**Full Stack Development with MERN**

**Project Documentation format**

1. **Introduction**

* **Project Title**: Book a doctor using MERN
* **Team Members:**

Varanasi Hemasai Reddy - Backend

Mohit Aggarwal - Frontend

Abhishek Gangwar - Backend and API Integration

Pankh Bansal - Frontend

1. **Project Overview**

* **Purpose:**

This project aims to develop a web-based doctor appointment booking system using the MERN stack that simplifies the process of scheduling medical appointments. The platform aims to provide patients with an easy way to search for doctors and book available time slots online while enabling doctors to manage their availability and view appointments. Additionally, the system offers admin functionalities for user management and supports secure authentication and online payments to enhance convenience, accessibility, and healthcare efficiency.

* **Features:**

1. User Registration & Login

* Secure registration via mail

1. Role-Based Dashboards

* Personalized dashboard for each user role.
* Patients: view bookings, make payments.
* Doctors: Manage slots, view appointments.
* Admins: Approve doctors, manage users.

1. Doctor Search

* View doctor profiles and available time slots.

1. Appointment Booking

* Real-time slot selection and booking by patients.
* Booking history and status tracking.

1. Payment Integration

* Stripe API integration for secure online payments.
* Payment confirmation linked to appointment status.

1. Admin Management Panel

* System activity monitoring and dashboard control.

1. Secure Authentication

* JWT-based token authentication and protected routes.

1. **Architecture**

**Frontend:**

The front end is built using React.js, which provides a component-based architecture to build reusable.

**Routing**: Managed using React Router DOM to switch between views like home, login, dashboard, doctor profile, etc.

**Pages**:

* Home, Login, Signup
* My Account (Patient dashboard)
* Doctor Dashboard
* Checkout Success

**Components:**

DoctorCard, BookingCard, Appointments, DoctorSidePanel, etc.

**State Management:**

* Basic state handled via useState, useContext (e.g., AuthContext for logged-in user)
* Hooks like useFetchData and useGetProfile handle data fetchin

**Styling:**

Tailwind CSS is used for consistent, responsive, utility-first styling

**Backend:**

The backend is responsible for handling business logic and serving APIs to the frontend.

**Structure:**

* Routes (/Routes) – Different files for auth, users, doctors, bookings
* Controllers (/Controllers) – Functions to handle logic for each route
* Models (/models) – Mongoose schemas for User, Doctor, and Booking
* Middleware:
* authenticate – Checks for valid JWT token
* restrict – Restricts route access based on user roles (e.g., patient, doctor)

**Key Functionalities:**

* User & doctor registration/login with token generation (JWT)
* Protected routes (e.g., profile access, appointments)
* Doctor appointment booking with Stripe Checkout
* Secure update and delete endpoints for users and doctors

**Middleware & Security:**

* CORS enabled to allow frontend-backend communication
* Passwords are encrypted (assumed to use bcrypt, if not, it's recommended)
* Token validation is required for most actions

**MongoDB:**

MongoDB is used to persistently store data like users, doctors, and bookings. We use \*Mongoose\*, an ODM (Object Data Modeling) library, to interact with MongoDB easily.

**Schemas:**

* UserSchema: Stores user info (name, email, password, photo, gender, etc.)
* DoctorSchema: Extends on user data with fields like specialization, experience, qualifications, photo, and time slots
* BookingSchema: Stores references to the user and doctor who booked, appointment date, payment status, and ticket price

**Data Relationships:**

* Booking references both Doctor and User using ObjectId
* Populate () method used in queries to fetch full user or doctor details during appointment queries

1. **Setup Instructions**

**Prerequisites:**

* Node.js ≥ v14
* MongoDB (local or Atlas)
* Stripe account (test mode for payments)

**Installation:**

**Step 1:** Clone the repository

git clone https://github.com/yourusername/mediconnect.git

cd

**Step 2:** Setup Environment Variables

Create. env in the backend/ folder with:

PORT=5000

**MONGO\_URL**= mongodb+srv://abhishekg1826:0501Aman1826@cluster0.hjp5t.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0

**JWT\_SECRET\_KEY**= Gvw6/WIYoJXHLvrqmbEMu7fN5xQVpQkT4T/TD9lf/4OQb/QjJJdHm3C3gXEsuuw5cMPyl1MhnGAaQCc6INA5Hl8yDadzbwecz0M/EKKYbiQQzh/WD/2i5htm2MRnHeDgSvHqvrnhYMPPf804IV1QVn/rbzV3YWQw03O7XtyjWBcLNGTs+okd+Yx7Q5K0C5GizadoMEEW4UGN6DPdlTnCV6ggQdCFn27NEmYrMW9RVzvJRzeesQx7Asfowsd39/i28AM2WZ6T948XWDTZHElZU6WbwmdGhvM3oLwfNFzc0mnhpu8cFD/L2XARHWWm1Ypi/r93wiG8FAaz184hMAj+1w==

**STRIPE\_SECRET\_KEY**= sk\_test\_51RBfuM2QPQfIfXLnWoHnqyNvSGsFUCdSrJpIAmObILhgjizYYEk2uLRLfD5TYBwE4iuPoqwNHqiMqtMZU9Fz0FLQ00iBv3qHna

**CLIENT\_SITE\_URL**=http://localhost:5173

**Step 3:** Install Dependencies

cd backend

npm install

cd frontend

npm install

1. **Folder Structure**

**Client Side:**

public: Contains static assets like favicon and logos.

Src: Root folder for all React source code.

Assets: Stores images, icons, and other media used in the UI.

Components: Reusable React components like Navbar, Footer, DoctorCard, FormInput.

Pages: Contains full-page components like Login, Register, Dashboard, etc.

Utils: Helper functions, API wrappers, toast messages, and constants.

Layouts: Common layout structures (e.g., for protected routes or shared page templates).

Router: Manages application routing using React Router DOM.

App.jsx: Root component where routing and layout are defined.

main.jsx: Application entry point that renders App.jsx into the DOM.

config.js: Contains configuration constants like base API URLs.

index.css: Global styles and Tailwind CSS directives.

• **Server:** (Backend = Nodejs + Express)

**auth**

* verifyToken.js: JWT-based authentication & role restrictions

**controllers**: Business logic for users, doctors, and bookings

* authController.js
* userController.js
* doctorController.js
* bookingController.js

**models:** Mongoose schemas

* UserSchema.js
* DoctorSchema.js
* BookingSchema.js

**Routes:** Express route handlers

* auth.js
* user.js
* doctor.js
* booking.js

**.env**: Environment variables (Mongo URI, Stripe keys, etc.)

**index.js:** Entry point for server setup

**package.json:** Backend dependencies and scripts

**config:** Configuration files (e.g., DB connection logic)

1. **Running the Application**

**Frontend:**

* cd frontend
* npm run dev

**Backend:**

* cd backend
* npm run start-dev

1. **API Documentation**

**Auth Routes:**

* POST /api/v1/auth/register – Register new user/doctor
* POST /api/v1/auth/login – Login user/doctor

**User Routes:**

* GET /api/v1/users/profile/me – Get current user profile
* GET /api/v1/users/appointments/me-appointments – Get my bookings
* PUT /api/v1/users/:id – Update user
* DELETE /api/v1/users/:id – Delete user

**Doctor Routes:**

* GET /api/v1/doctors/ – Get all doctors
* GET /api/v1/doctors/:id – Get doctor by ID
* GET /api/v1/doctors/profile/me – Get logged-in doctor profile
* PUT /api/v1/doctors/:id – Update doctor
* DELETE /api/v1/doctors/:id – Delete doctor

**Booking Routes:**

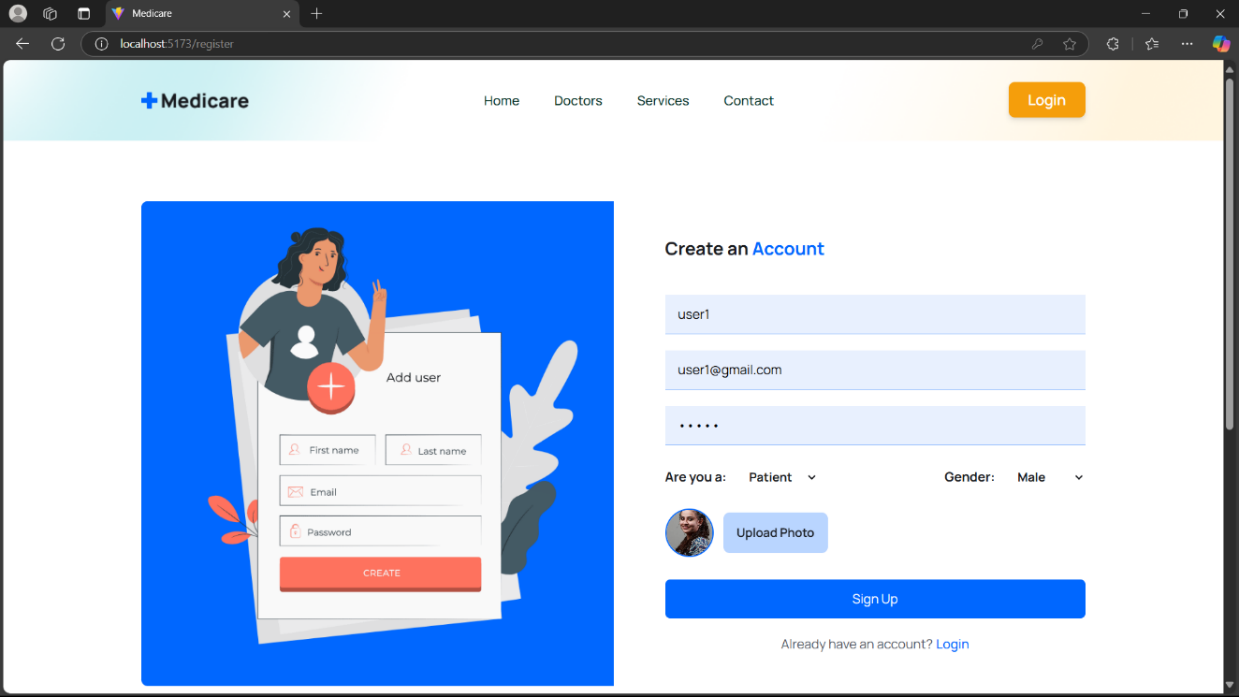
* POST /api/v1/bookings/checkout-session/:doctorId/:userId – Create Stripe session

1. **Authentication**

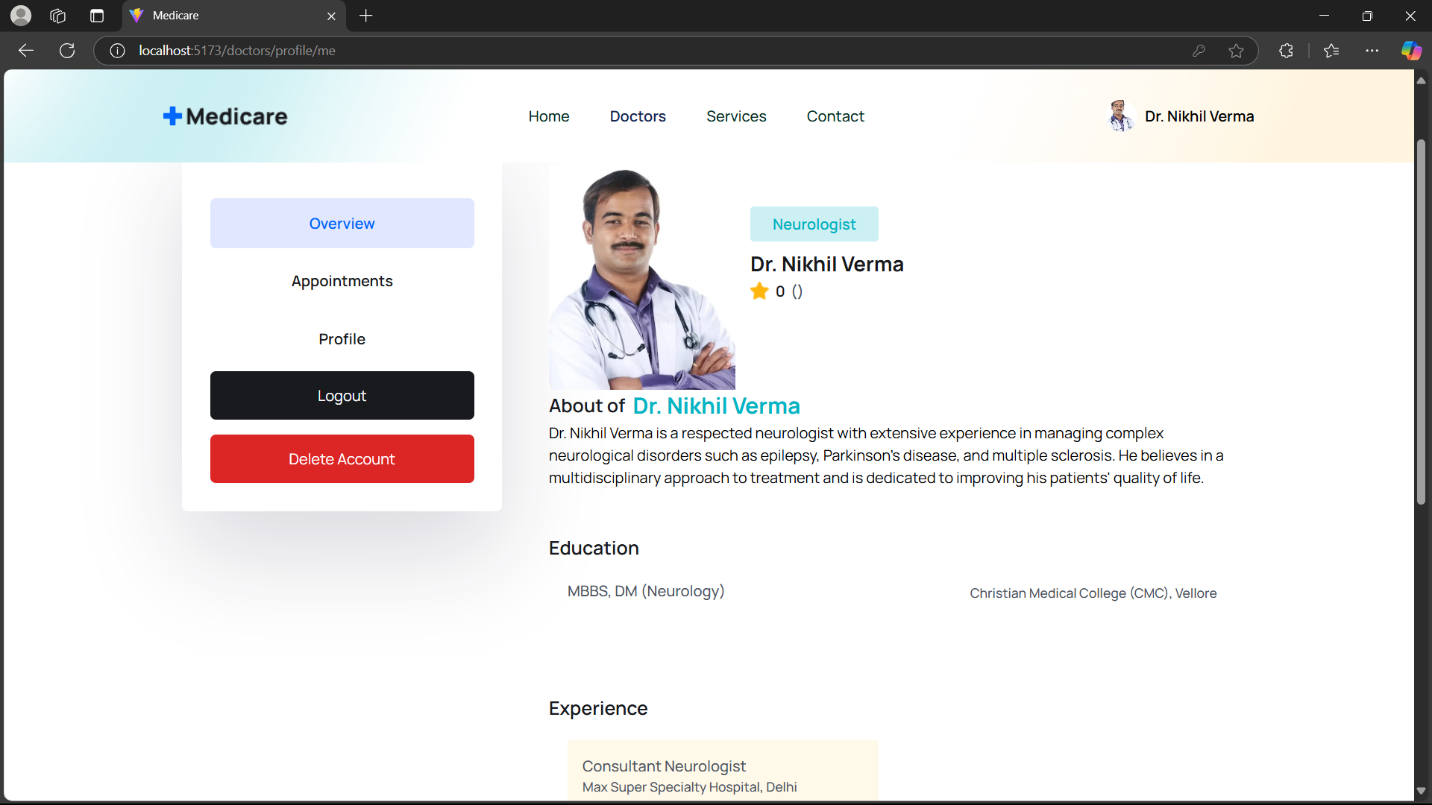
* JWT-based Authentication
* Token generated on login and stored in localStorage
* Middleware authenticate verifies tokens
* Role-based access via restrict(['patient']) or restrict(['doctor'])

1. **User Interface**

**Registration:**



**Doctor Overview:**



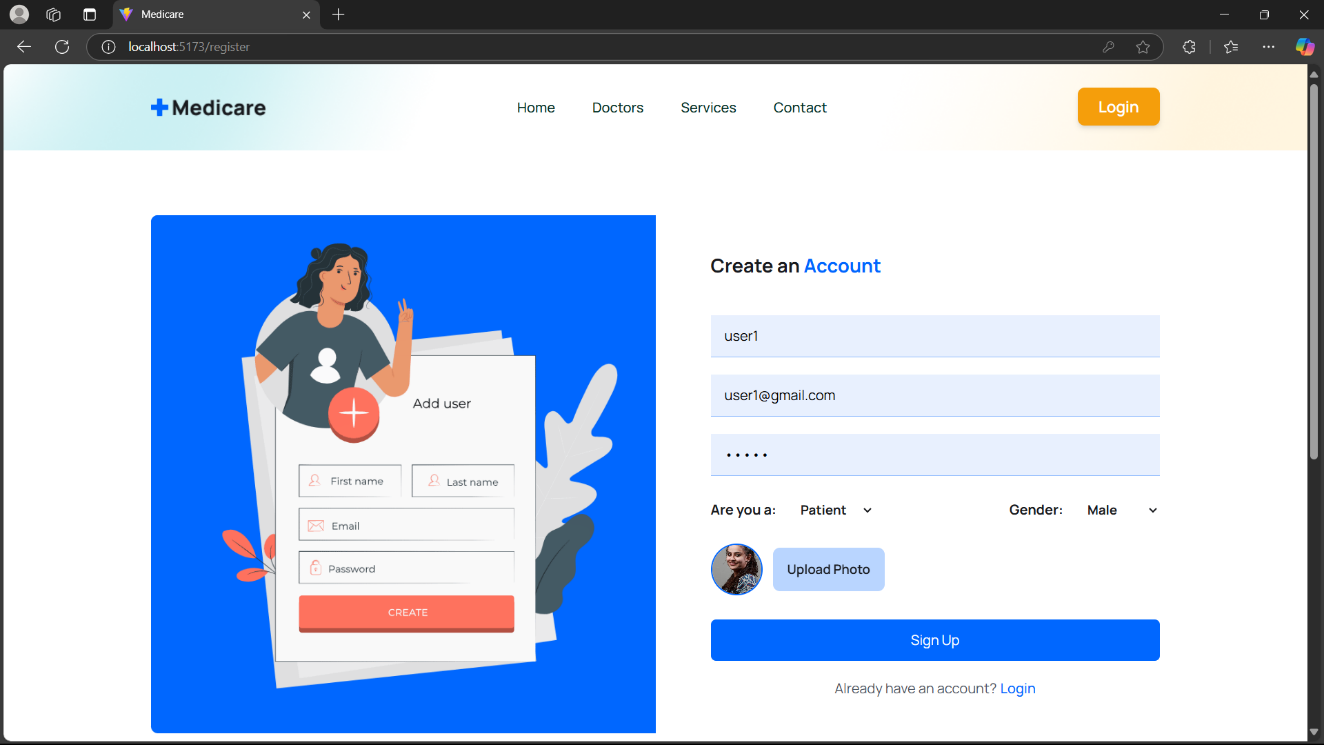
1. **Testing**

Manual testing using:

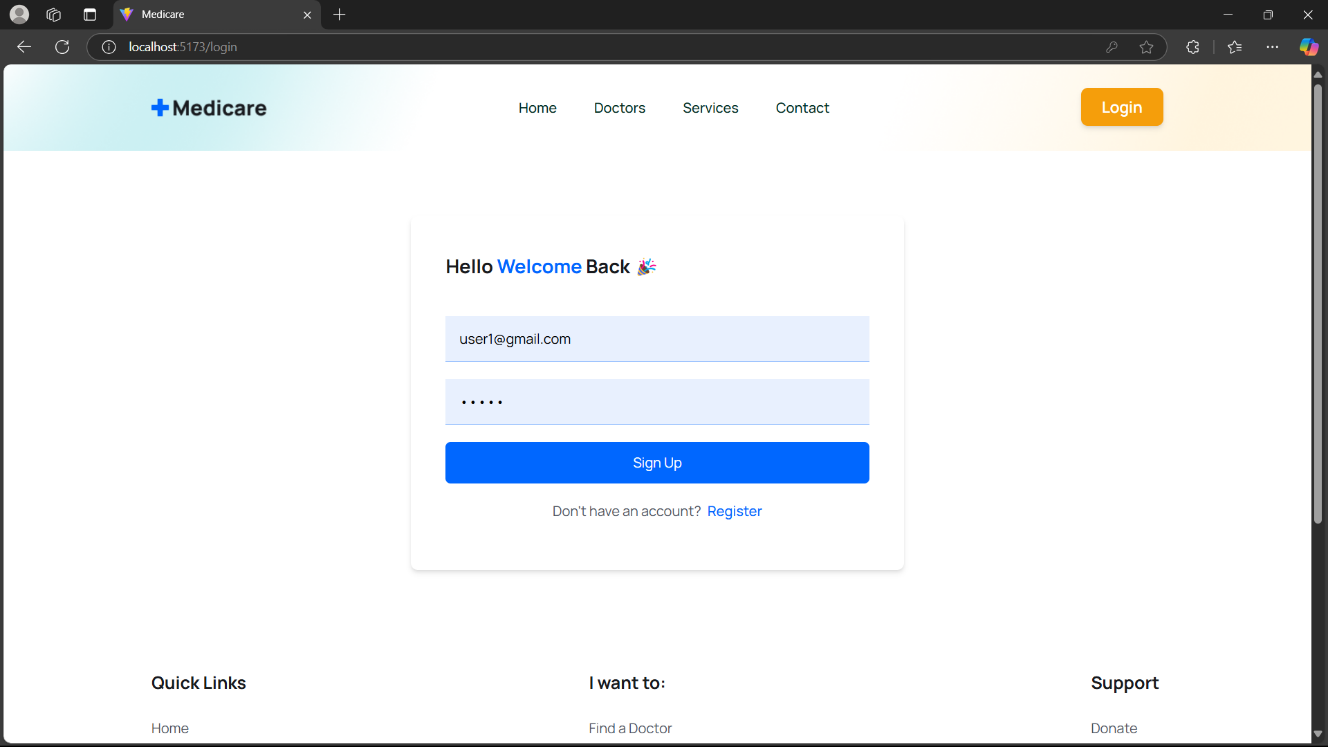
* Postman for API endpoints
* React Developer Tools for state inspection
* Basic validation for forms
* Error handling for failed API calls and image uploads

1. **Screenshots or Demo**

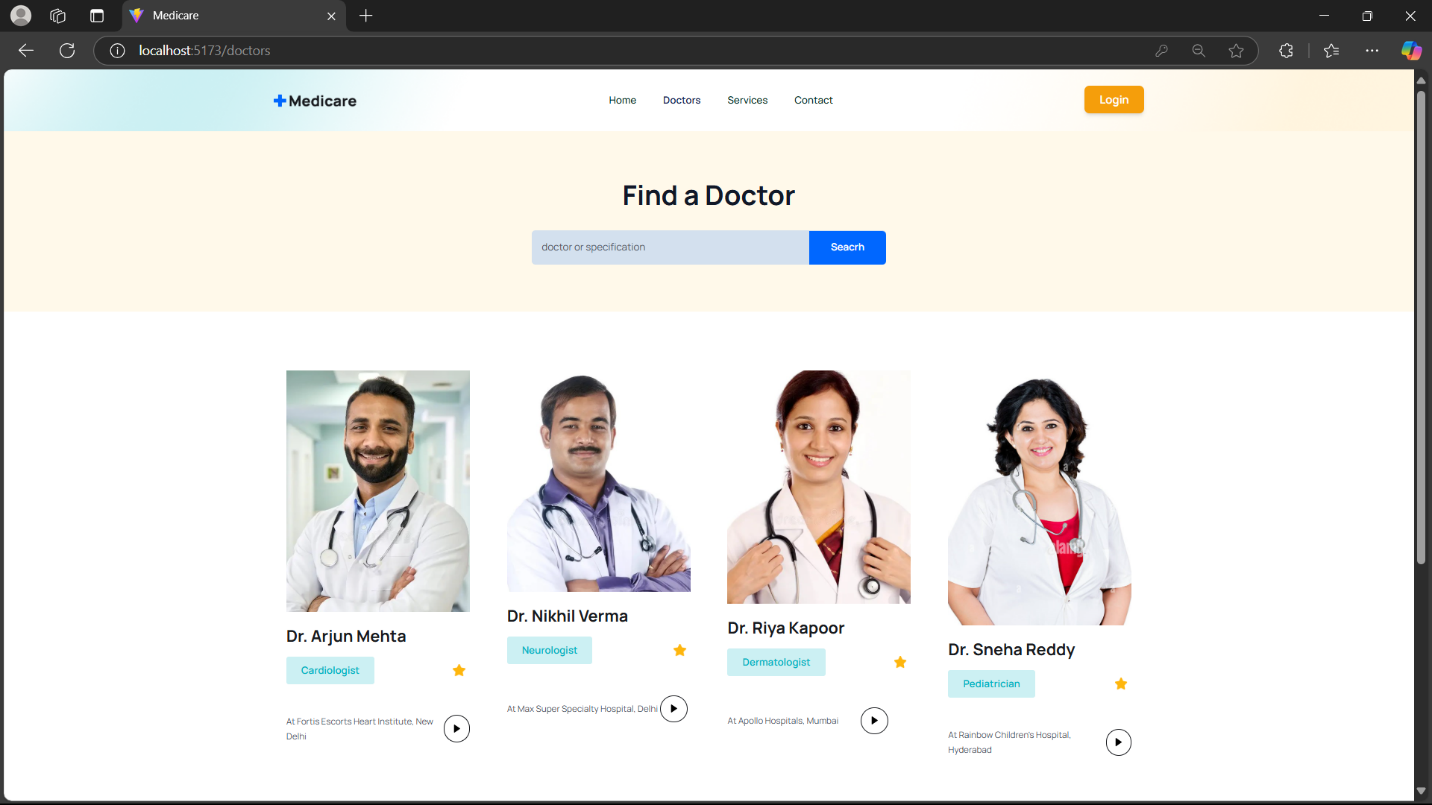
**Registration**



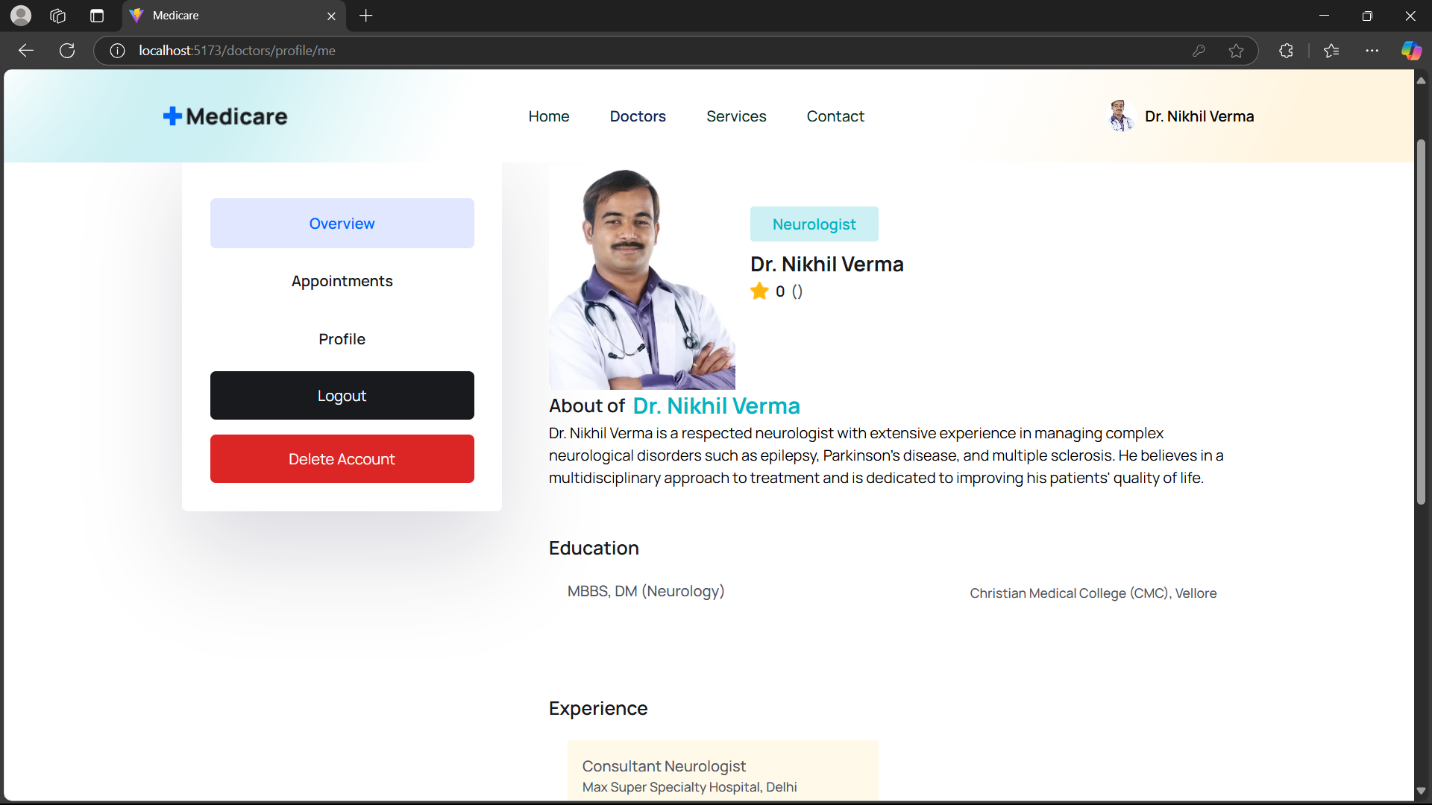
**Login**



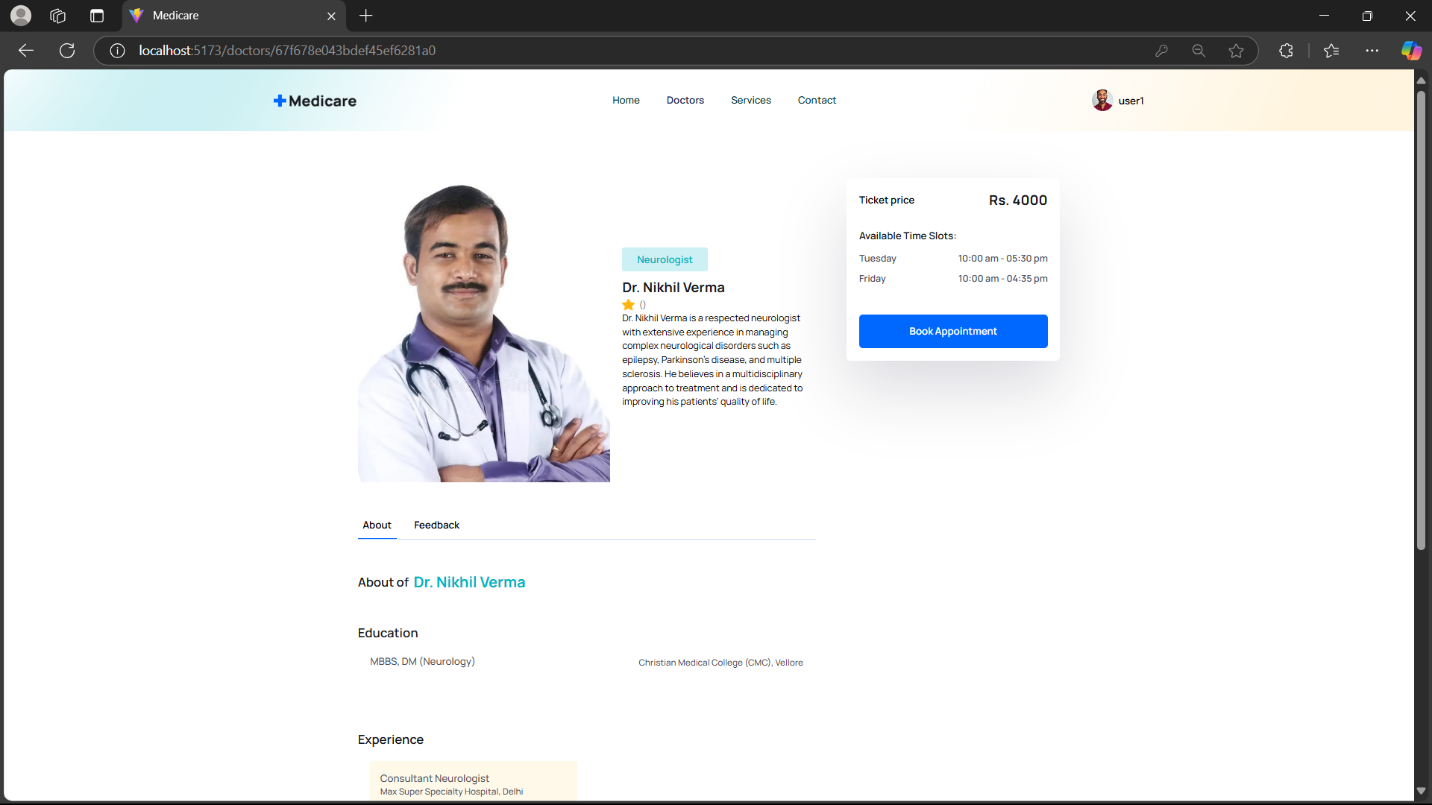
**Doctors**



**Doctors Overview**



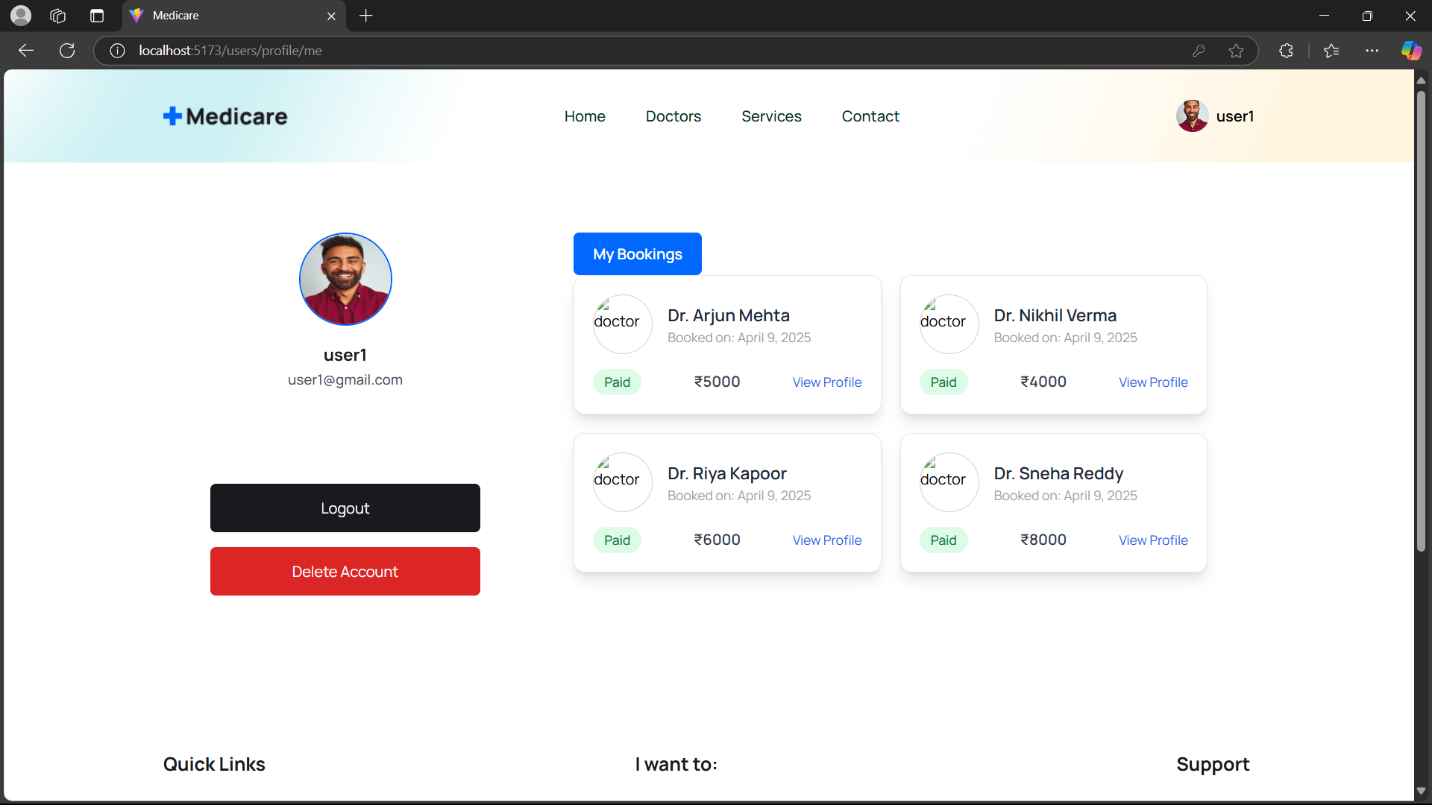
**About the Doctor**



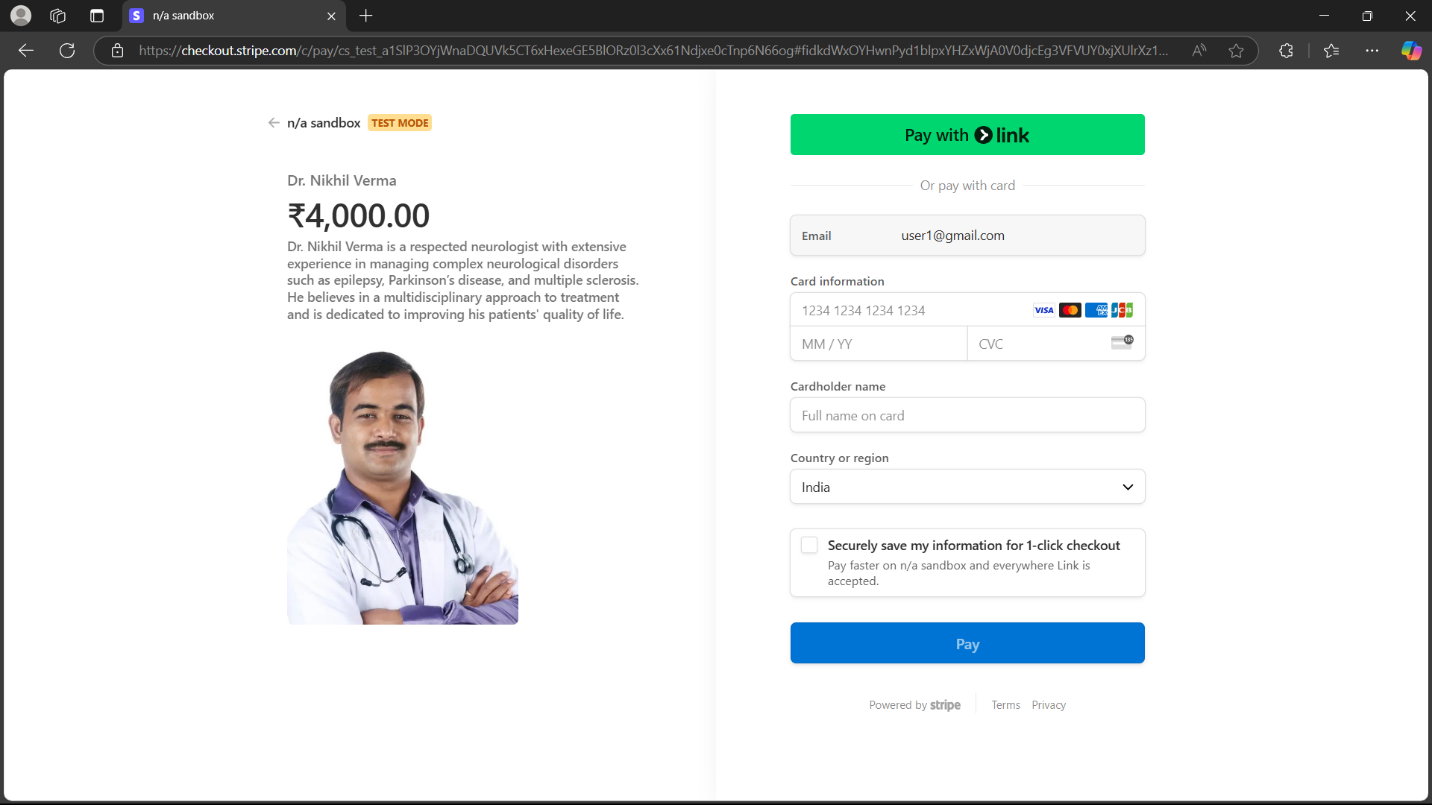
**Appointment**



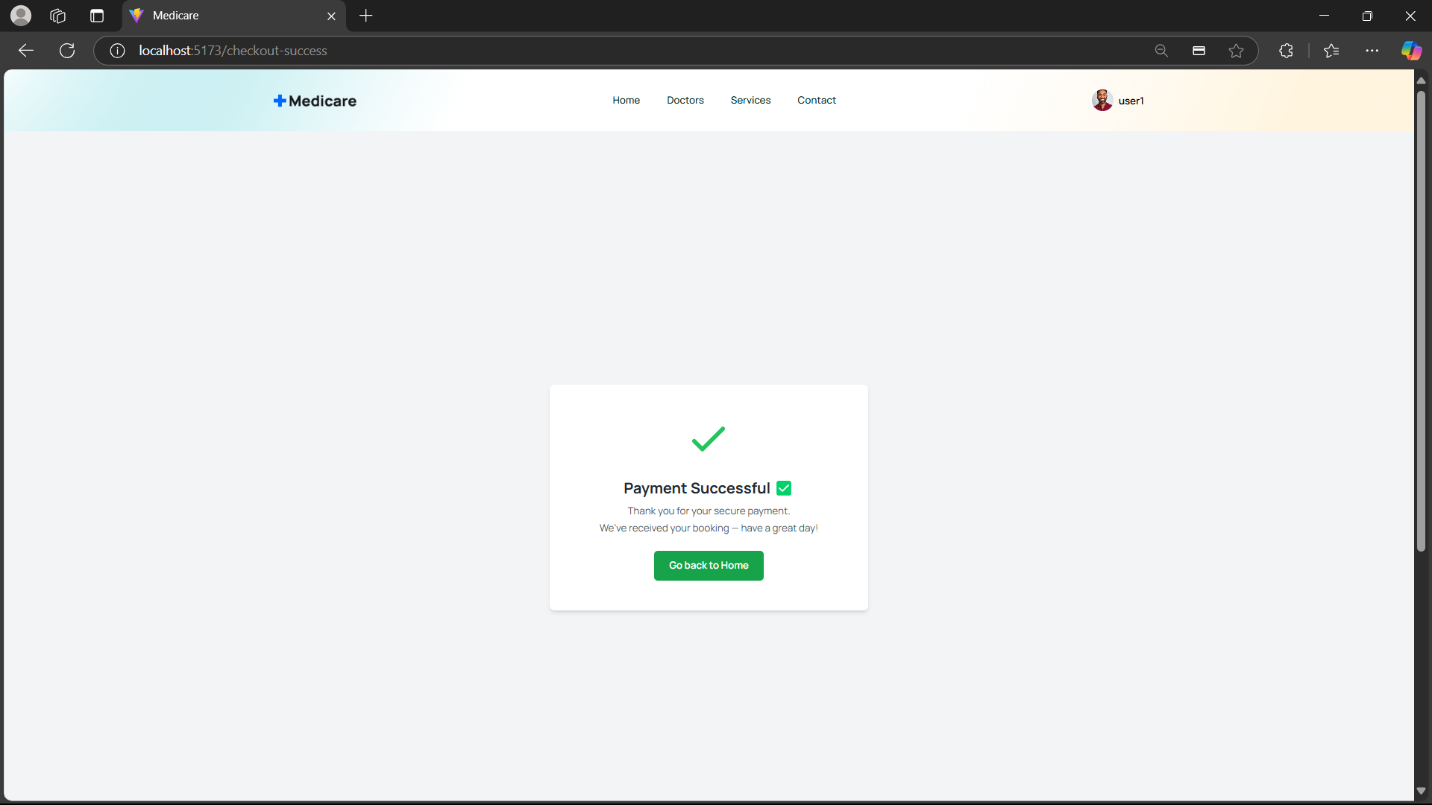
**User Bookings**



**Payment**



**Payment Successful**



**Demo Link :-**

https://drive.google.com/file/d/1J5gXmthSkCDK6ui0DsVA4JbRiePr3sR1/view?usp=drive\_link

1. **Known Issues**

* Doctor photo may not display if not uploaded via Cloudinary
* Admin dashboard not implemented
* Limited validation on form inputs

1. **Future Enhancements**

* Developing a mobile app version of the platform for Android and iOS would provide greater accessibility and convenience for users on the go.
* Integration of SMS or email notifications to remind patients and doctors of upcoming appointments or updates can improve user engagement and reduce missed bookings.
* Allowing doctors and patients to sync appointments with Google Calendar or Outlook can streamline scheduling and time management.
* Adding advanced search filters for specialties, availability, and locations can make it easier for patients to find the right doctor quickly.
* Incorporating multiple language options will make the platform more inclusive, especially for users in rural or diverse regions.
* An analytical panel with appointment trends, user activity, and system usage statistics can help admins make informed decisions.
* Implementing smart algorithms to suggest the best available slots based on user history and preferences.
* Allowing patients to rate and review doctors’ post-appointment can help maintain service quality and trust.