated and sent back to the frontend. This token is used for authorizing further requests.
4. **Response**:
   * The server returns the **JWT token** and user details (name, email, role) in the response.



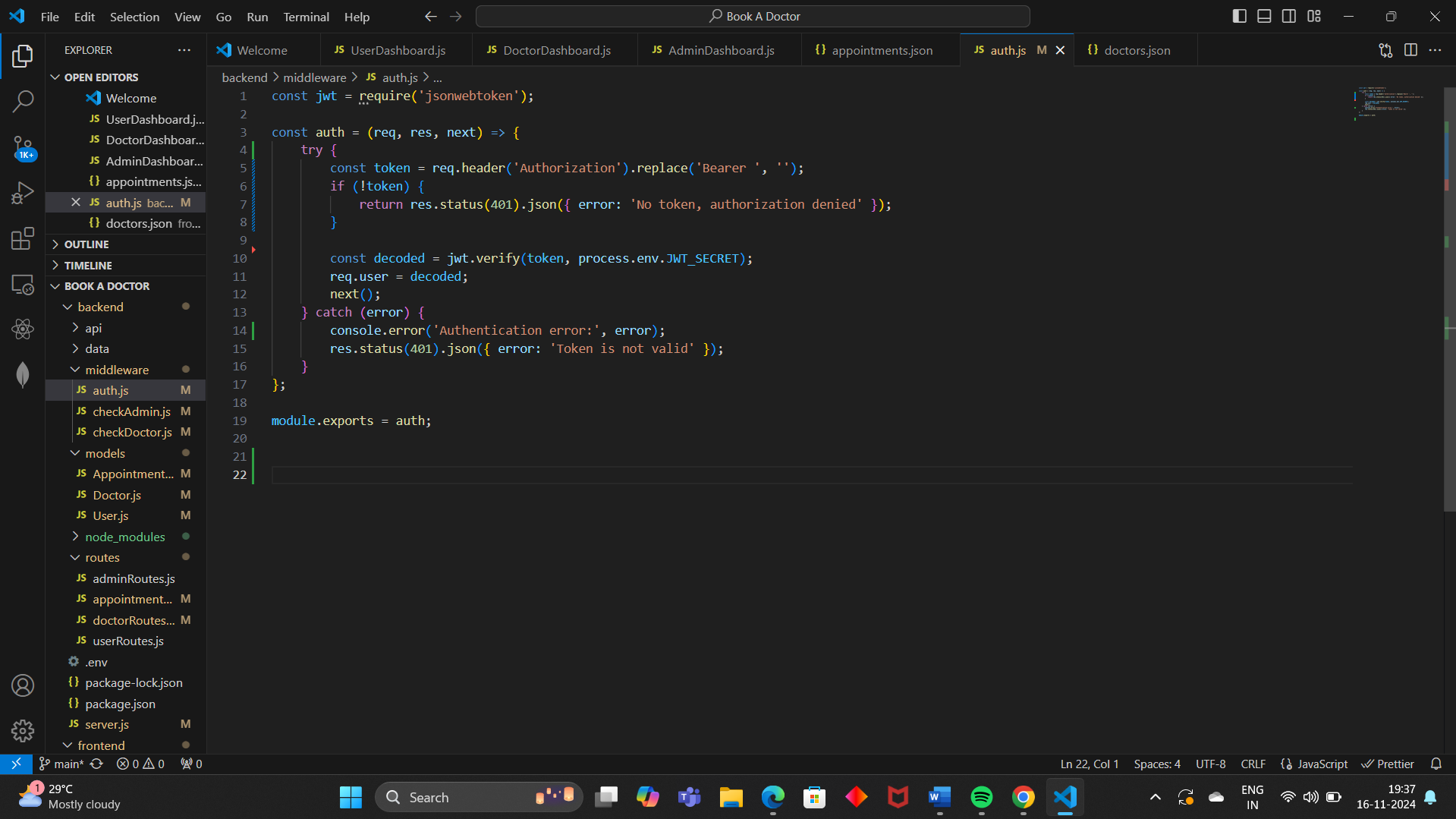
AUTHENTICATION AND AUTHORIZATION

**1. Authentication**

**Authentication** is the process of verifying the identity of a user. In this application, users (patients, doctors, and admins) need to log in with valid credentials (email and password) to prove their identity.

**Steps Involved in Authentication:**

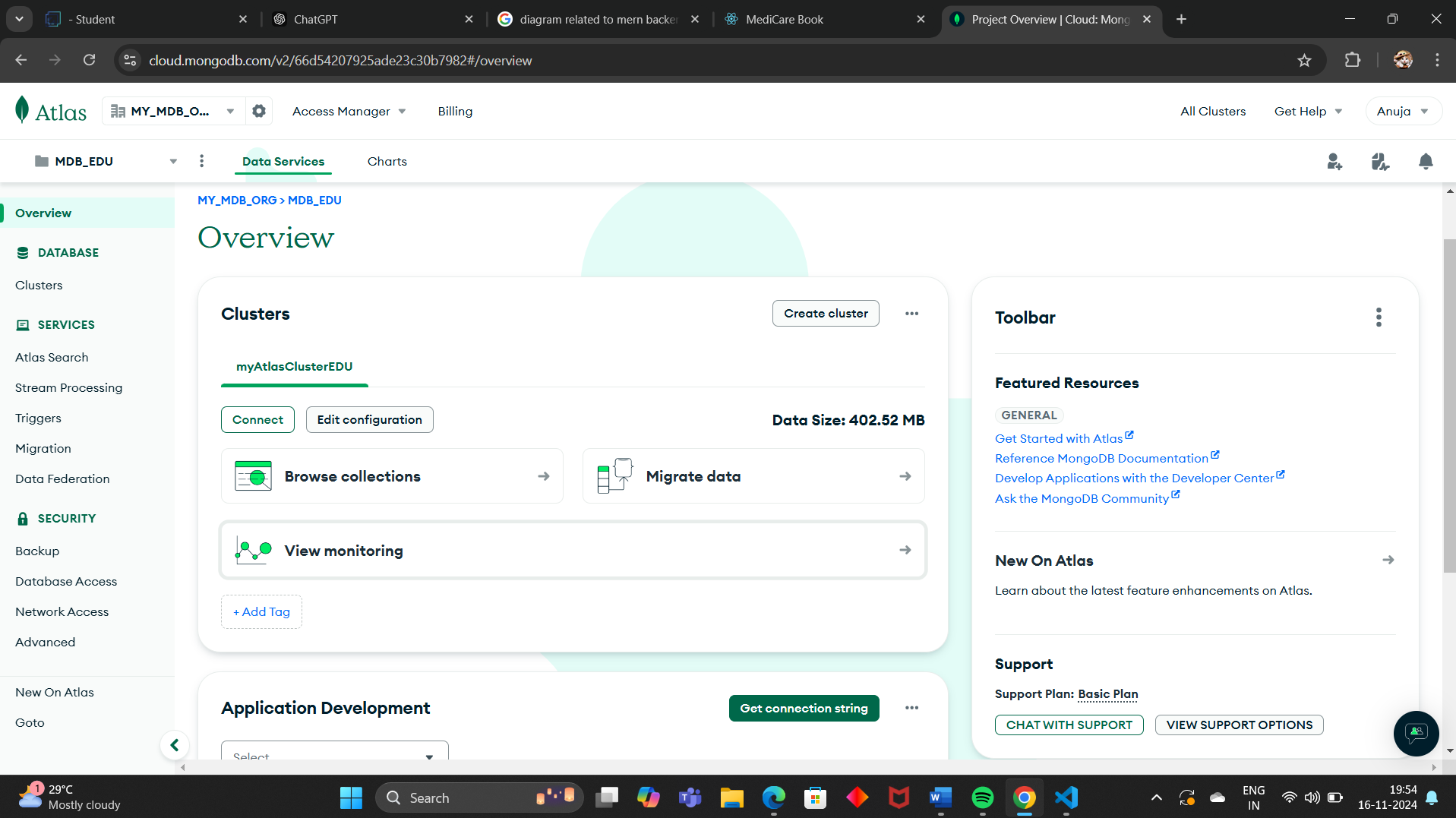
1. **User Registration**:
   * The user provides their information (name, email, password).
   * The password is hashed using a secure hashing algorithm (e.g., bcrypt).
   * A **JWT token** is generated and sent to the user after successful registration.
   * The token is used for all future requests to verify the user’s identity.
2. **User Login**:
   * The user enters their **email** and **password**.
   * The email is checked against the database to find a matching user.
   * The JWT token is then used to authenticate subsequent requests made by the user.
3. **JWT Token**:
   * The **JWT token** is a string that encodes the user’s information (e.g., user ID, role) and has an **expiration time**.
   * The token is included in the **Authorization** header of every request made to protected routes.



**MONGODB ATLAS DATABASE**

MongoDB Atlas is a Database-as-a-Service (DBaaS) that allows users to set up and manage MongoDB databases in the cloud without the need for physical hardware or complex manual setups. It provides several features like:

* **Scalability**: Atlas automatically scales your MongoDB database to handle growing data requirements.
* **Security**: It offers robust security features like encryption, authentication, and network isolation.
* **Backup and Recovery**: Atlas provides automated backups and recovery options.
* **Monitoring and Alerts**: Atlas offers built-in monitoring and alerting features to track database performance and health.
* **Global Clusters**: Atlas supports multi-region clusters to improve data locality and reduce latency.

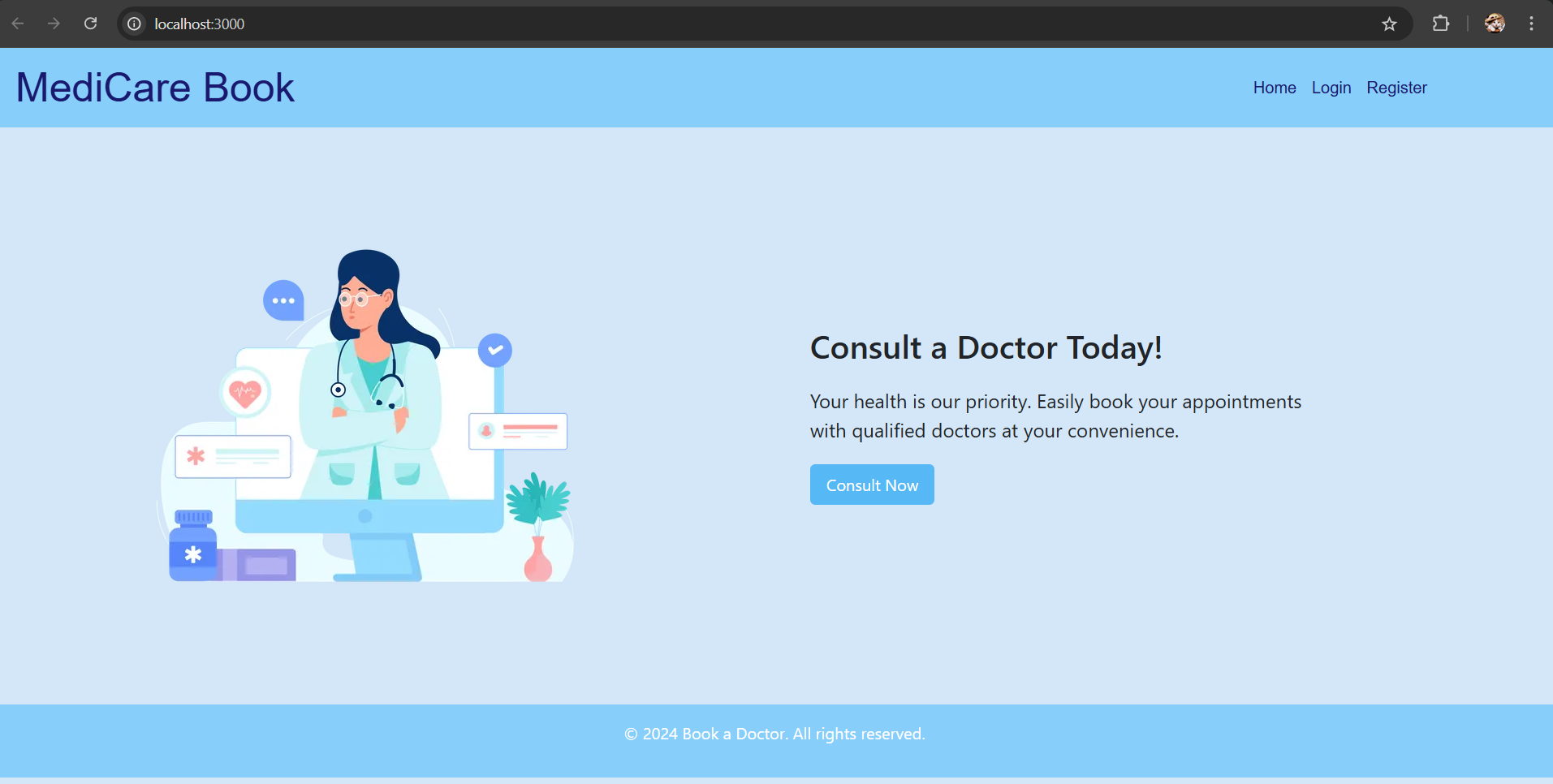


**USER INTERFACE DESIGN**

**1. Landing Page (Common for All Users)**

**Objective**: The landing page serves as the entry point of the application, where users can register or log in.

* **Features**:
  + **Registration Button**: Redirects users to the registration form.
  + **Login Button**: Redirects users to the login form.
  + **Welcome Message**: A simple welcome note explaining the purpose of the platform (e.g., "Book appointments with trusted doctors").
  + **Minimalistic Design**: Focus on ease of navigation and a clean layout with CTA buttons.

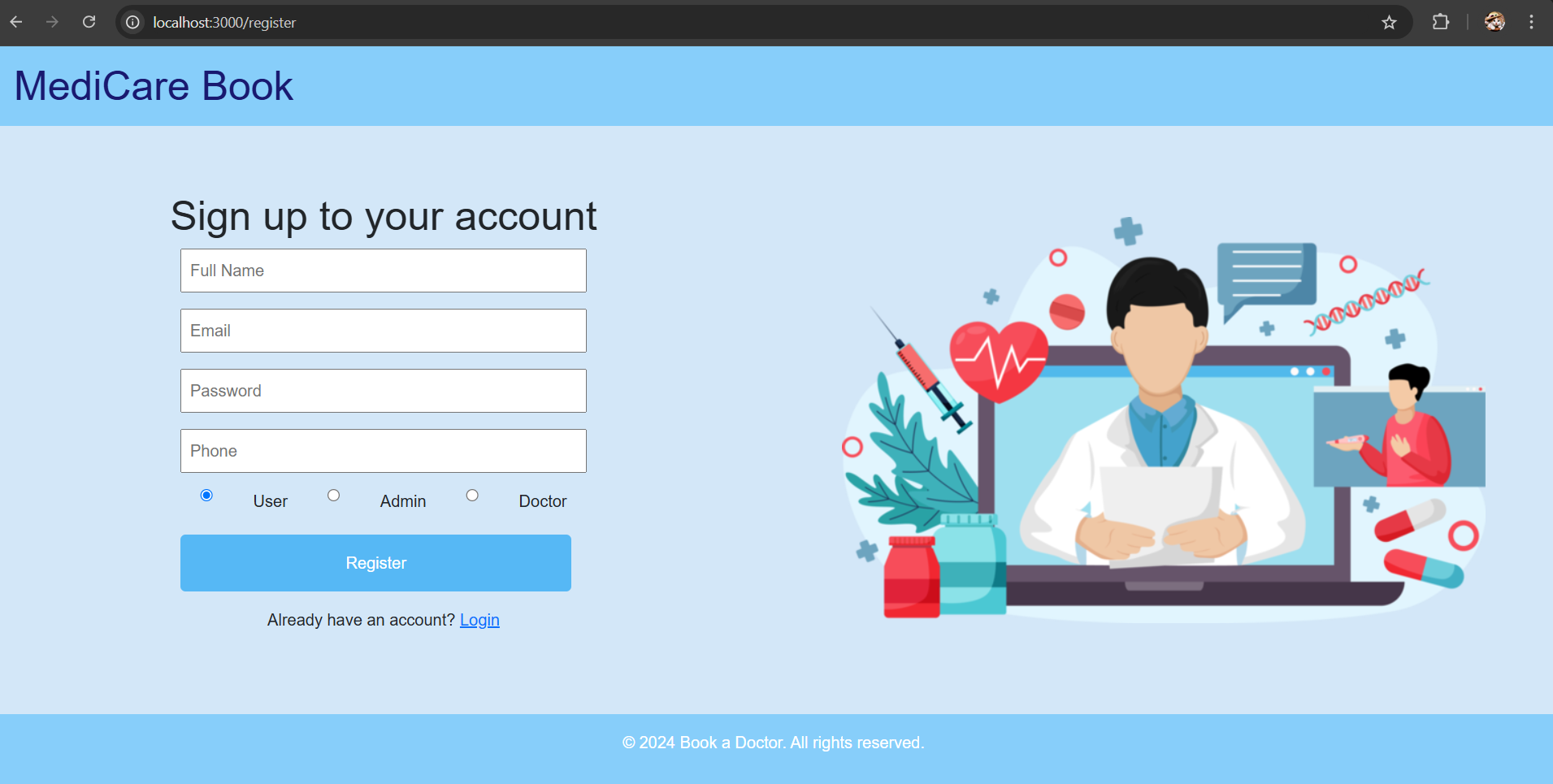


**1. Register Page (Patient/Doctor/Admin Registration)**

**Objective**: The Register page allows users to create a new account by providing basic information such as name, email, password, and role (Patient, Doctor, Admin).

**Features:**

* **Input Fields**:
* **Name**: The user's full name.
* **Email**: The user's email address (unique identifier).
* **Password**: A strong password for authentication.
* **Confirm Password**: To verify that the user typed the password correctly.
* **Role**: Dropdown to select the user’s role (Patient, Doctor, or Admin). This role determines the dashboard access level.
* **Buttons**:
* **Register**: Submits the form data and registers the user.
* **Already have an account?**: A link that redirects to the Login page.

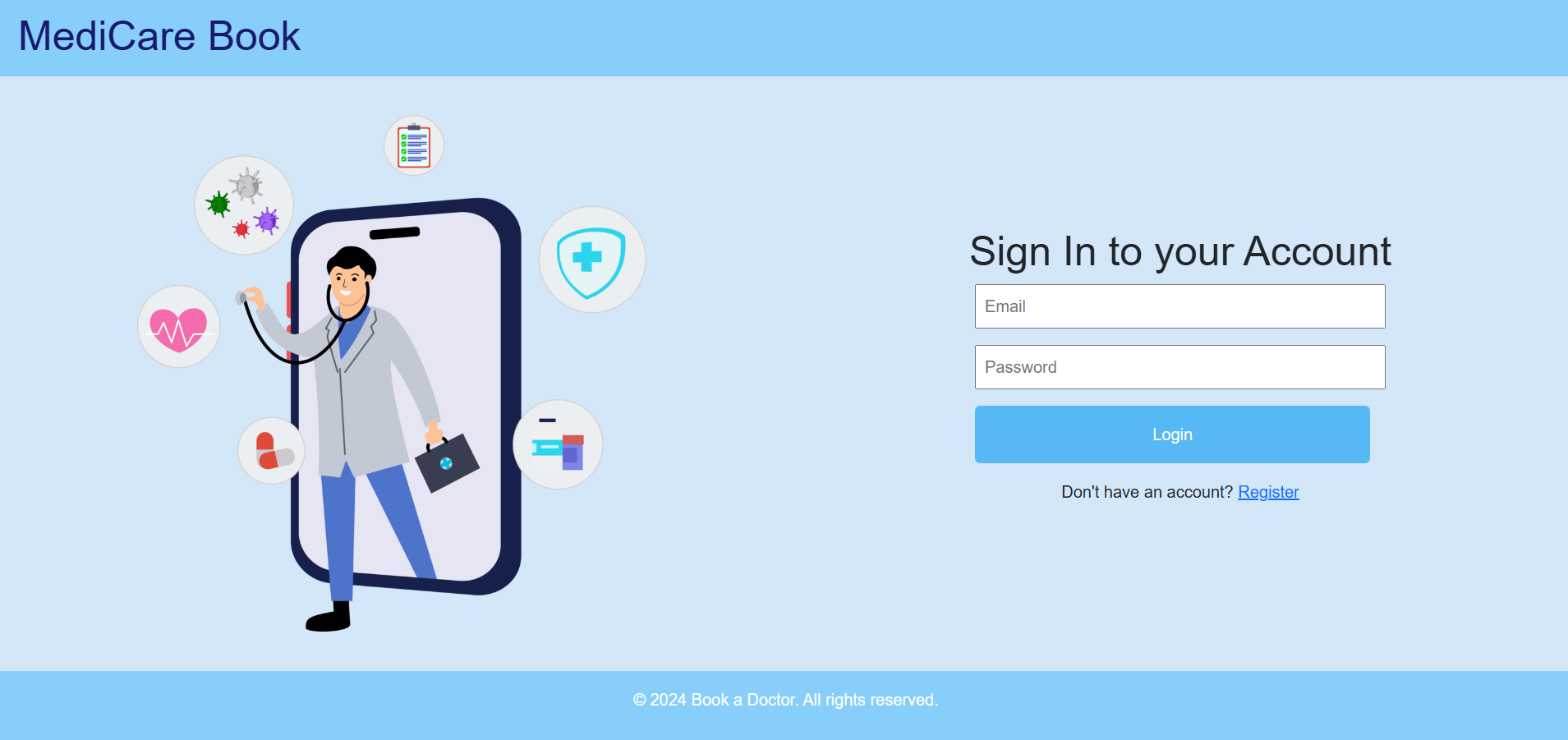


**2. Login Page (Patient/Doctor/Admin Login)**

**Objective**: The Login page allows registered users to log into their accounts by entering their email and password.

**Features:**

* **Input Fields**:
  + **Email**: The user’s email address (should match the one used during registration).
  + **Password**: The user’s password to authenticate.
* **Buttons**:
  + **Login**: Submits the email and password for authentication.
  + **Forgot Password?**: A link to reset the password (if forgotten).
  + **Don’t have an account?**: A link that redirects to the Register page for new users.
* **Validation**:
  + Email and password must be validated against the database.
  + If the credentials are invalid, an error message should be shown.

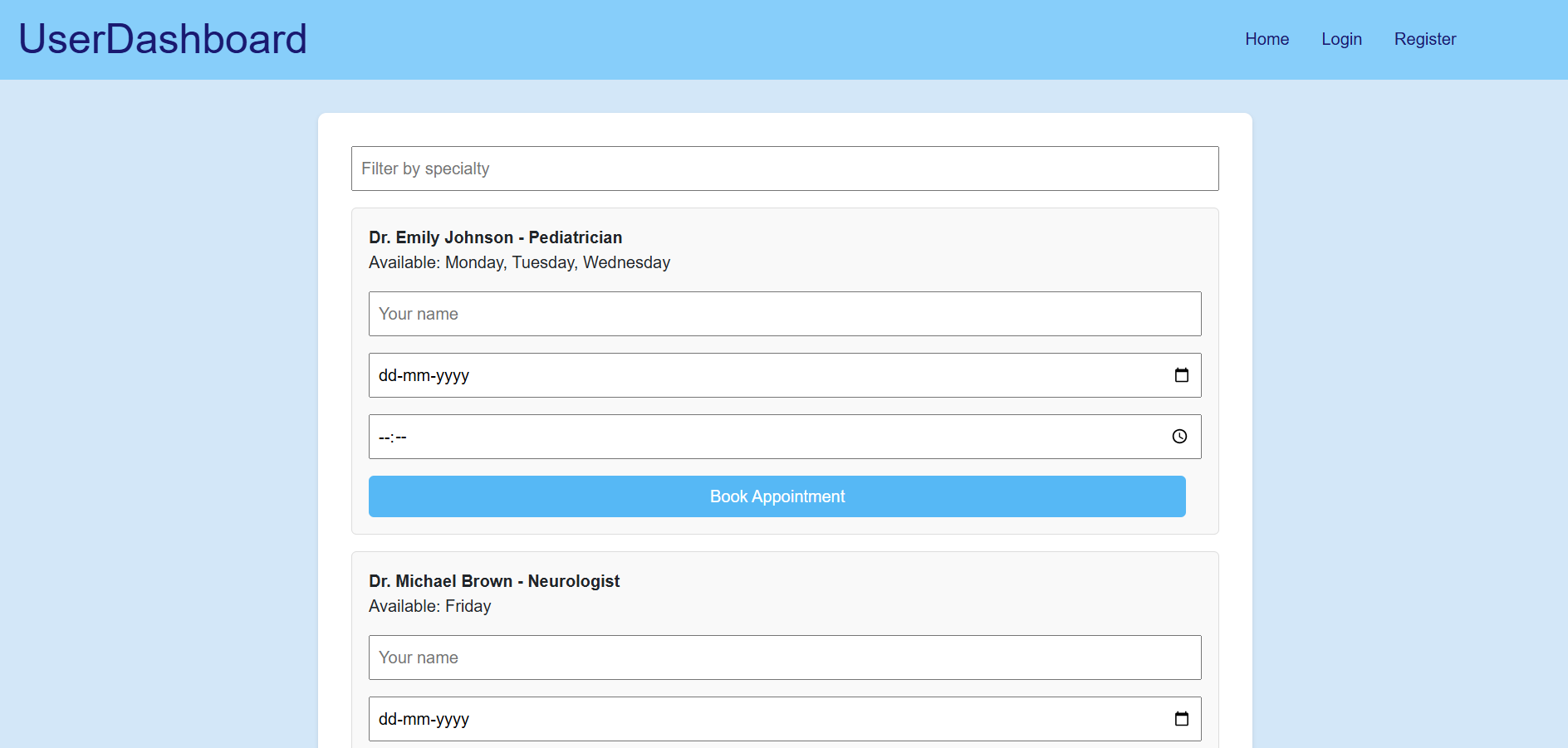


**3. User Dashboard (For Patients)**

**Objective**: After a successful login, patients are redirected to their dashboard where they can view and manage their appointments.

**Features**:

* + **Doctor List**: A list of doctors with their specialization and available days.
  + **Search Functionality**: Search for doctors by name or specialty.
  + **Appointment Form**: Select a doctor, input the patient’s name, and choose a date and time to book an appointment.
  + **Upcoming Appointments**: Display upcoming appointments with the doctor (name, date, time).
  + **Appointment Status**: Each appointment should have a status (e.g., pending, confirmed, canceled).

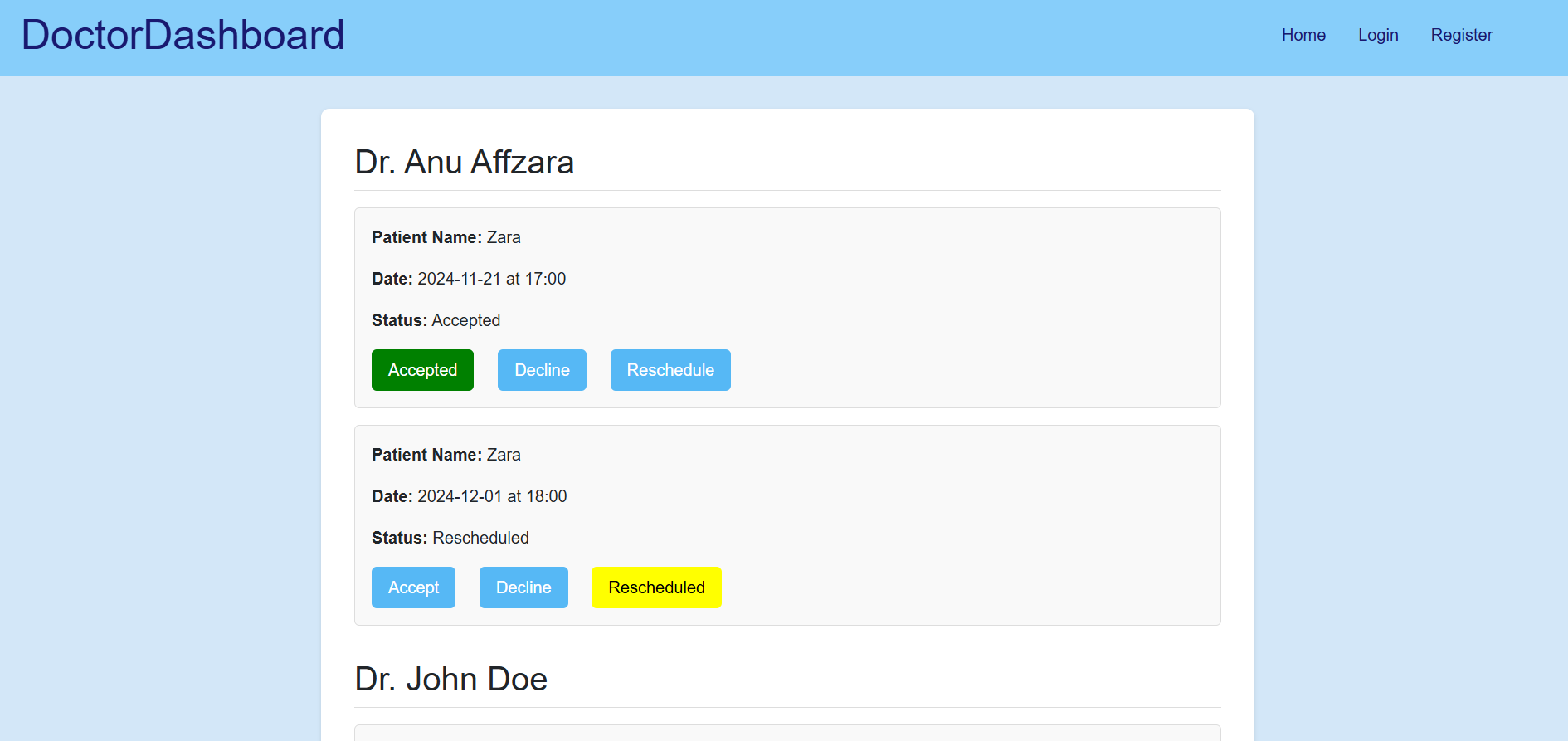


**3. Doctor Dashboard (For Doctors)**

**Objective**: After logging in, doctors are redirected to their dashboard, where they can view and manage patient appointments.

**Features**:

* + **Patient List**: A list of upcoming patient appointments.
  + **Appointment Actions**: Options to **accept**, **decline**, or **reschedule** appointments.
  + **Appointment Details**: View patient’s name, date, time, and reason for the appointment.
  + **Edit Profile**: Option to update personal details and availability.

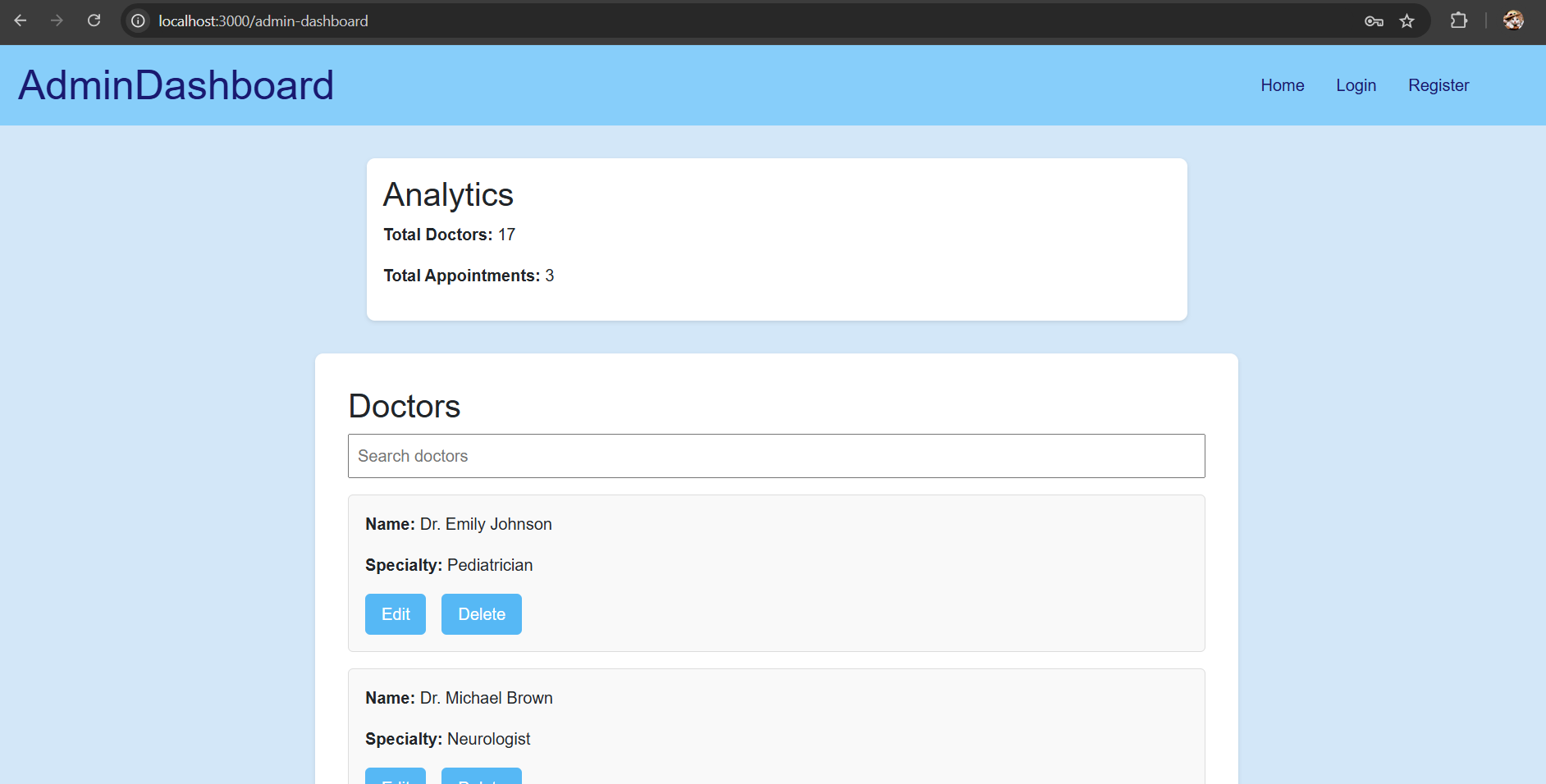


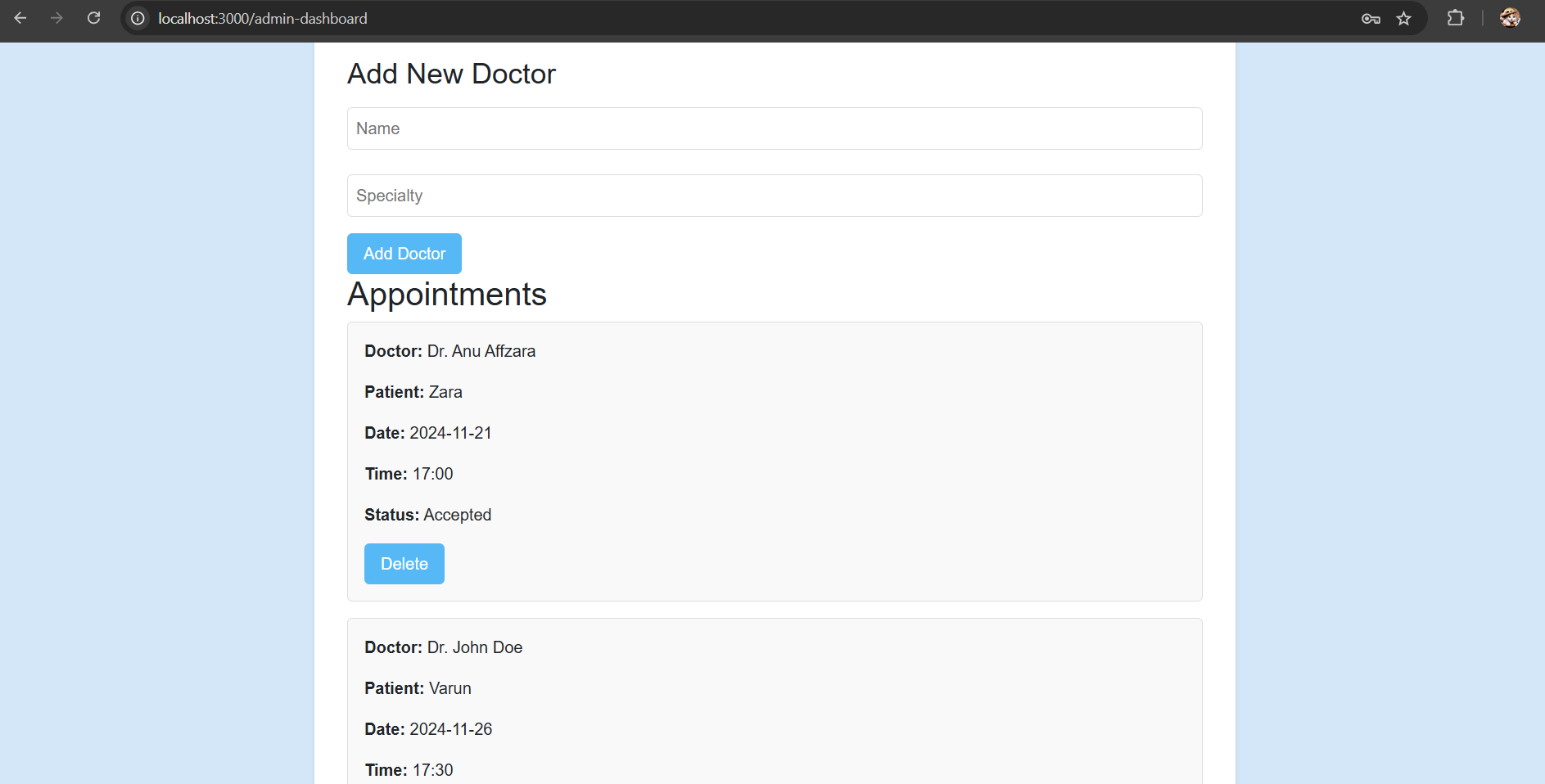
**4. Admin Dashboard (For Admins)**

**Objective**: Admins can manage the entire system, including adding/editing doctors, deleting patients' appointments, and overseeing all user accounts.

**Features**:

* + **Doctor List**: View and manage doctor profiles, including add, edit, or delete actions.
  + **Patient Appointment Management**: View, edit, or cancel any patient appointments.
  + **Manage Users**: View and manage user (patient) accounts.
  + **Search and Filters**: Search doctors by specialization or patient by name.
  + **Action Buttons**: Buttons for actions like "Add Doctor," "Delete Appointment," and "Edit Doctor."





**Challenge: User Authentication and Security**

* **Problem**: Ensuring secure user authentication while preventing unauthorized access, especially since sensitive health data is involved.
* **Solution**: Implement **multi-factor authentication (MFA)** for enhanced security, use **hashed passwords** for storage, and **JWT tokens** for secure session management. Regular **security audits** and data encryption (HTTPS) will ensure safety.

**CONCLUSION**

The **"Book a Doctor"** application successfully addresses the need for a seamless and efficient doctor-patient appointment booking system. By leveraging the **MERN stack** (MongoDB, Express.js, React, Node.js), it provides a secure, scalable, and user-friendly platform. Key features like real-time appointment scheduling, user authentication, and dedicated dashboards for patients, doctors, and admins ensure that the platform caters to the needs of all users effectively.

As the platform evolves, future enhancements such as **telemedicine integration**, **AI-based doctor recommendations**, and **mobile application development** can further improve user experience and accessibility. While challenges like **data privacy**, **scalability**, and **real-time updates** exist, the proposed solutions ensure the platform can grow and adapt to meet the needs of a larger user base, making healthcare more accessible and efficient.

REFERENCE

1. **MERN Stack Documentation**
   * MongoDB: <https://www.mongodb.com/docs/>
   * Express.js: <https://expressjs.com/>
   * React: https://reactjs.org/docs/
   * Node.js: https://nodejs.org/en/docs/
2. **JWT Authentication**
   * JSON Web Tokens (JWT): <https://jwt.io/>
   * Understanding JWT: https://www.digitalocean.com/community/tutorials/understanding-jwt-encoding-decoding
3. **MongoDB Atlas**
   * MongoDB Atlas Setup: <https://www.mongodb.com/cloud/atlas>
4. **React (Frontend Framework)**
   * React Documentation: https://reactjs.org/docs/getting-started.html
5. **Node.js and Express Error Handling**
   * Error Handling in Express: https://expressjs.com/en/guide/error-handling.html