**TITLE PAGE**

**Fracto – Online Doctor Appointment Booking Platform**

**Name**  : Mosa Naveena Grace

**Date**  : 8 September 2025

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| S.NO | CONTENTS | PAGE.NO |
| 1.  2.  3.  4.  5.  6.  7. | Problem Definition and Objectives  Frontend & Backend Architecture  Component Breakdown & API Design  Database Design & Storage Optimization  Testing  Implementation Screenshots  Conclusion | 2  3  4 – 6  7 – 8  9-10  11- 19  20 |

**Problem Definition and Objectives**

**Definition**

Booking doctor appointments is often time-consuming and inconvenient. Patients usually have to visit hospitals or clinics to check availability, which leads to long waiting times and confusion. There is no single platform where people can easily search for doctors based on city, specialization, or available time slots.

On the other side, doctors and admins face difficulties in managing appointments, cancellations, and patient records manually. This creates mismanagement, delays, and poor patient experience.

So, there is a need for an online system where patients can quickly find doctors, book or cancel appointments, and admins can manage everything in one place.

**Objectives**

* Develop a responsive **Angular frontend** for patients and admins.
* Implement a **secure backend** using ASP.NET Core MVC with JWT-based authentication.
* Design and maintain a **SQL Server database** to store users, doctors, appointments, ratings, and specializations.
* Allow **users** to:
  + Register, log in, and manage their profile.
  + Search doctors by city, specialization, rating, and available slots.
  + Book and cancel appointments.
* Allow **admins** to:
  + Manage users and doctors (CRUD operations).
  + Approve or cancel appointments.
  + Send confirmation and cancellation notifications.
* Provide **Swagger-based API documentation** for easy testing and integration.

**Frontend & Backend Architecture**

**1. Overview of Technology Stack**

**Frontend (Angular)**

* **Framework:** Angular
* **Purpose:** Provides an interactive and responsive user interface for patients and admins.
* **Features:**
  + User authentication (login/register).
  + Search doctors by city, specialization, ratings.
  + Appointment booking and cancellation.
  + Admin dashboards for managing doctors, users, and appointments.
  + State management using Angular services.
  + Routing handled by Angular Router.

**Backend (ASP.NET Core MVC with Web API)**

* **Framework:** ASP.NET Core MVC with RESTful Web API
* **Purpose:** Handles all business logic, security, and communication between frontend and database.
* **Features:**
  + JWT-based authentication & authorization.
  + CRUD operations for users, doctors, appointments, ratings.
  + Business logic for appointment scheduling and availability checks.
  + Image upload support (doctor profile pictures).
  + API documentation and testing via Swagger.

**Database (SQL Server with Entity Framework Core)**

* **Database Engine:** Microsoft SQL Server
* **ORM:** Entity Framework Core
* **Purpose:** Stores all structured data including users, doctors, appointments, ratings, and specializations.
* **Schema Highlights:**
  + **Users Table:** UserId, Username, Password, Role.
  + **Doctors Table:** DoctorId, Name, SpecializationId, City, Rating.
  + **Appointments Table:** AppointmentId, UserId, DoctorId, AppointmentDate, TimeSlot, Status.
  + **Specializations Table:** SpecializationId, SpecializationName.
  + **Ratings Table:** RatingId, DoctorId, UserId, Rating.

**Component Breakdown & API Design**

**1. Frontend Component Breakdown (Angular)**

**a. State Management**

* Angular services + RxJS observables used for managing state.
* Authentication state (JWT token, logged-in user) stored in **localStorage/sessionStorage**.
* Doctor search filters (city, specialization, rating) maintained in service-level state.
* Appointment booking state passed between components via services.

**b. Routing (Angular Router)**

* Routes are protected using **AuthGuard**

**c.Major UI Components**

* **Shared Components**
  + navbar.component.ts → Navigation bar with login/logout and links.
* **Auth Pages**
  + login.component.ts → User login form.
  + register.component.ts → New user registration form.
* **Doctor Pages**
  + doctor-list.component.ts → Lists available doctors with filtering.
* **Appointments**
  + book-appointment.component.ts → Appointment booking form.
  + my-appointments.component.ts → User’s appointment history.
* **Admin**
  + manage-doctors.component.ts → Add/edit/remove doctors.
  + admin-appointments.component.ts → View and manage all appointments.
* **Ratings**
  + ratings-page.component.ts → Submit and view doctor ratings.

**2. Backend API Design (ASP.NET Core Web API)**

**Authentication & Security**

* **JWT (JSON Web Token)** used for authentication.
* Login generates JWT; token is passed in **Authorization header** (Bearer <token>) for protected routes.
* Role-based authorization (User, Admin).

**API Design :-**

**AdminUsers**

GET /api/admin/users

POST /api/admin/users

PUT /api/admin/users/{id}

DELETE /api/admin/users/{id}

**Appointments**

POST /api/Appointments

GET /api/Appointments

PUT /api/Appointments/cancel/{id}

GET /api/Appointments/user/{userId}

GET /api/Appointments/pending

PUT /api/Appointments/approve/{id}

**Auth**

POST /api/Auth/register

POST /api/Auth/login

**Doctors**

GET /api/Doctors

POST /api/Doctors

GET /api/Doctors/{id}

PUT /api/Doctors/{id}

DELETE /api/Doctors/{id}

GET /api/Doctors/{id}/timeslots

**Ratings**

POST /api/Ratings

GET /api/Ratings/doctor/{doctorId}/avg

GET /api/Ratings/doctor/{doctorId}

**Specializations**

GET /api/Specializations

POST /api/Specializations

GET /api/Specializations/{id}

PUT /api/Specializations/{id}

DELETE /api/Specializations/{id}

**Users**

GET /api/Users/{id}

POST /api/Users/{id}/avatar

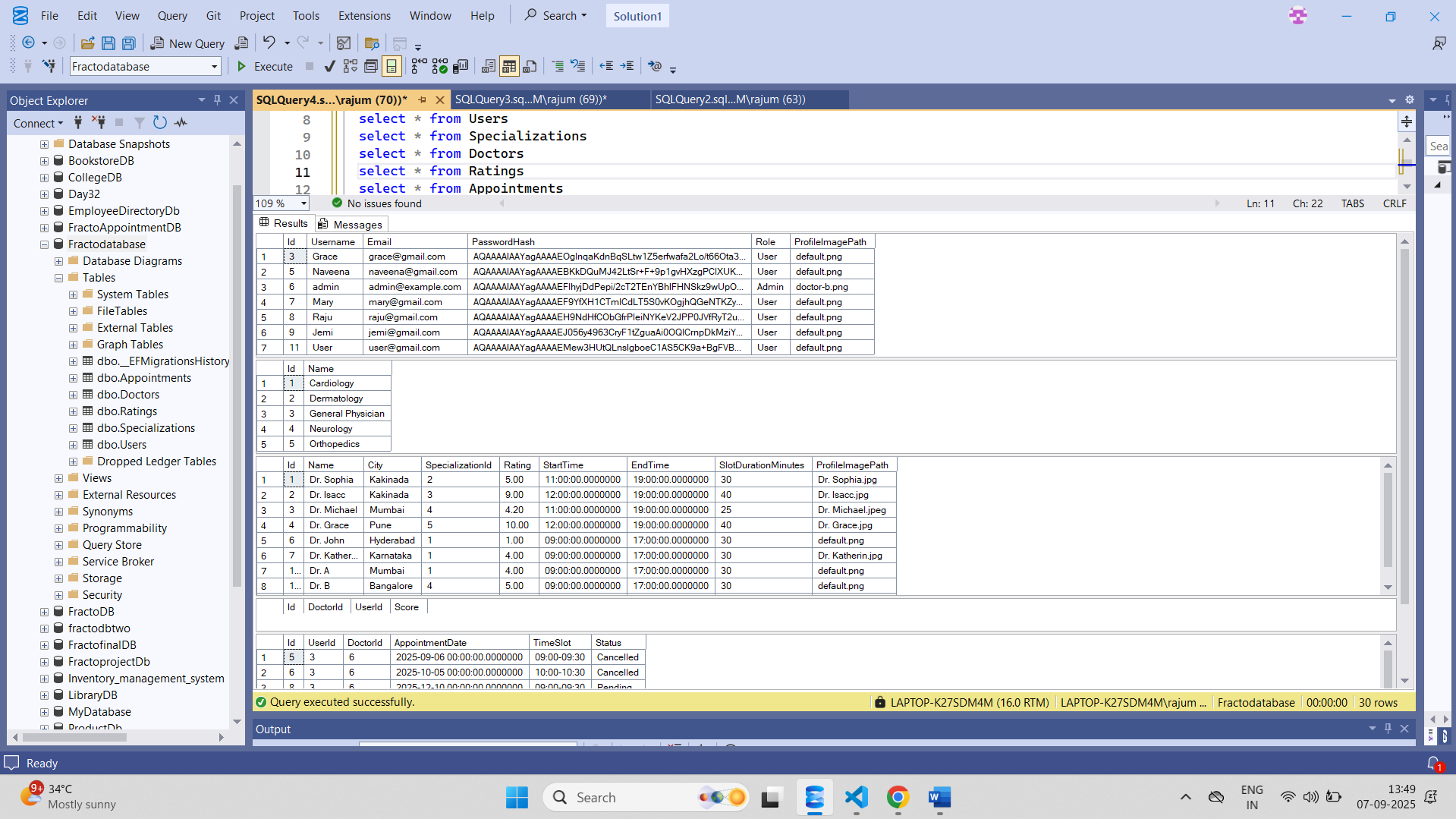
**Database Design & Storage Optimization**

**1. Entity-Relationship Diagram (ERD)**

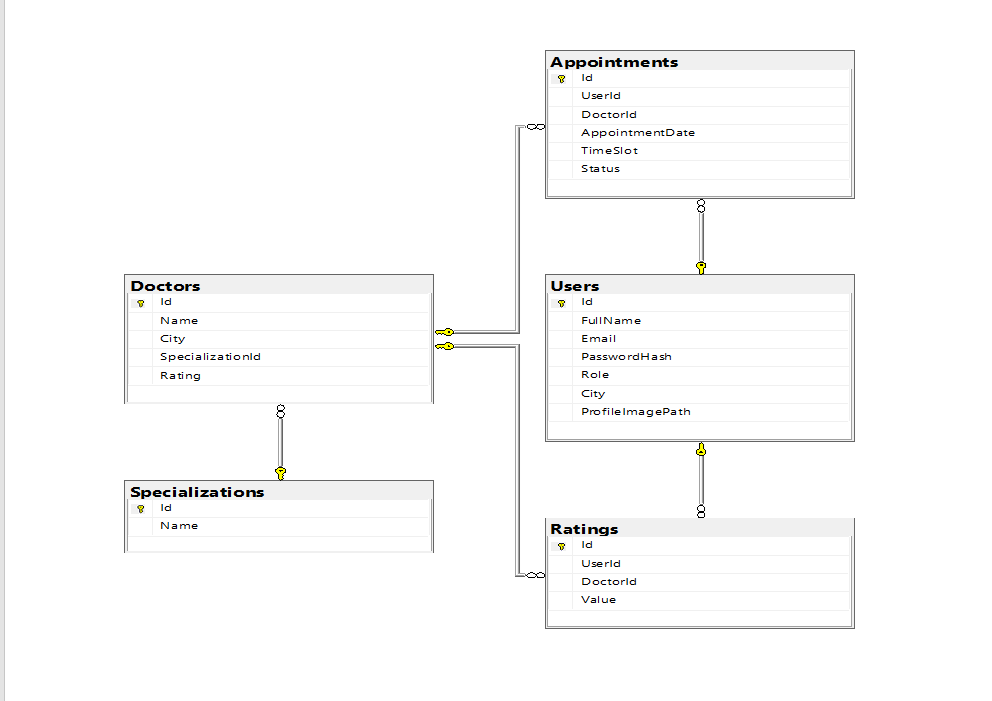
Here’s the **ERD structure** (tables and relationships):

**Relationships**

* **Users ↔ Appointments:** One-to-Many (a user can have many appointments).
* **Doctors ↔ Appointments:** One-to-Many (a doctor can have many appointments).
* **Doctors ↔ Specializations:** Many-to-One (a doctor belongs to one specialization).
* **Doctors ↔ Ratings:** One-to-Many (a doctor can have multiple ratings).
* **Users ↔ Ratings:** One-to-Many (a user can give ratings to different doctors).

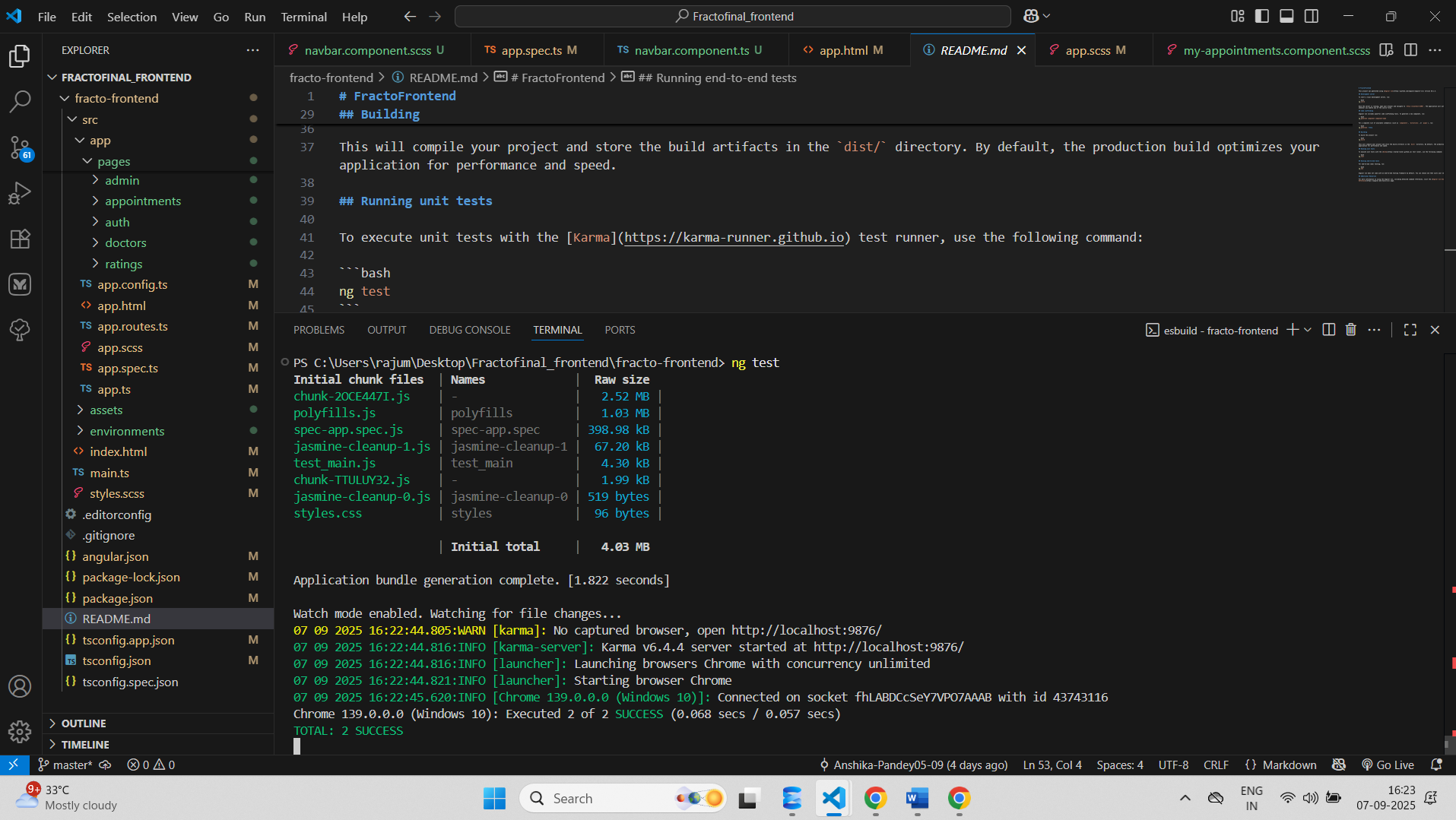


**ER Diagram:**

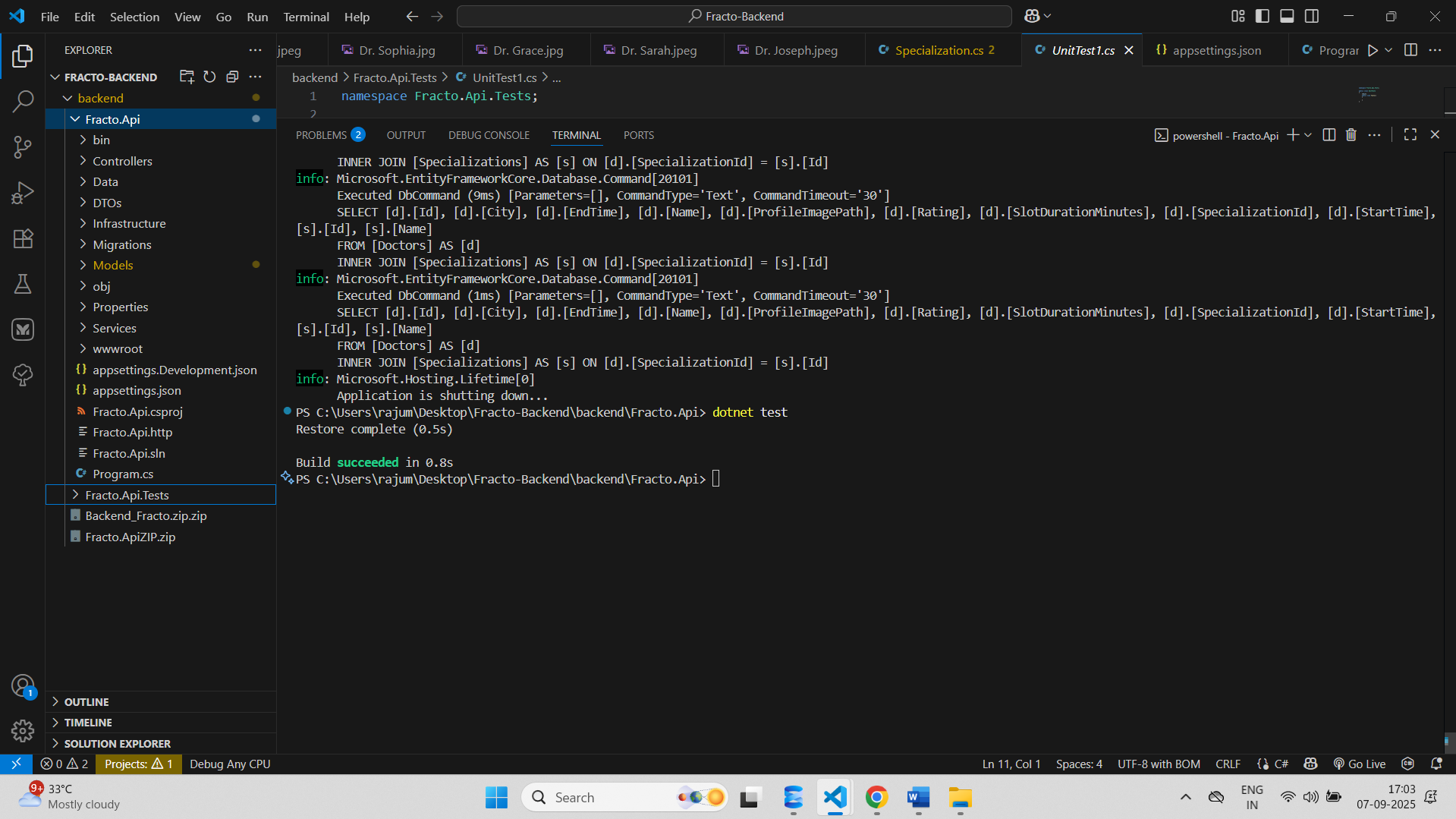


**Testing**

Frontend Testing :-

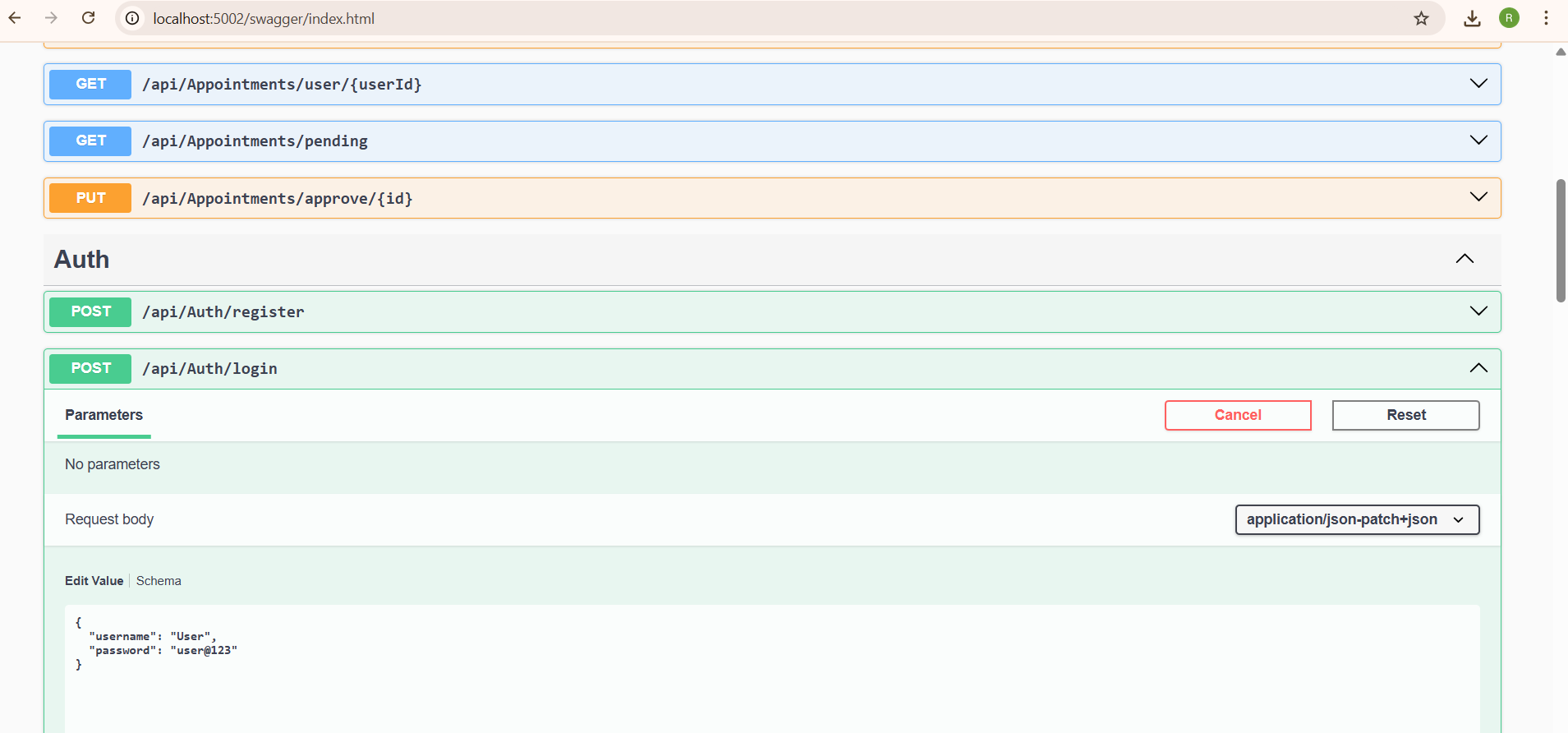


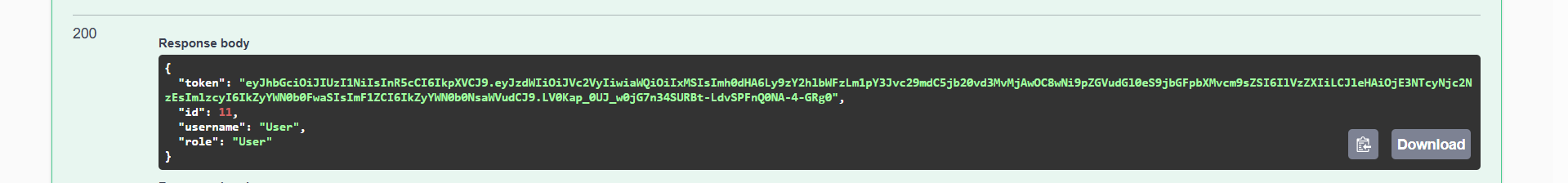
Backend Testing :-



End-to-End Testing :-

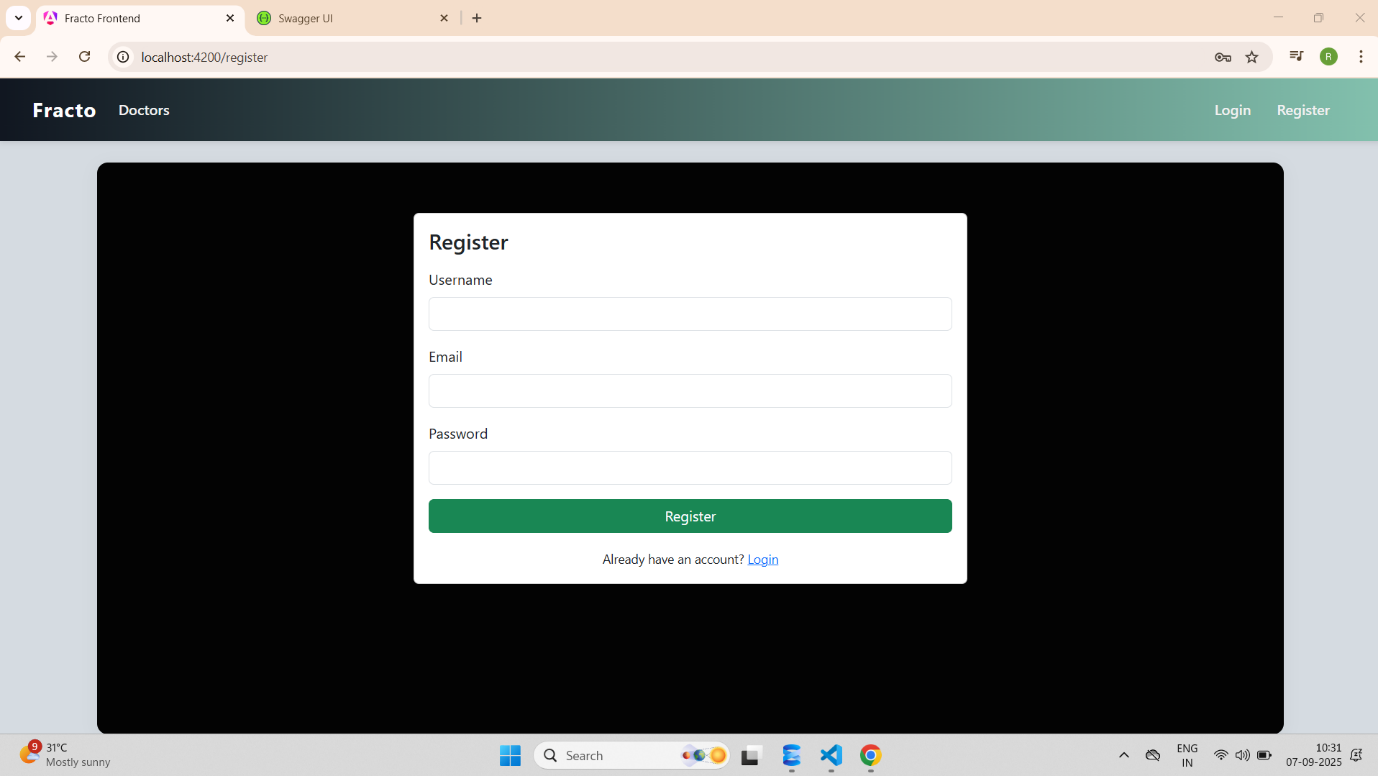
API’s Tested with Swagger UI :-



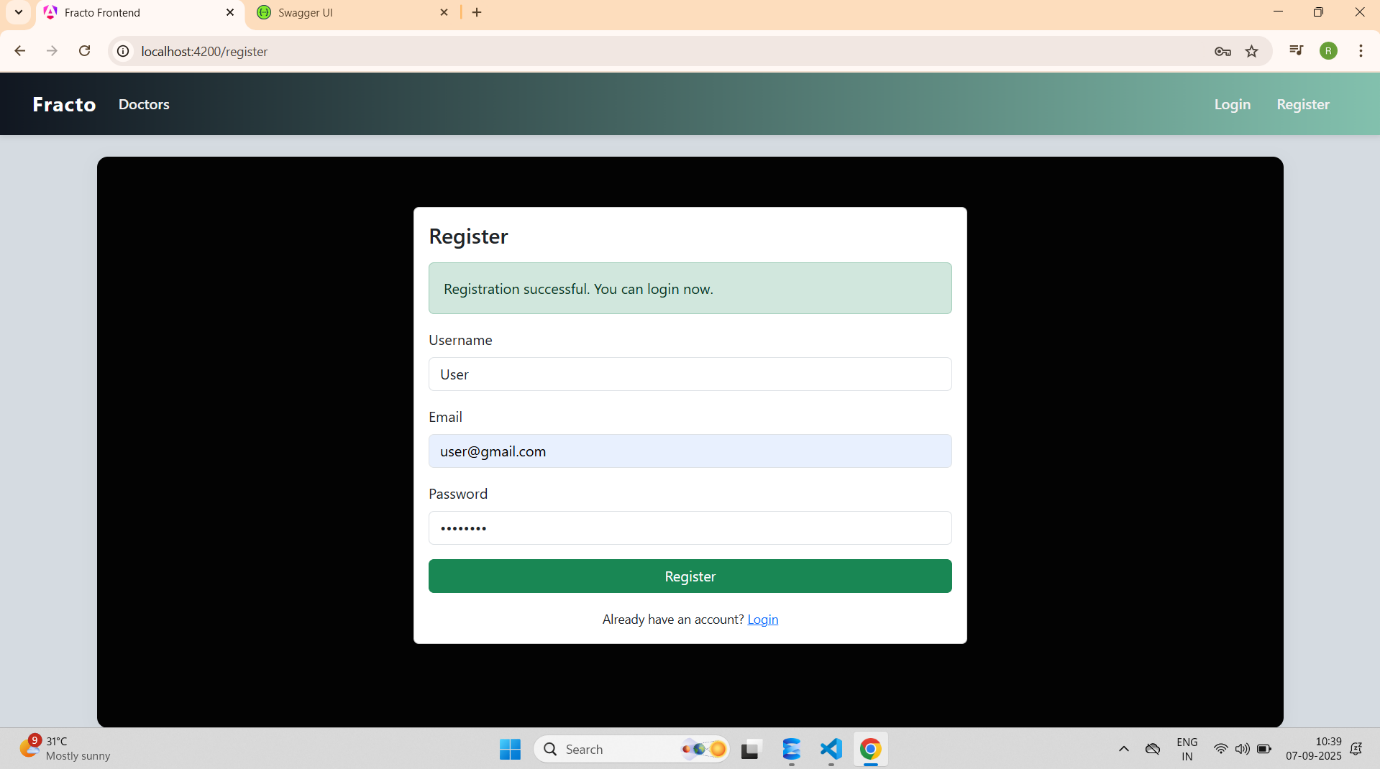


**Implementation Screenshots**

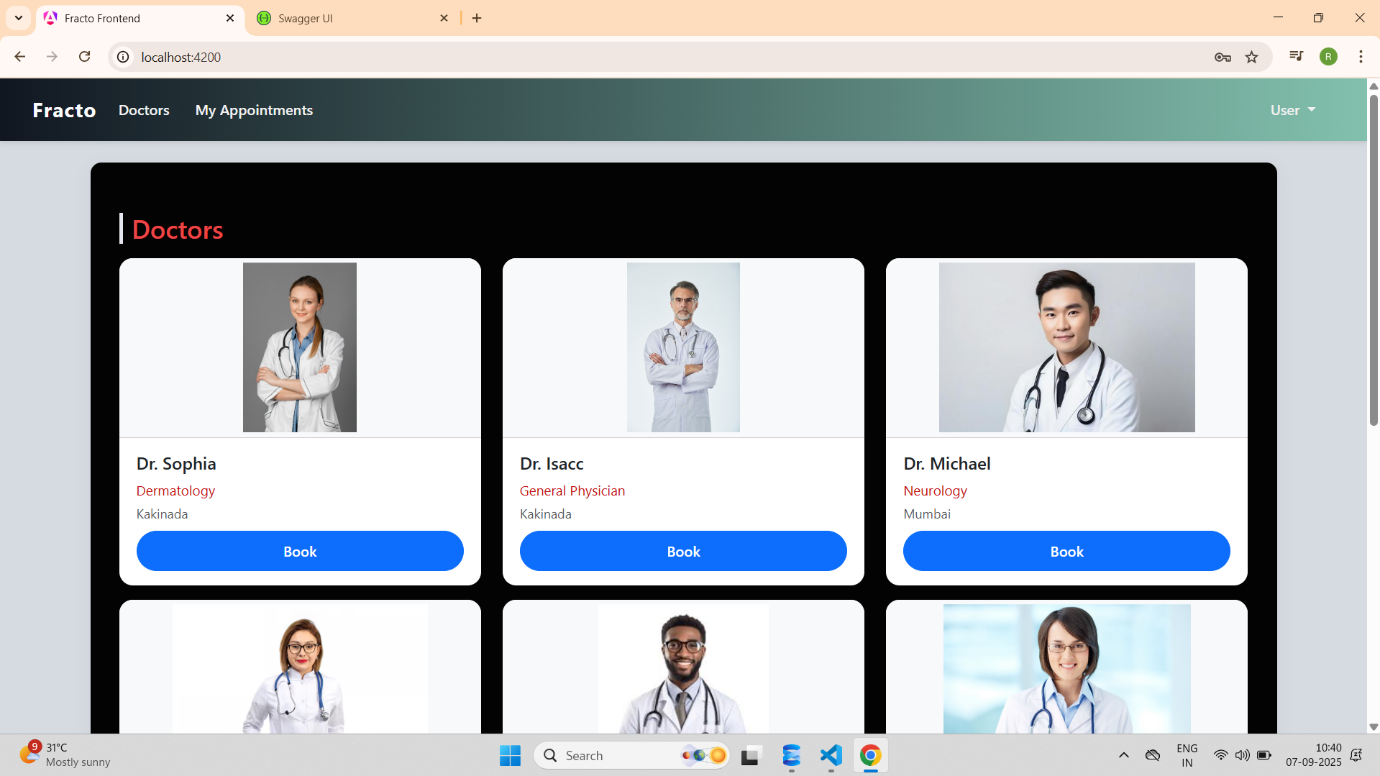
User Registration Page :-



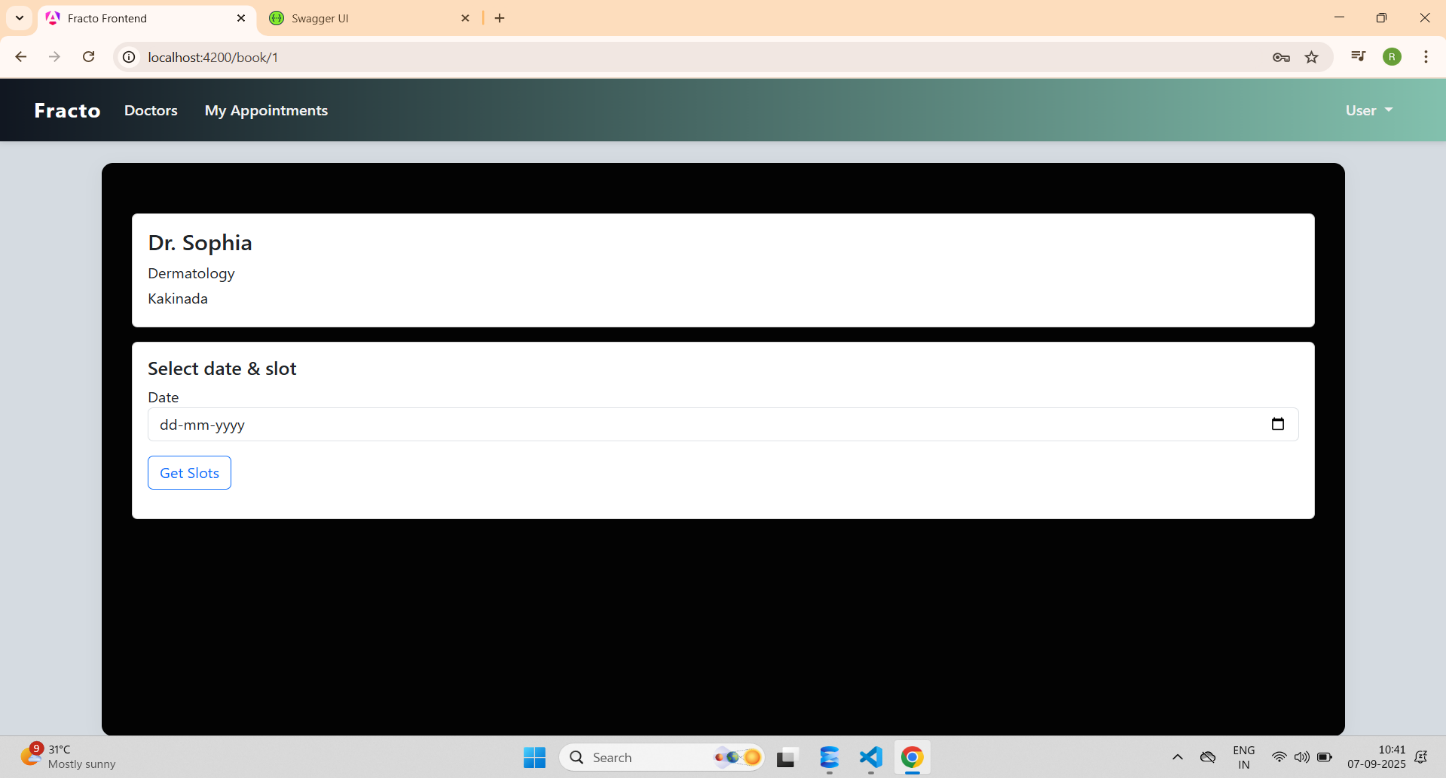
User Registration Successful :-



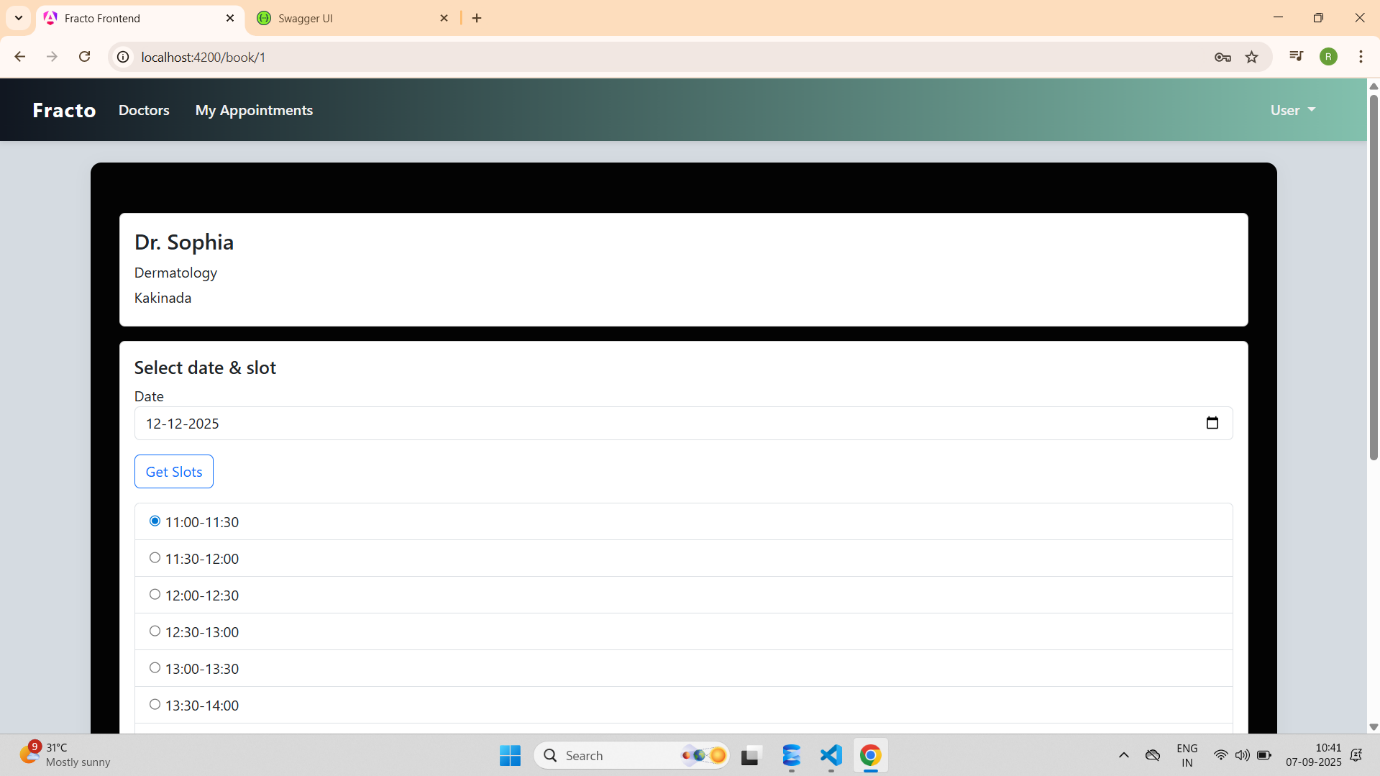
User Home Page :-

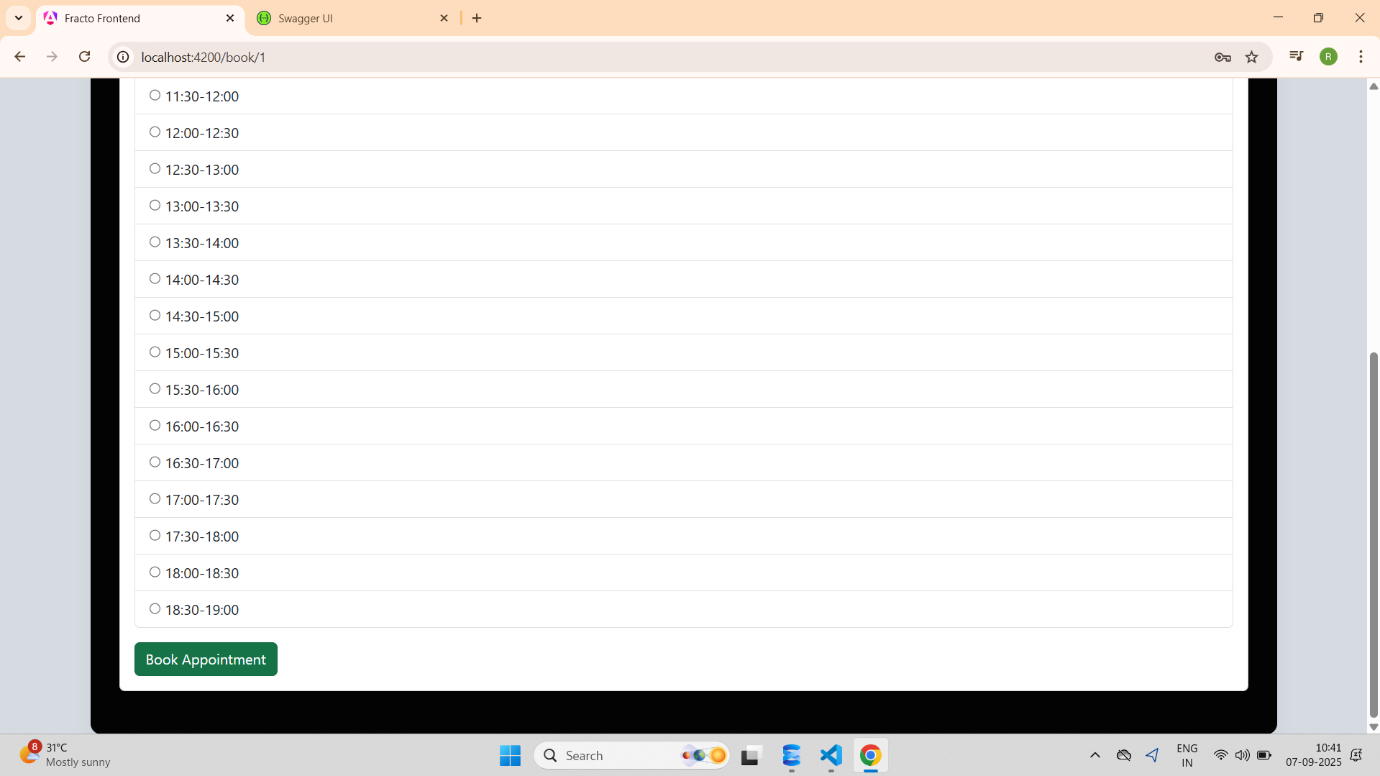


User Appointment Booking :-

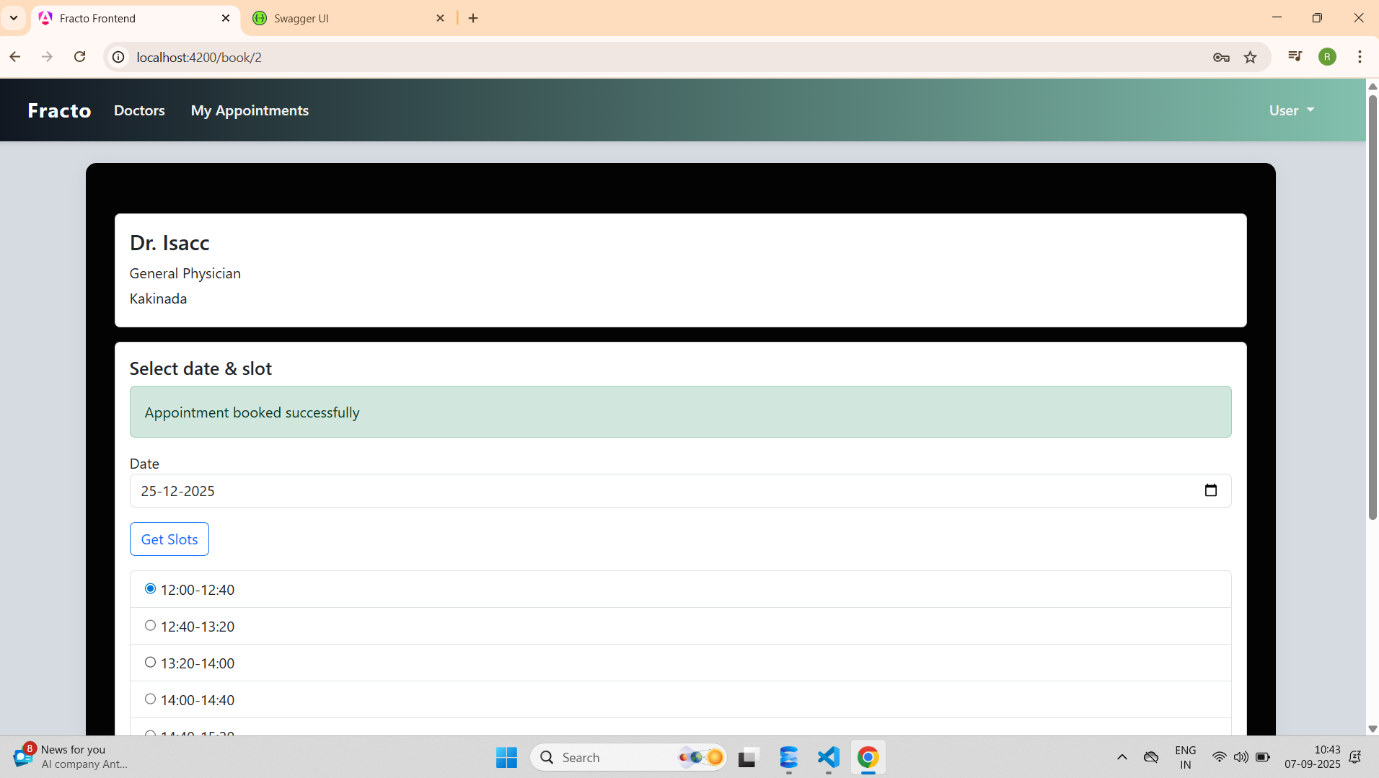


User can Select Specialization, City and Date & Slot :-

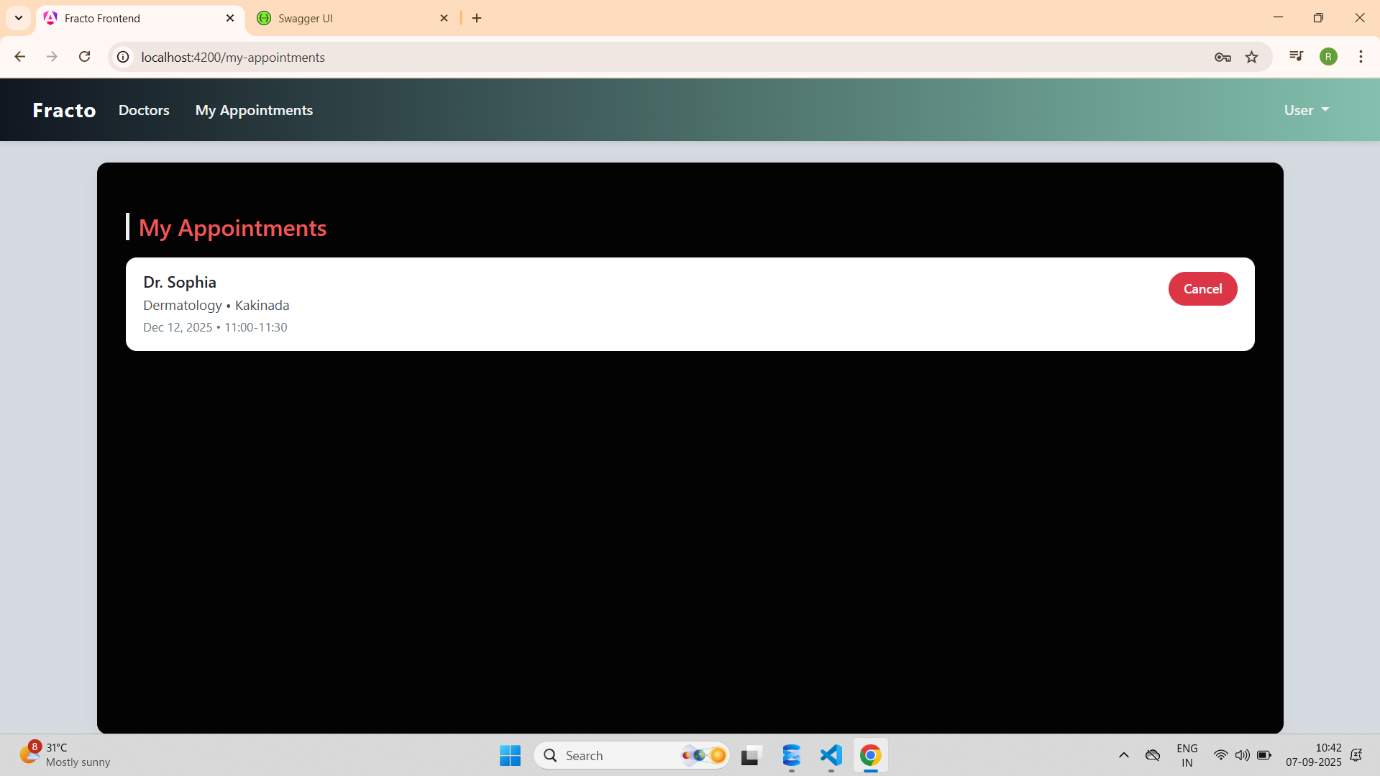




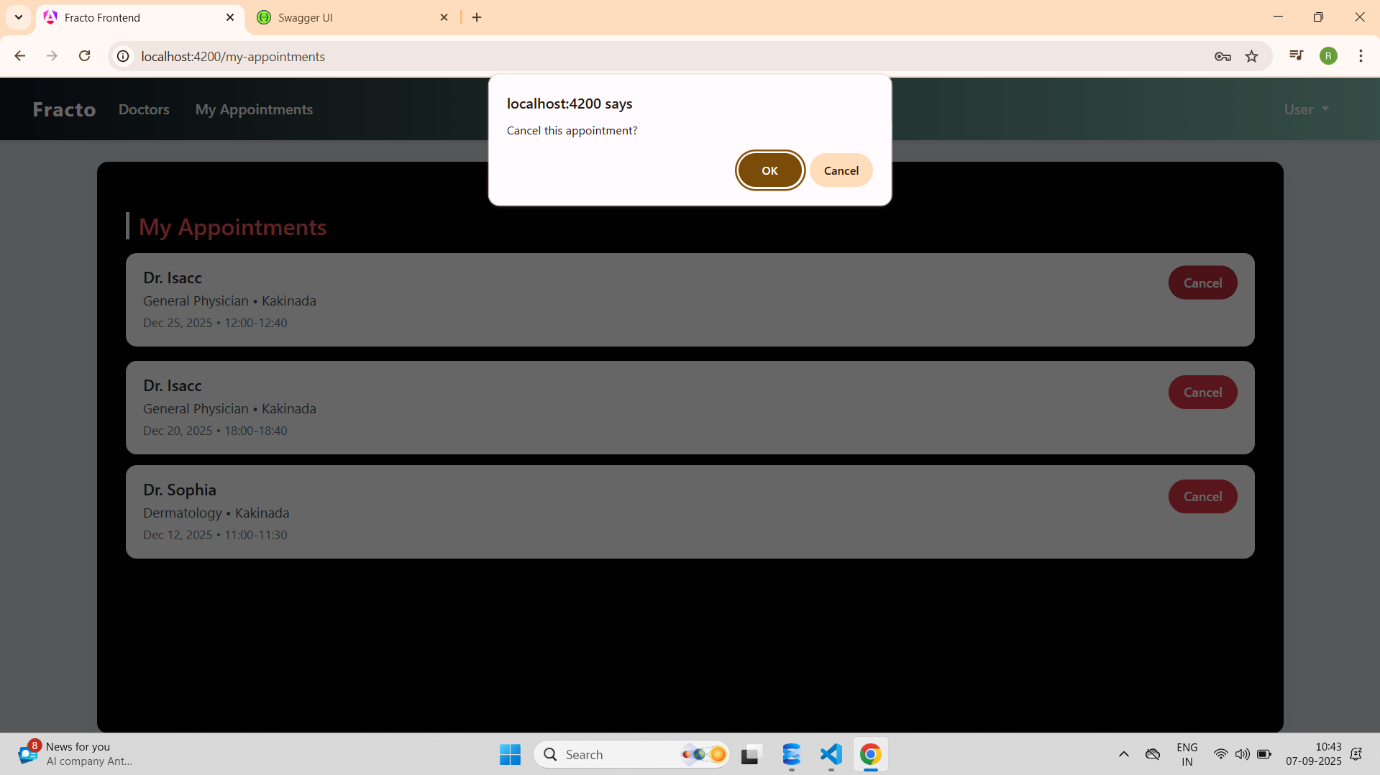
User Appointment Booked Successfully :-



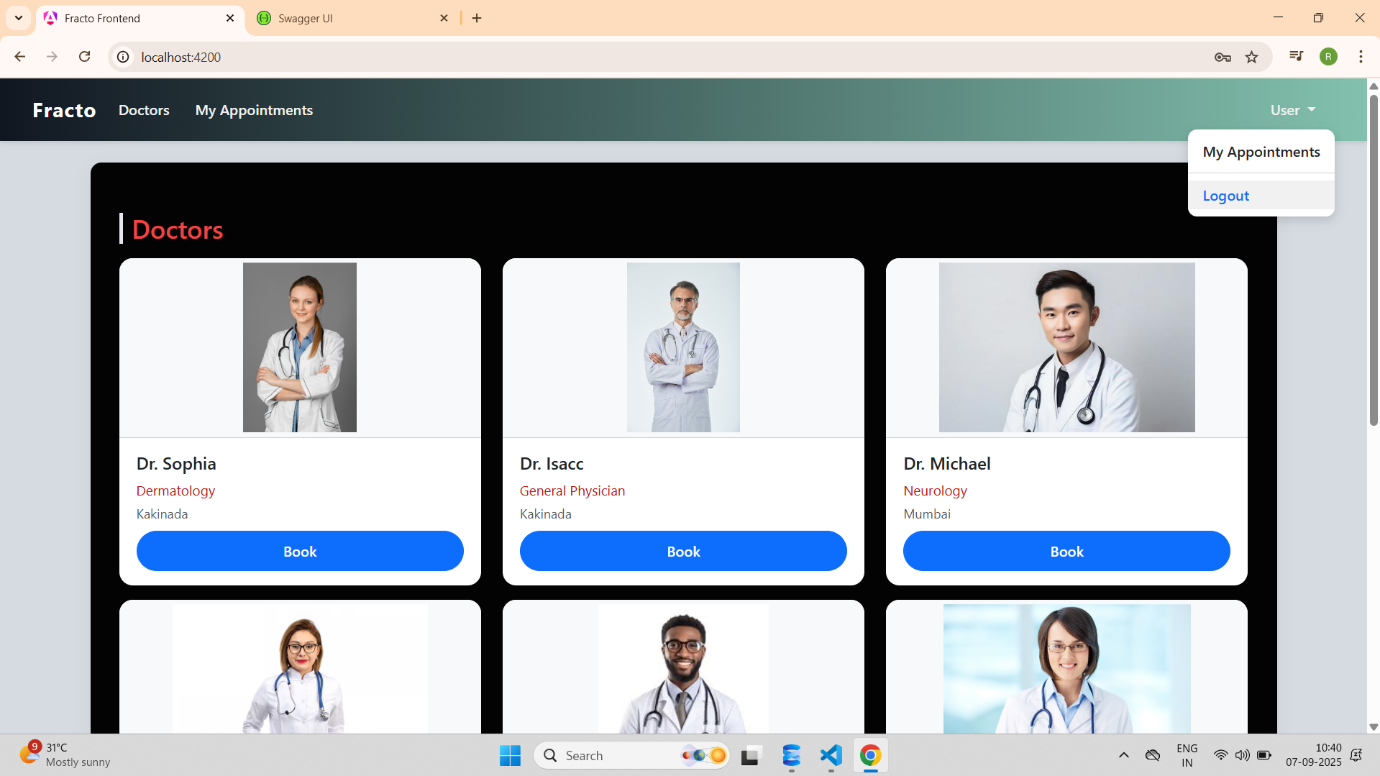
User Appointments Page after Booking :-



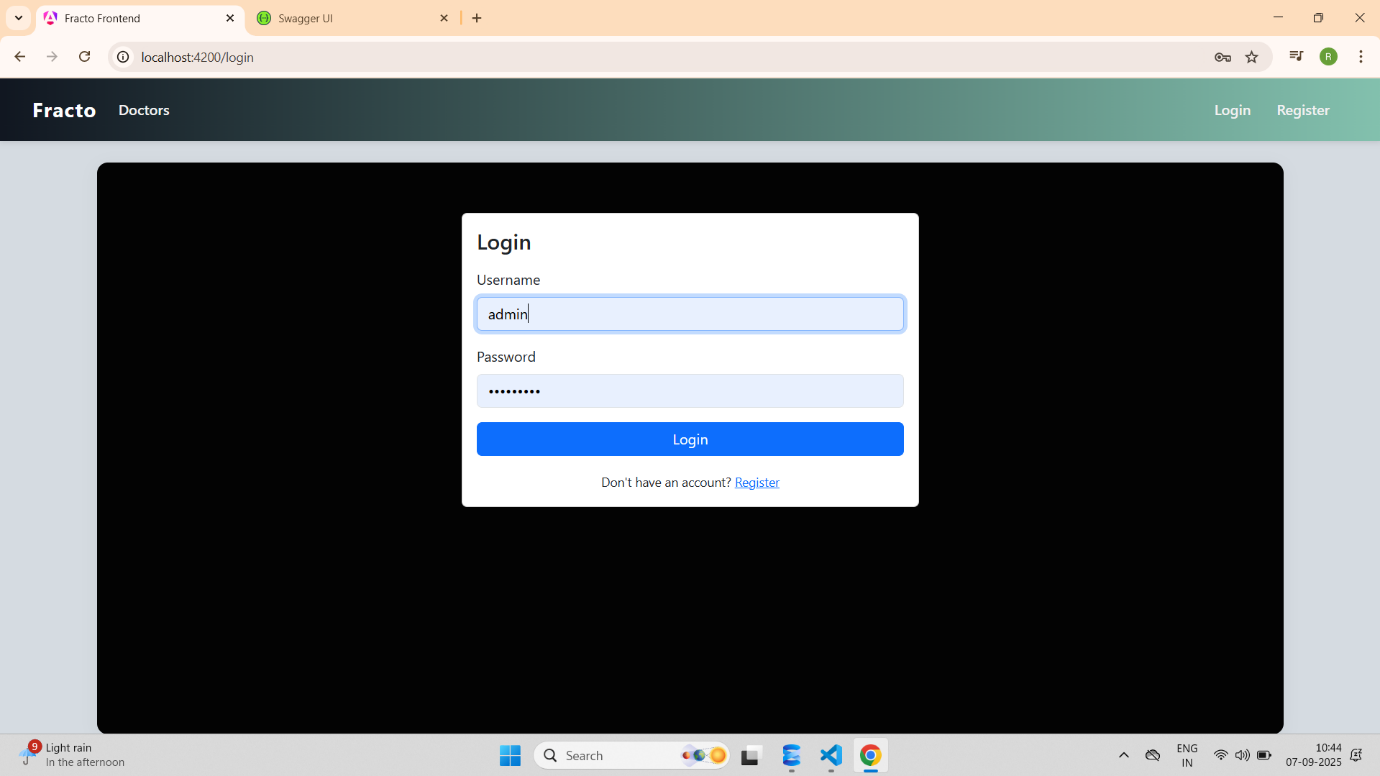
User Can Cancel the Appointments :-



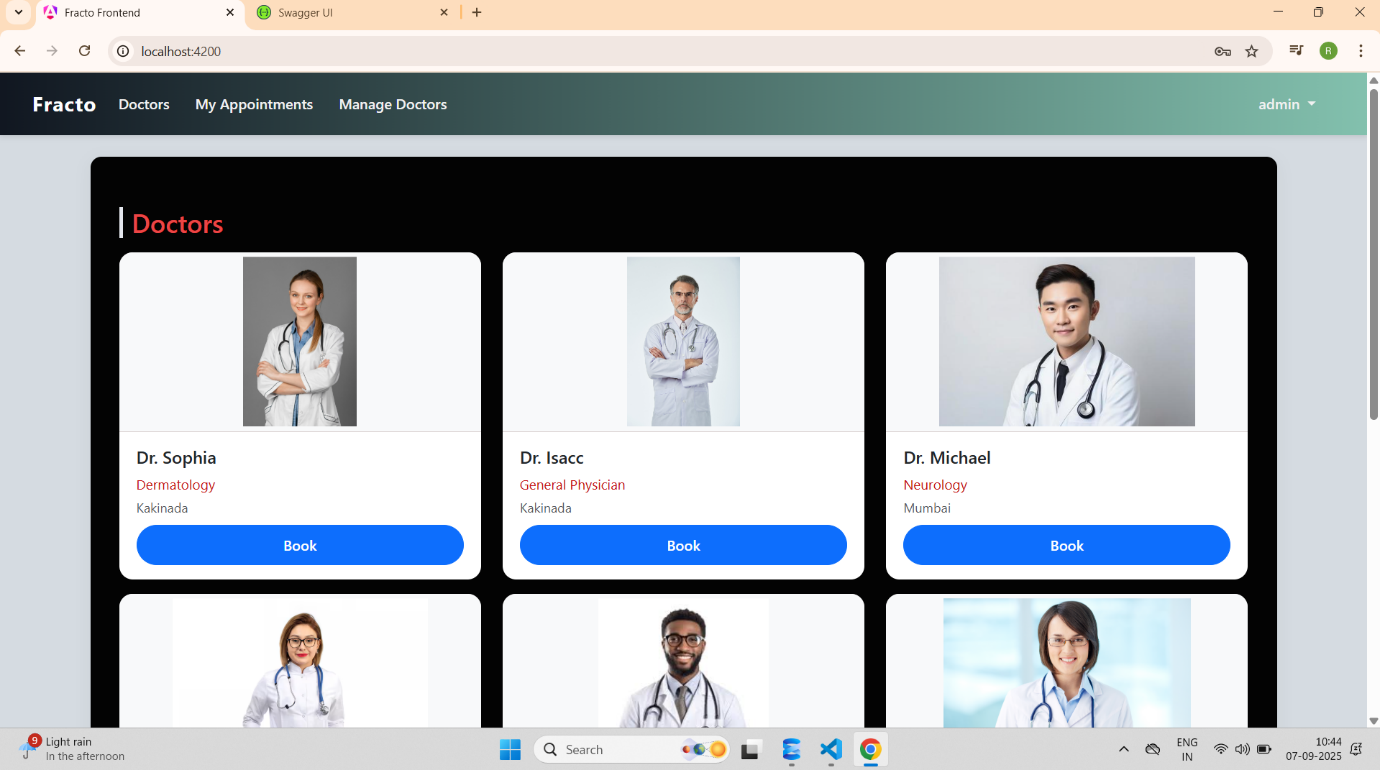
User can Logout : -



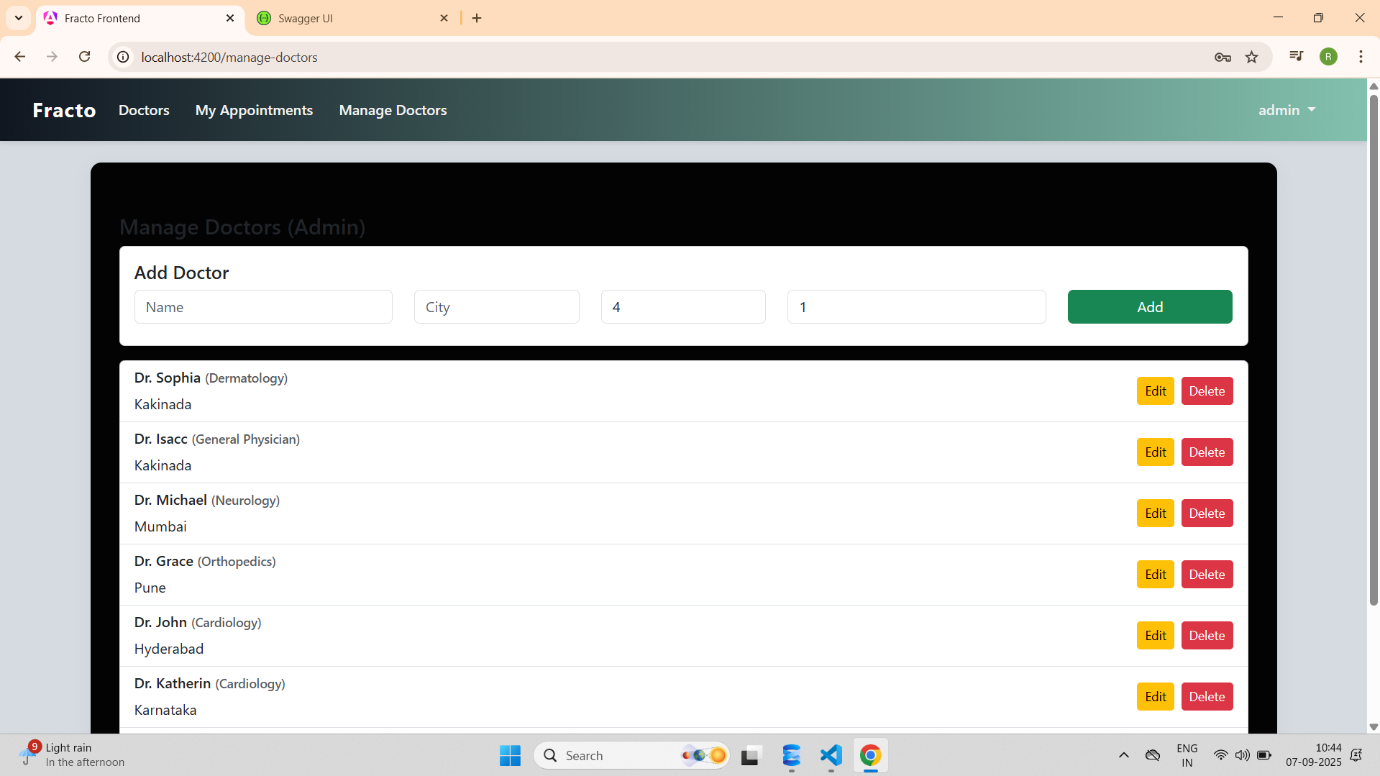
Admin Login Page :-



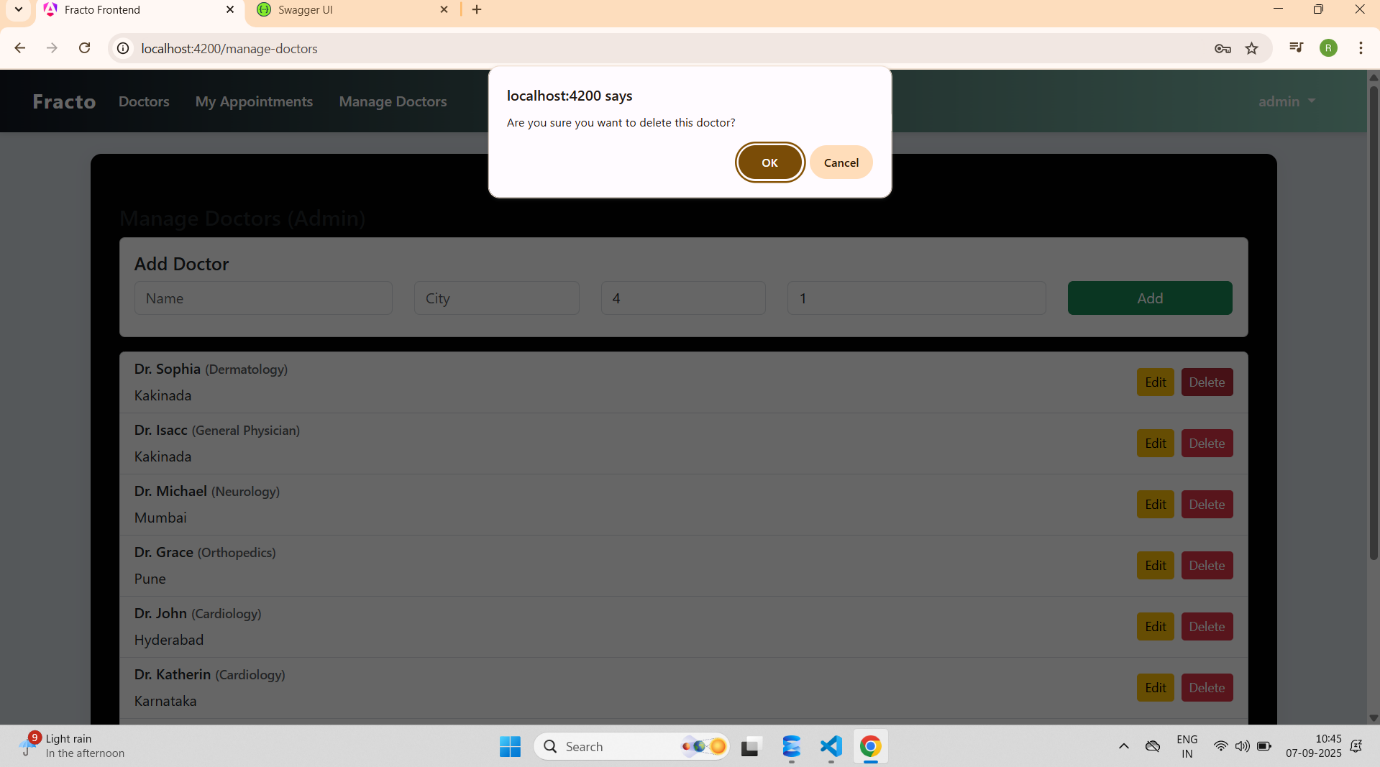
Admin Home Page :-

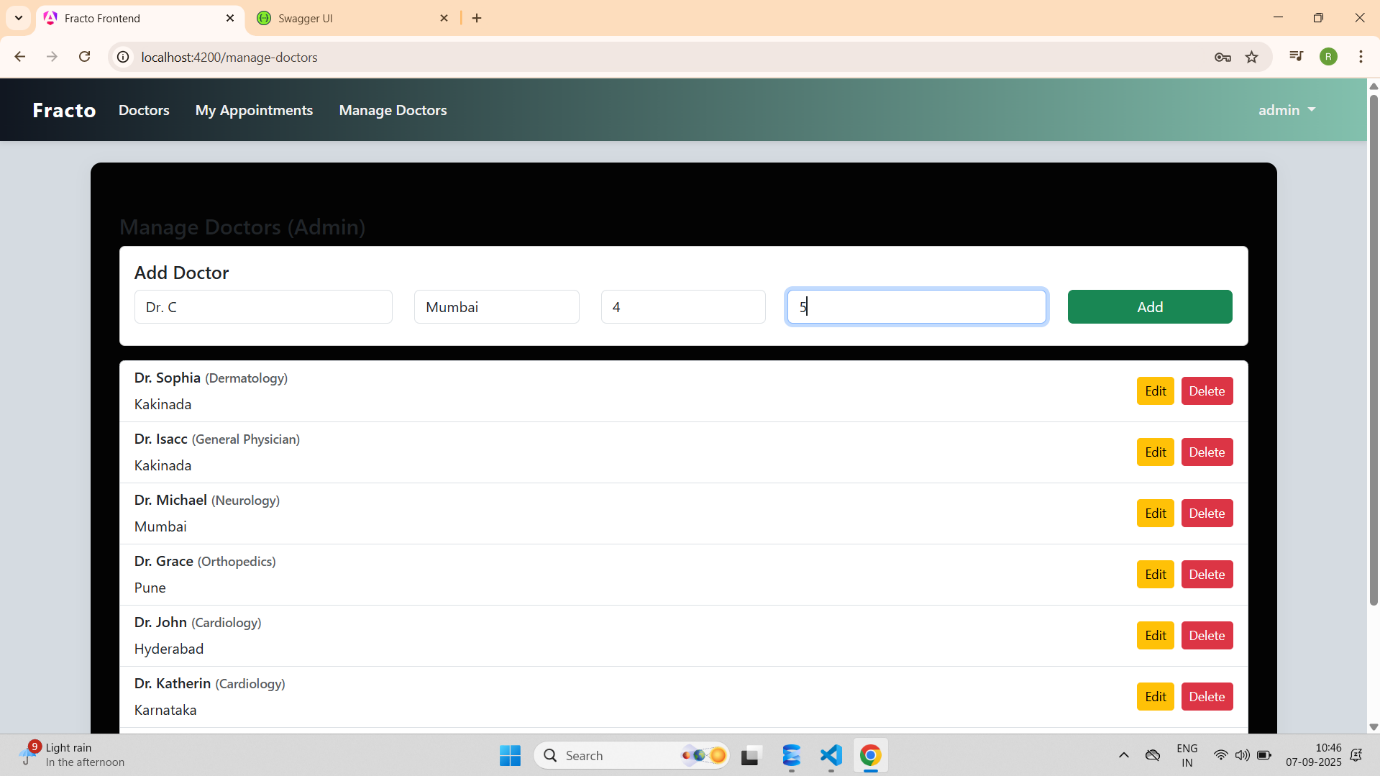


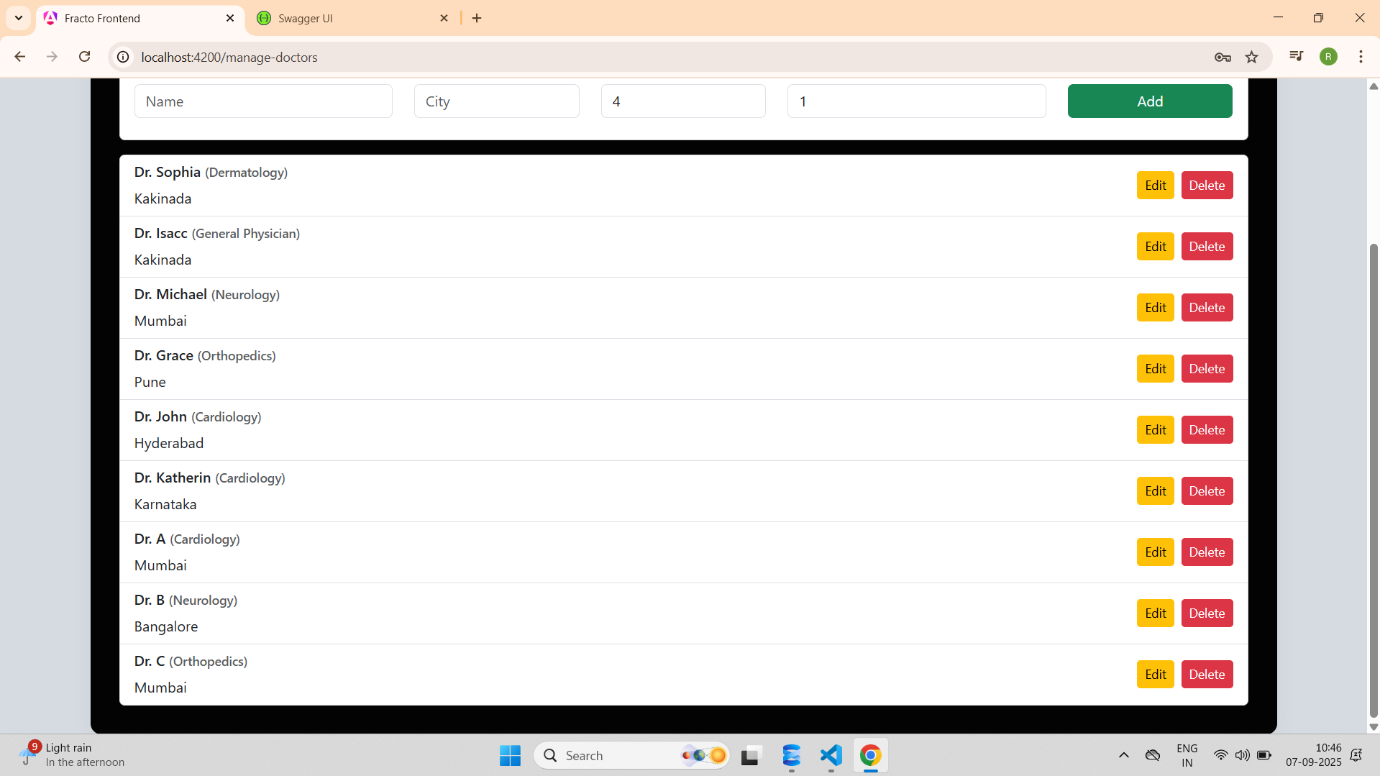
Admin Can Manage Doctors :-



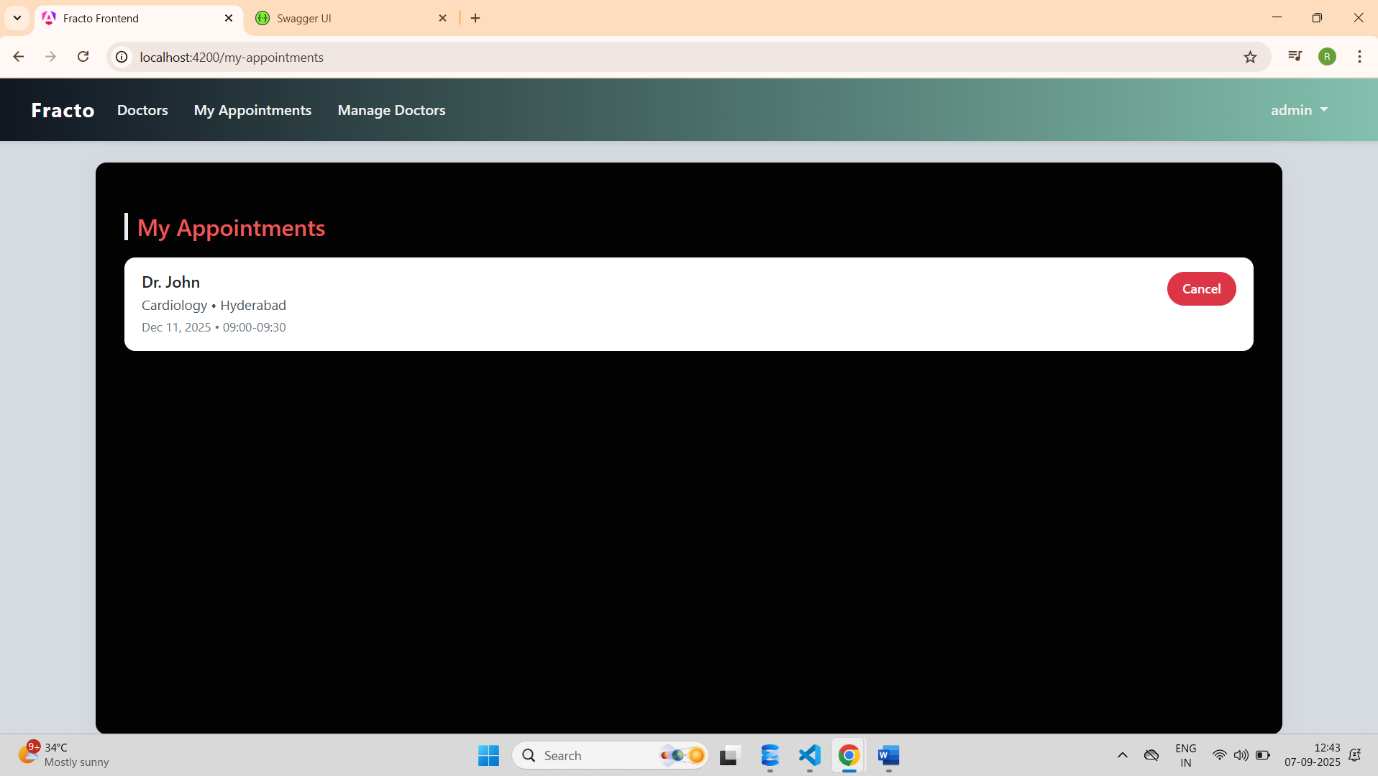
Admin can Create, Add and Delete Doctors



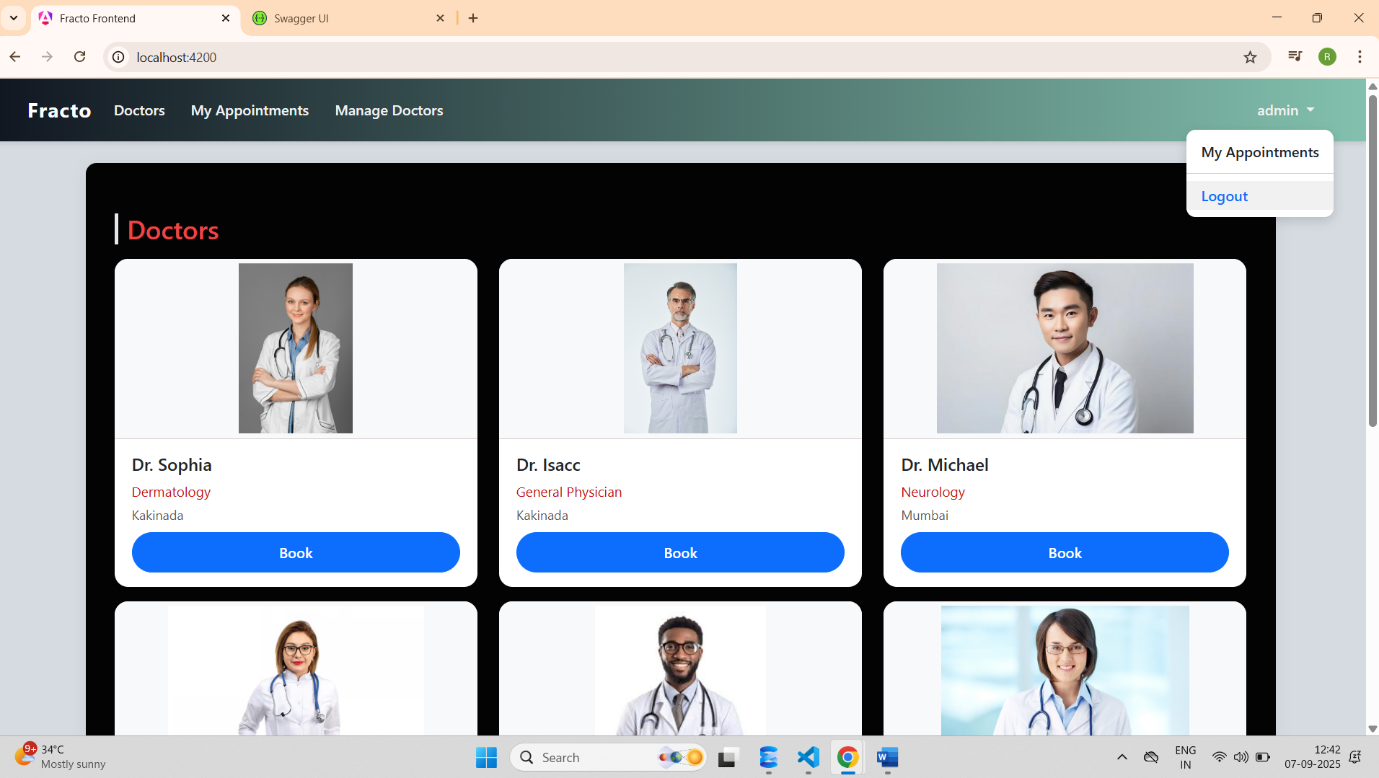




Admin can also Book,Cancel Doctor Appointments :-



Admin can Logout :-



**Conclusion :-**

The **Fracto project** was successfully developed as a complete online doctor appointment booking system.  
It achieved its goal of allowing patients to search, book, and cancel appointments with ease.  
Admins can efficiently manage users, doctors, and appointments through the platform.