**DocSpot – Seamless Appointment Booking for Health**

Team Members:

TEAM ID : LTVIP2025TMID59705

| **MEMBER** | **RESPONSIBILITY** |
| --- | --- |
| 1. Komaragiri Sri Ram Madhav | Testing & Documentation. |
| 2. Karri Emmanuel Watson | Frontend & Backend. |
| 3. Karri Purnima | Resource Gathering & Database Connection. |
| 4. Karicheti Krishna Chaitanya | Frontend & Documentation. |

**1. Introduction**

Project Title:  
DocSpot – Seamless Appointment Booking for Health

**2. Project Overview**

***Purpose:***  
DocSpot is a full-stack web application designed to streamline the process of booking medical appointments between patients and healthcare providers. The goal is to eliminate the traditional hassles of calling clinics or waiting on hold by offering a user-friendly digital platform where users can register, browse doctors, book, manage, and confirm appointments online.

***Features:***

* User registration and login (Patients & Doctors)
* Browse and filter doctors by specialty, location, availability
* Book appointments with date selection and document upload
* Real-time appointment status updates
* Doctor dashboard for managing schedules
* Admin panel for approving doctors and monitoring activity
* Email/SMS notifications for confirmation and reminders
* Post-visit follow-up summaries and prescriptions

**3. Architecture**

***Frontend:***  
Built using React.js, the frontend follows a component-based architecture:

* Component Structure: Functional components with hooks (useState, useEffect)
* Routing: React Router v6 for navigation
* State Management: Context API for global state (user auth, appointments)
* UI Library: Material-UI for responsive design
* HTTP Requests: Axios for API calls

***Backend:***  
The backend is built using Node.js and Express.js :

* RESTful API endpoints
* Middleware: cors, dotenv, morgan, express-validator
* Authentication: JWT middleware for protected routes
* Modular structure: Routes, Controllers, Models

***Database:***

MongoDB (NoSQL) is used with Mongoose ORM.

**4. Setup Instructions**

***Prerequisites:***

Ensure you have the following installed:

Node.js (v16.x or higher)

npm or yarn

MongoDB Atlas account or local MongoDB instance

Git

***Installation Steps:***

* Clone the repository

git clone <https://github.com/KarriPurnima/DOCSPOT>

cd docspot

* Set up environment variables:

Create .env file in the server directory

PORT=5000

MONGO\_URI=mongodb://localhost:27017/docspot

JWT\_SECRET=mysecretpassword

* Install dependencies :

Backend

cd server

npm install

#Frontend

cd ../client

npm install

**Folder Structure**

Client:

/client

├── public/

├── src/

│ ├── components/

│ │ ├── DoctorList.jsx

│ │ ├── AppointmentForm.jsx

│ │ └── Navbar.jsx

│ ├── context/

│ │ └── AuthContext.js

│ ├── pages/

│ │ ├── Home.jsx

│ │ ├── Login.jsx

│ │ ├── Register.jsx

│ │ └── Dashboard.jsx

│ ├── App.js

│ └── index.js

└── package.json

SERVER:

/server

├── controllers/

│ ├── userController.js

│ └── appointmentController.js

├── models/

│ ├── User.js

│ └── Appointment.js

├── routes/

│ ├── userRoutes.js

│ └── appointmentRoutes.js

├── middleware/

│ └── authMiddleware.js

├── config/

│ └── db.js

├── .env

├── app.js

└── package.json

**6. Running the Application**

To start both servers:

***Frontend:***

Navigate to the client directory:

cd client

Start the development server:

npm start

The frontend application runs on: http://localhost:3000

***Backend:***

Navigate to the server directory:

cd server

Start the backend server:

npm start

The backend application runs on: http://localhost:5000

**7. API Documentation**

Base URL:

http://localhost:5000/api

Auth Endpoints:

POST /auth/register

Registers a new user (patient or doctor) with name, email, password, and role.

POST /auth/login

Logs in a registered user and returns a JWT token for authentication.

Appointment Endpoints:

GET /appointments (Protected)

Retrieves all appointments for the logged-in user.

POST /appointments (Protected)

Creates a new appointment with selected doctor, date, reason, and optional documents.

PUT /appointments/:id/status (Protected)

Updates the status of an existing appointment (e.g., confirmed, cancelled).

DELETE /appointments/:id (Protected)

Deletes an appointment by its ID.

User Endpoints:

GET /users/doctors (Protected)

Returns a list of available doctors for booking.

GET /users/me (Protected)

Returns the profile information of the currently logged-in user.

**8. Authentication**

Authentication is handled using JWT (JSON Web Tokens) :

Upon successful login, the system generates a JWT token.

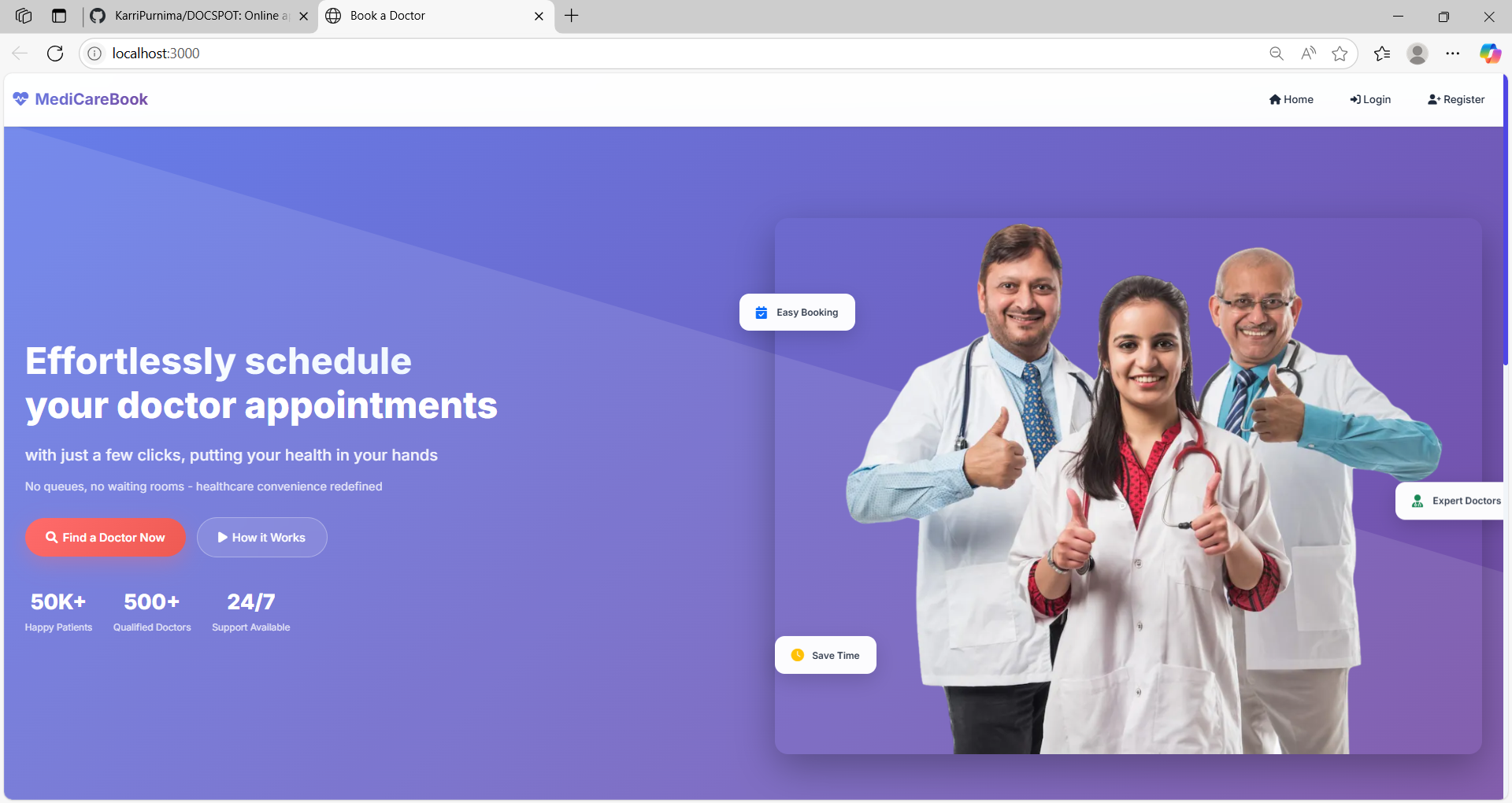
This token is stored in the browser's localStorage.

All protected routes require a valid token for access.

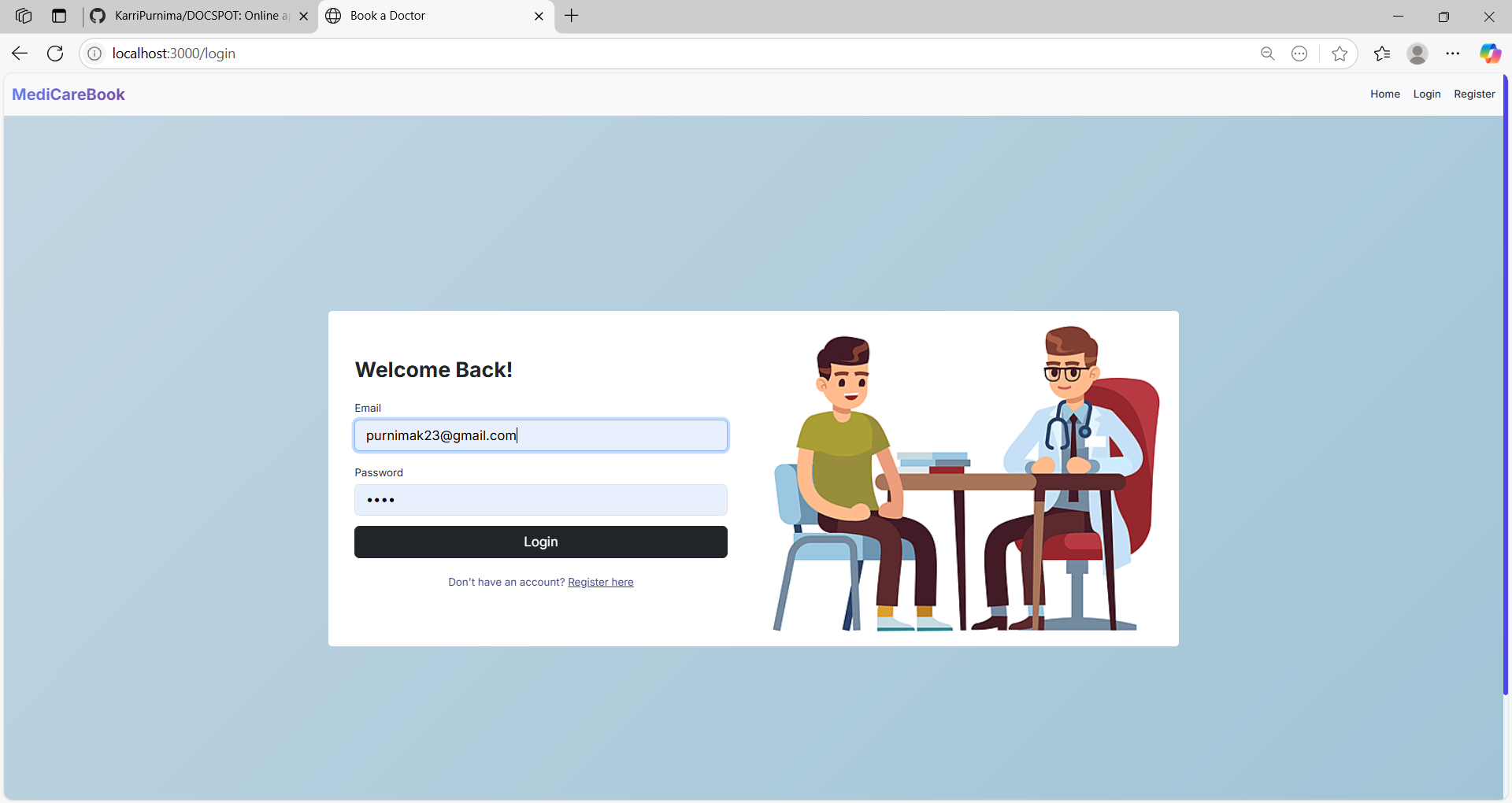
Middleware verifies the token and authorizes access based on user roles: patient, doctor, or admin.

**9. Screenshots or Demo**

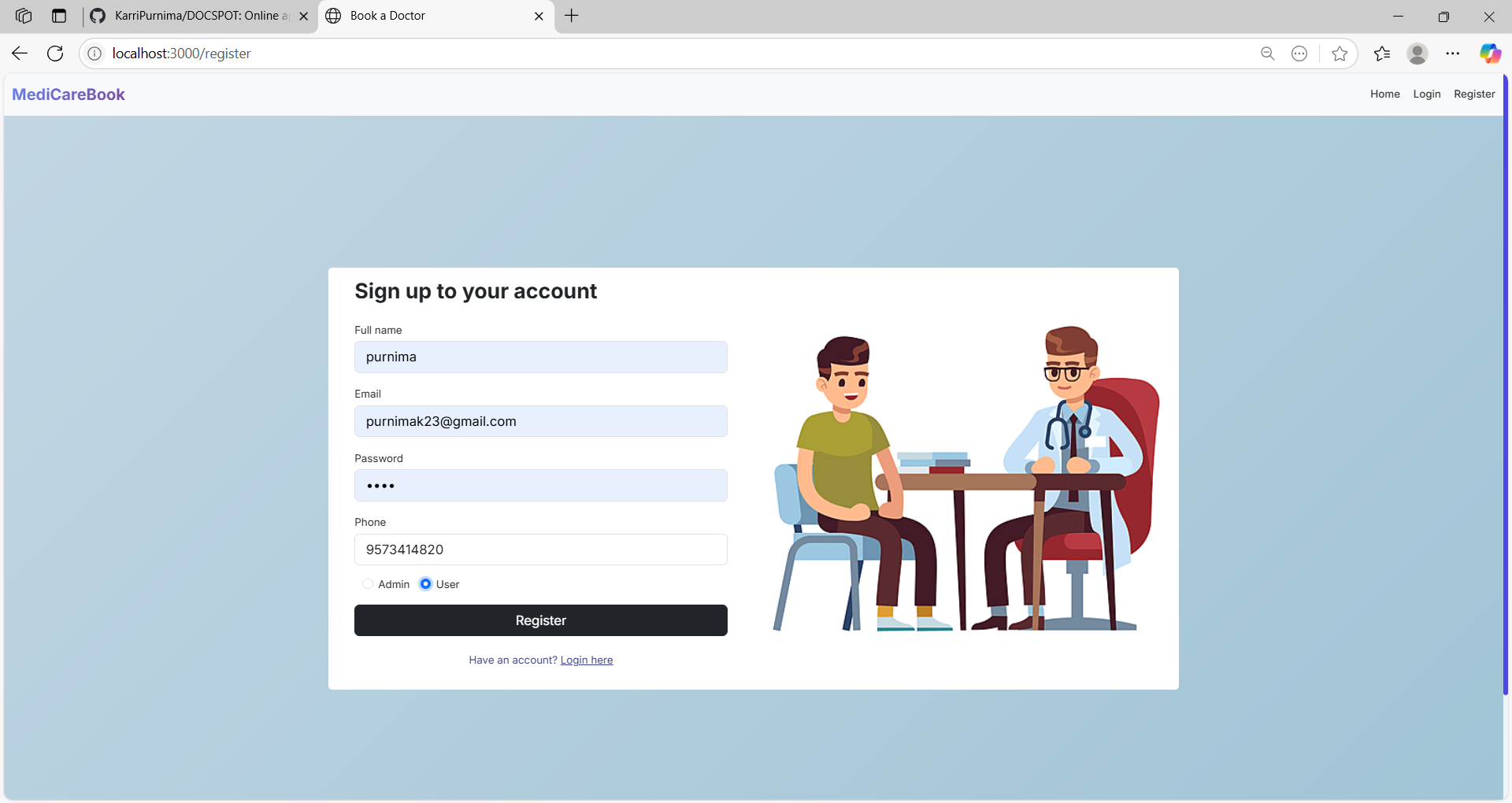
HOME PAGE :



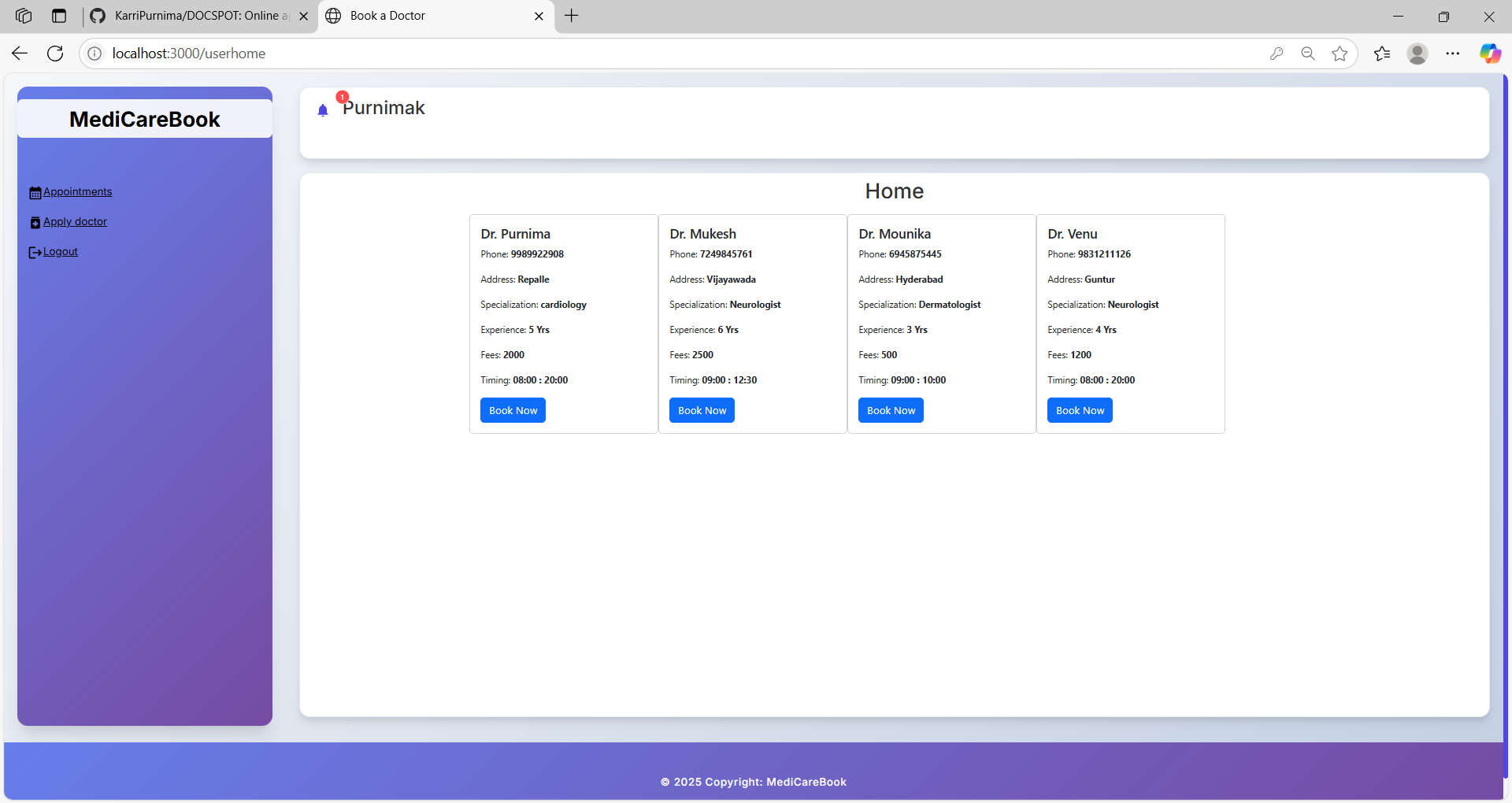
LOGIN PAGE:



REGISTER PAGE :



DASHBOARD:



**10. Testing**

***Testing Strategy:***

Frontend components are tested using Jest and React Testing Library.

Backend APIs are tested using Mocha and Chai for validation of responses and error handling.

End-to-end testing is performed using Cypress to simulate real-user interactions.

**11. Known Issues**

The appointment status may not update instantly without manually refreshing the page.

Document upload functionality currently supports only PDF file types.

The admin approval workflow requires manual triggering in some deployment environments.

Dashboard tables have limited responsiveness on mobile devices.

**12. Future Enhancements**

Integrate video consultation features for telehealth services.

Allow patients to rate and review doctors after appointments.

Implement an AI-based symptom checker for better doctor recommendations.

Sync appointments with external calendars like Google Calendar for reminders.

Add multi-language support to cater to diverse users.

Provide assistance with insurance claims directly through the app.