**FRACTO**

Capstone Project Report

Submitted by : Maneesha M K

Batch : WIPRO NGA - .Net Full Stack Angular - FY26 - C3  
Instructor : Mr. Ramesh Nediyadath

Date : 08/09/2025

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| S.No. | Topic |
| 1 | INTRODUCTION |
|  | 1.1 Problem Definition |
|  | 1.2 Objectives |
|  |  |
| 2 | SYSTEM REQUIREMENTS |
|  |  |
| 3 | ARCHITECTURE OVERVIEW |
|  | 3.1 Frontend (Angular) |
|  | 3.2 Backend (ASP.NET Core) |
|  | 3.3 Integration |
|  |  |
| 4 | DATABASE SCHEMA |
|  | 4.1 Database Designs |
|  | 4.2 ER Diagrams |
|  |  |
| 5 | API ENDPOINTS |
|  | 5.1 AuthController |
|  | 5.2 DoctorsController |
|  | 5.3 RatingsController |
|  | 5.4 SpecializationsController |
|  | 5.6 UsersController |
|  |  |
| 6 | TESTING AND SECURITY |
|  | 6.1 Manual Testing |
|  | 6.2 Unit Testing |
|  | 6.3 Integration Testing |
|  | 6.4 Security |
|  |  |
| 7 | OUTPUT SCREENSHOTS |
|  |  |
| 8 | CONCLUSION & FUTURE WORK |
|  | 8.1 Conclusion |
|  | 8.2 Future Work |
|  |  |

**ABSTRACT**

Fracto is a comprehensive, full-stack online doctor appointment booking system designed to streamline the process of scheduling medical consultations. The application features a dual-portal architecture, with a user-facing Angular frontend for patients to search for doctors by city, specialization, and availability, select time slots, and manage their appointments, and an administrative interface for managing users, doctors, and appointments. The backend is built on [ASP.NET](https://asp.net/) Core MVC, providing a secure RESTful API with JWT authentication, EF Core for database management with SQL Server, and server-side handling of profile image uploads. The project's scope also includes implementing a doctor rating system, API documentation with Swagger, and optional real-time notification features using Signal to ensure a seamless and interactive user experience.

**INTRODUCTION**

**1.1 Problem Definition:**

The traditional process of booking doctor appointments is inefficient, relying on phone calls and manual scheduling, which creates frustration for patients and administrative burden for clinics.

**1.2 Objectives:**

* Develop a secure, dual-role platform for patients and admins.
* Enable patients to easily search, book, and cancel appointments online.
* Provide advanced search filters for doctors by specialization, location, and ratings
* Implement a secure authentication system and an admin dashboard for full CRUD operations.
* Ensure scalability using modern frameworks like Angular and ASP.NET Core.

**2. SYSTEM REQUIREMENTS**

● Frontend: Angular-based UI for user interaction.

● Backend: ASP.NET Core MVC with Web API for business logic and data access.

● ORM: Entity Framework Core (EF Core) for database interactions.

● Database: SQL Server for storing user, doctor, appointment, and rating data.

● Image Uploads: Store uploaded profile images in a server-side folder.

● Notification: Real-time notifications using SignalR (optional).

**ARCHITECTURE OVERVIEW**

**3.1 Frontend (Angular) :**

Framework: Angular

State Management: Context API / NgRx

HTTP Client: Angular HttpClient for API call

Routing: Angular Router for navigation and role-based access (User/Admin)

Components: Login, booking, doctor search, admin dashboard

**3.2 Backend (ASP.NET Core):**

Framework: ASP.NET Core MVC + Web API

Authentication: JWT Tokens

Database Access: Entity Framework Core (EF Core)

API Docs: Swagger

File Upload: Handle via IFormFile, store paths in DB

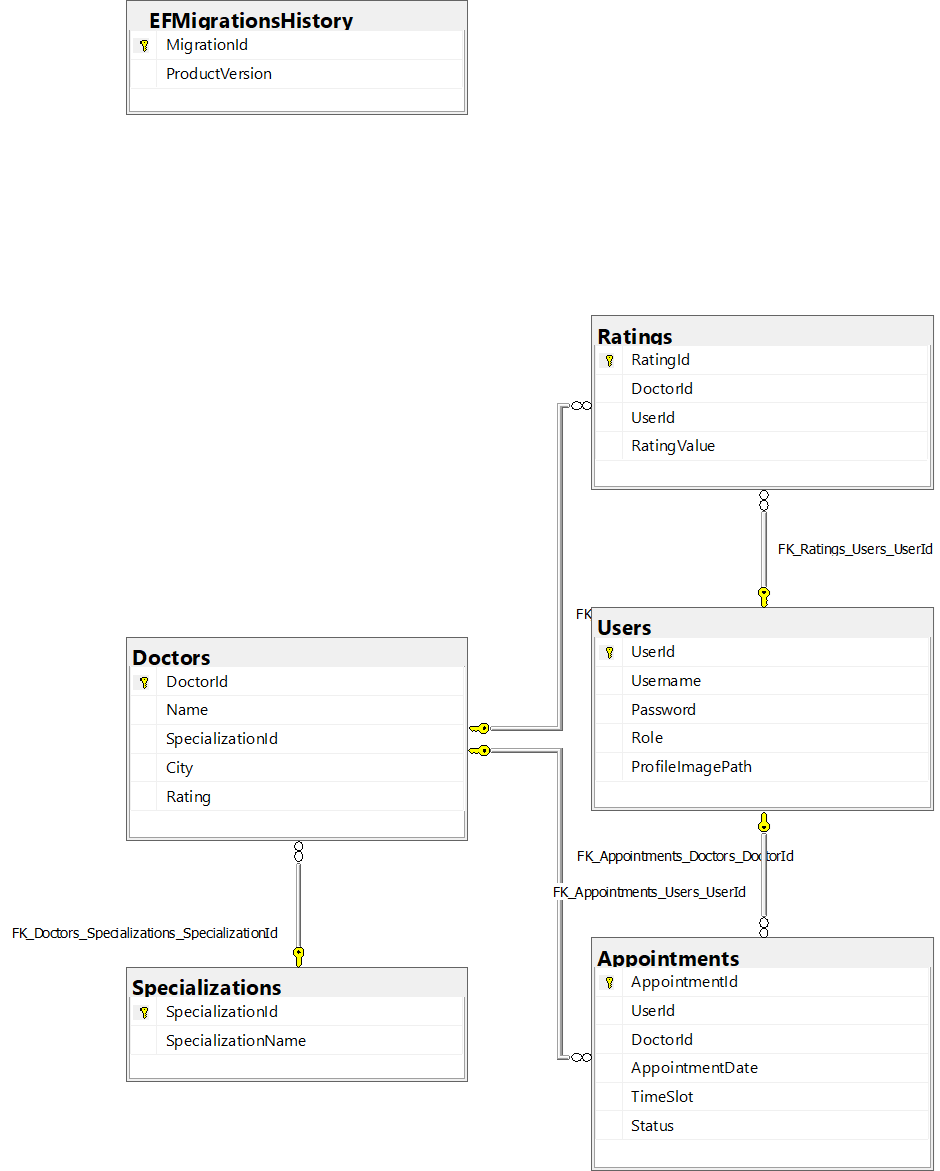
**3.3 Integration:**

Angular frontend consumes RESTful APIs from ASP.NET Core

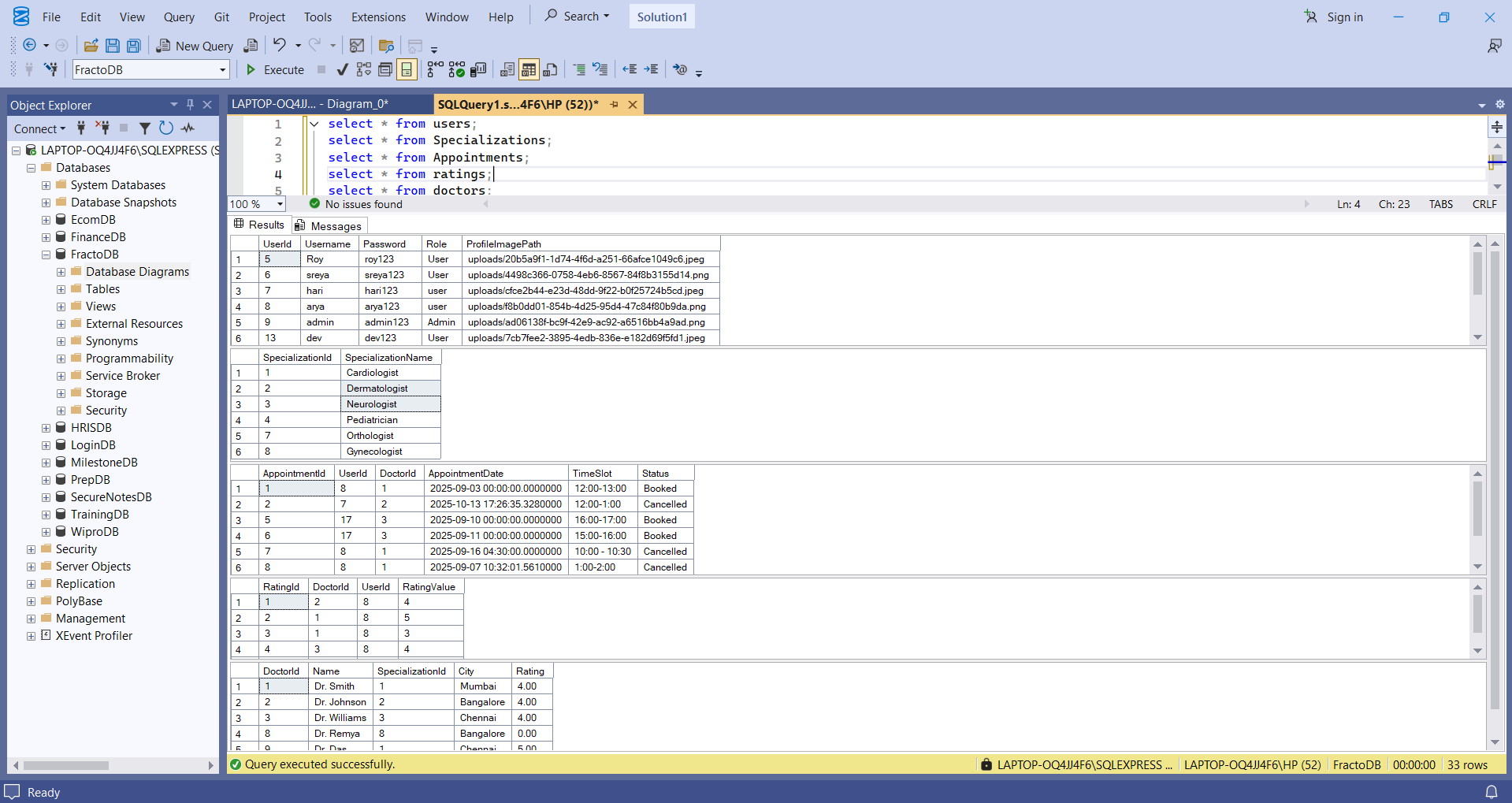
JWT secures communication between frontend and backend

DB managed via EF Core; images stored on server, paths in DB

1. **DATABASE SCHEMA**

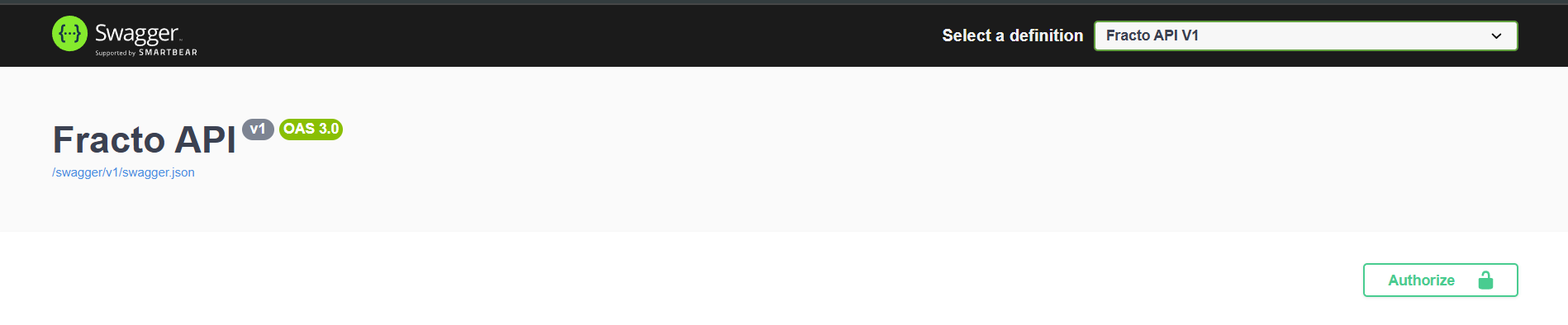
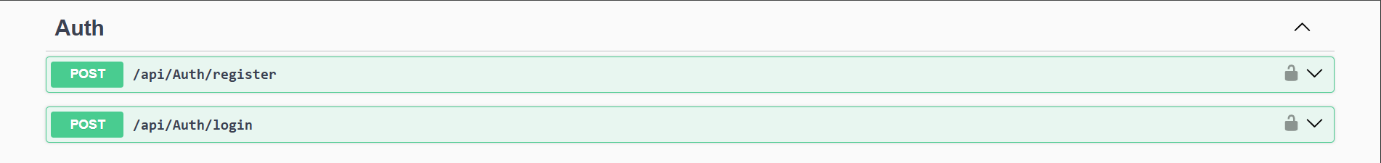
**4.1 Database Design**

**4.2 Tables**

****

**5 . API ENDPOINTS**   
 **5.1 AuthController :**

* **POST /api/Auth/register**  
  Registers a new user. Accepts user details such as username, email, and password. Returns a success message or error if the user already exists.
* **POST /api/Auth/login**  
  Authenticates a user using email and password. Returns a JWT token for accessing protected endpoints.

**5.2 DoctorsController :**

* **GET /api/Doctors**

Retrieves a list of all doctors.

* **POST /api/Doctors**

Creates a new doctor entry (Admin only).

* **GET /api/Doctors/{id}**

Retrieves details of a specific doctor by ID.

* **PUT /api/Doctors/{id}**

Updates a doctor’s information.

* **DELETE /api/Doctors/{id}**

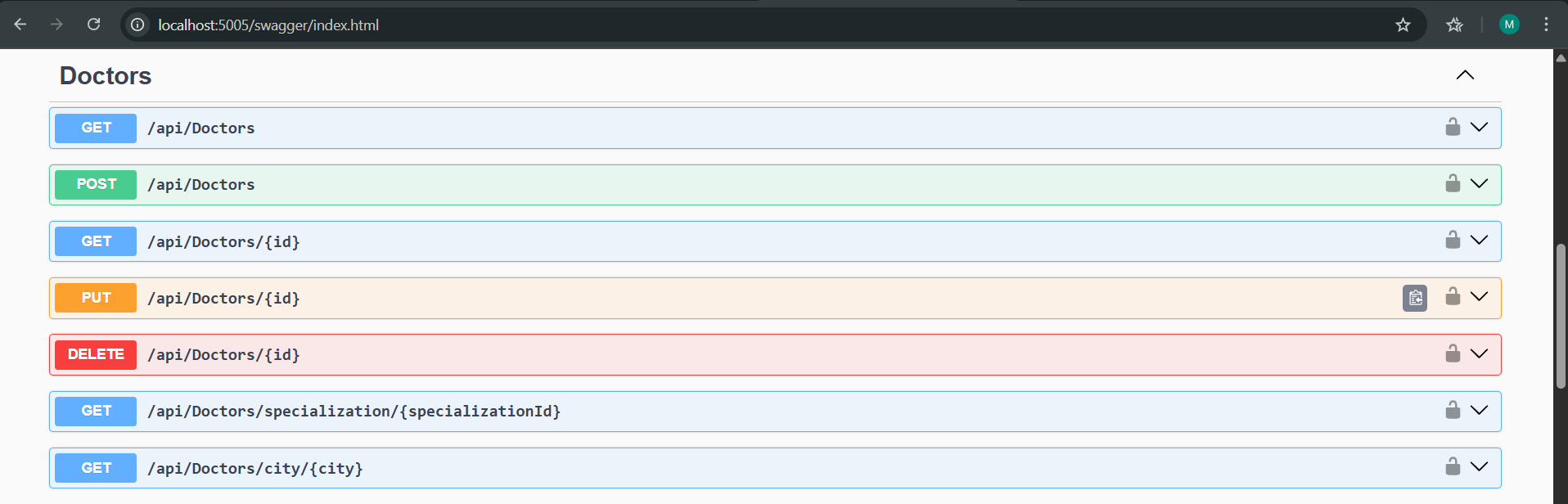
Deletes a doctor by ID (Admin only).

* **GET /api/Doctors/specialization/{specializationId}**

Retrieves doctors by specialization ID.

* **GET /api/Doctors/city/{city}**

Retrieves doctors by city name.



**5.3 RatingsController :**

* **GET /api/Ratings**

Retrieves all ratings.

* **POST /api/Ratings**

Submits a new rating for a doctor.

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

Retrieves all ratings for a specific doctor.

* **GET /api/Ratings/my**

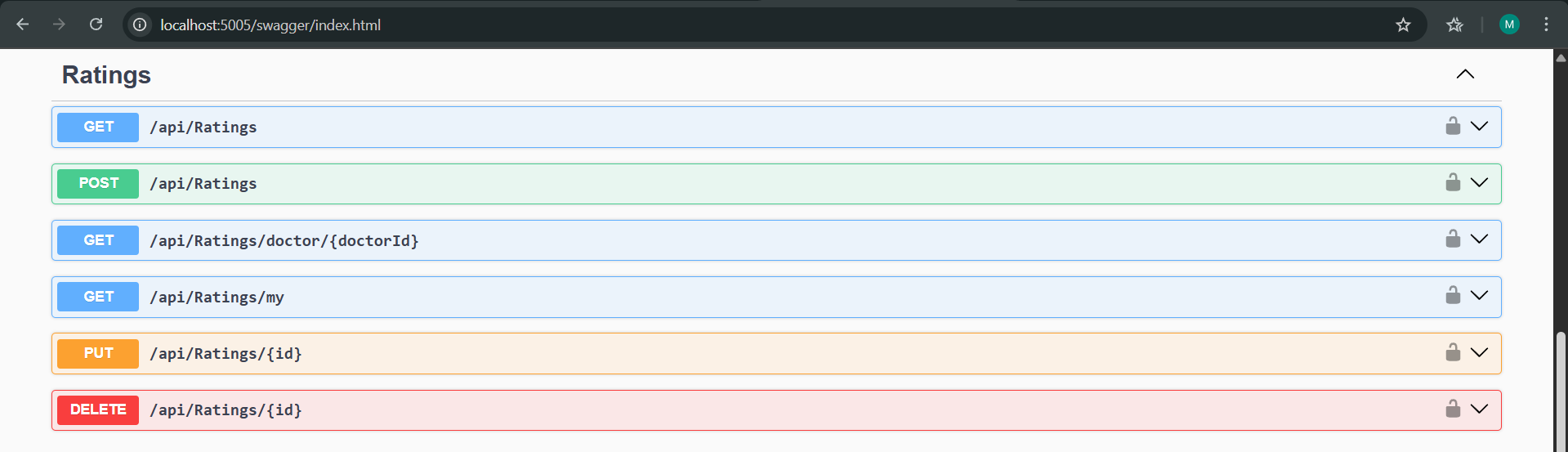
Retrieves ratings submitted by the currently authenticated user.

* **PUT /api/Ratings/{id}**

Updates a specific rating by ID.

* **DELETE /api/Ratings/{id}**

Deletes a rating by ID.



**5.4 SpecializationsController :**

* **GET /api/Specializations**

Retrieves all specializations.

* **POST /api/Specializations**

Creates a new specialization (Admin only).

* **GET /api/Specializations/{id}**

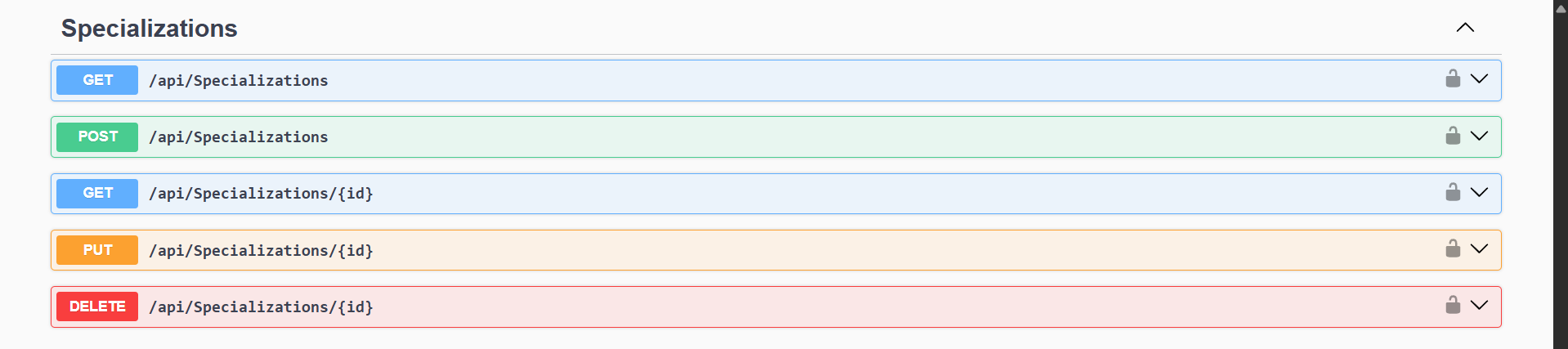
Retrieves a specific specialization by ID.

* **PUT /api/Specializations/{id}**

Updates a specialization.

* **DELETE /api/Specializations/{id}**

Deletes a specialization by ID (Admin only).



**5.5 UsersController :**

* **GET /api/Users**

Retrieves a list of all users (Admin only).

* **POST /api/Users**

Creates a new user (Admin only).

* **GET /api/Users/{id}**

Retrieves details of a specific user by ID.

* **PUT /api/Users/{id}**

Updates a user’s information.

* **DELETE /api/Users/{id}**

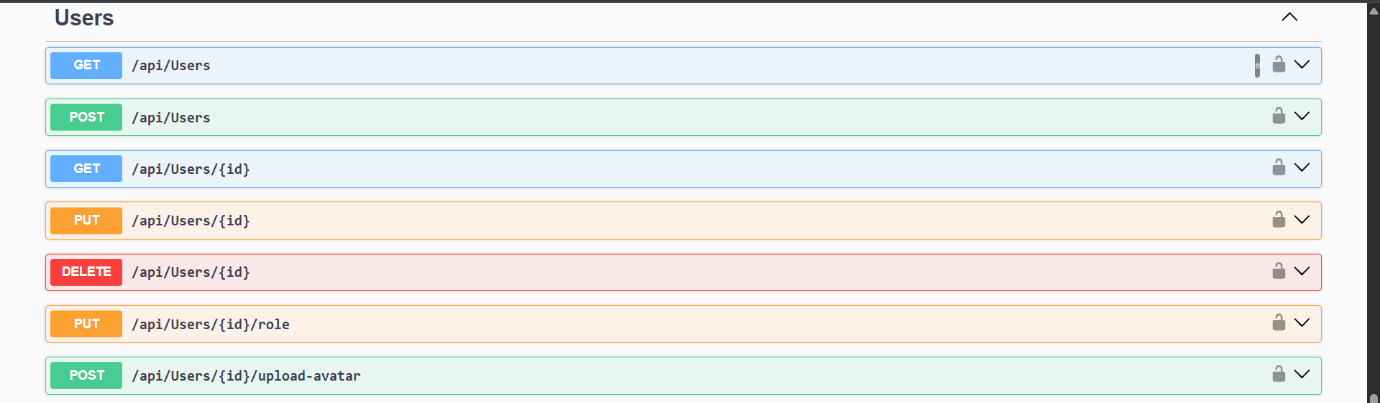
Deletes a user by ID (Admin only).

* **PUT /api/Users/{id}/role**

Updates a user’s role (Admin only).

* **POST /api/Users/{id}/upload-avatar**

Uploads a profile avatar for a user.



**6.Testing & Security**

**6.1 Manual Testing:**

All API endpoints (Auth, Users, Doctors, Appointments, Ratings, Specializations) were tested via Postman and Swagger UI. Verified inputs, error handling, status codes, and role-based access control using JWT tokens.

**6.2 Unit Testing**:

Core business logic and controller methods (e.g., appointment booking, rating calculation) were tested using xUnit to isolate and validate functionality.

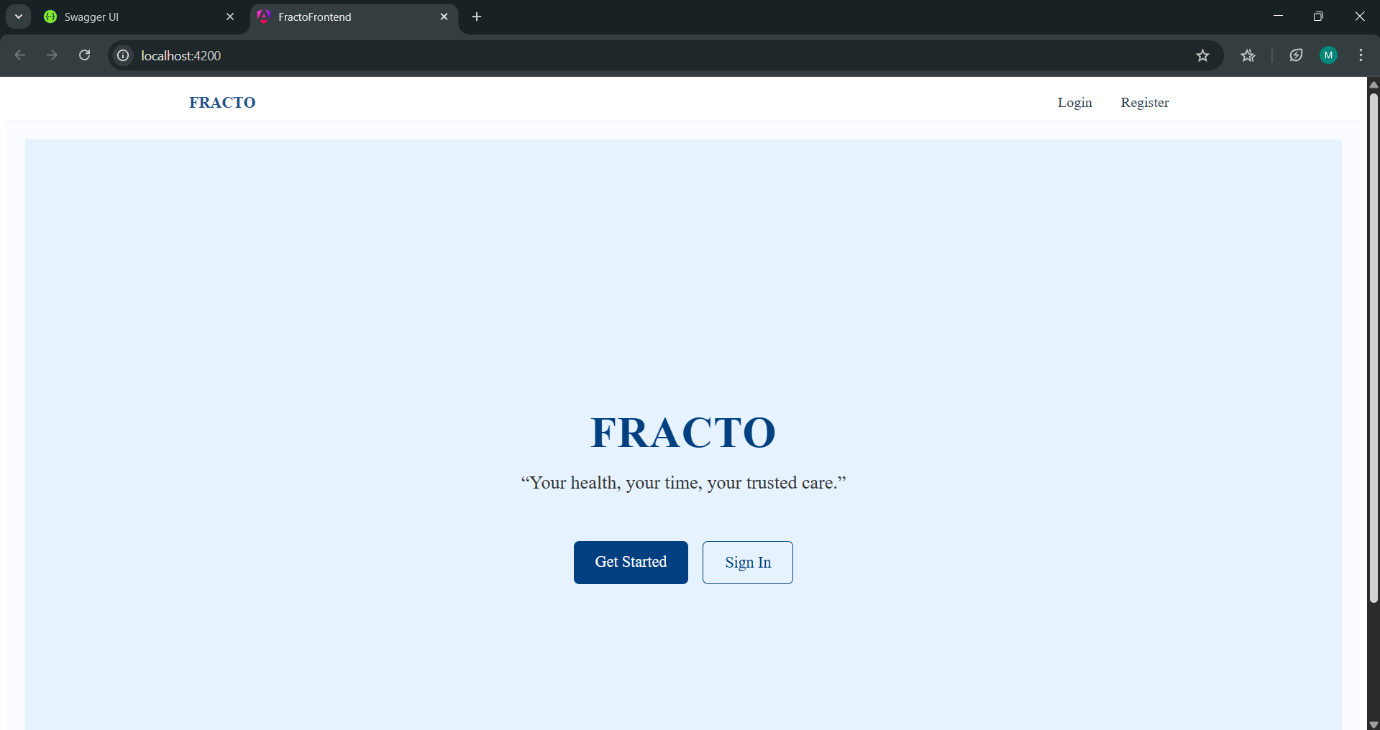
**6.3 Integration Testing**:

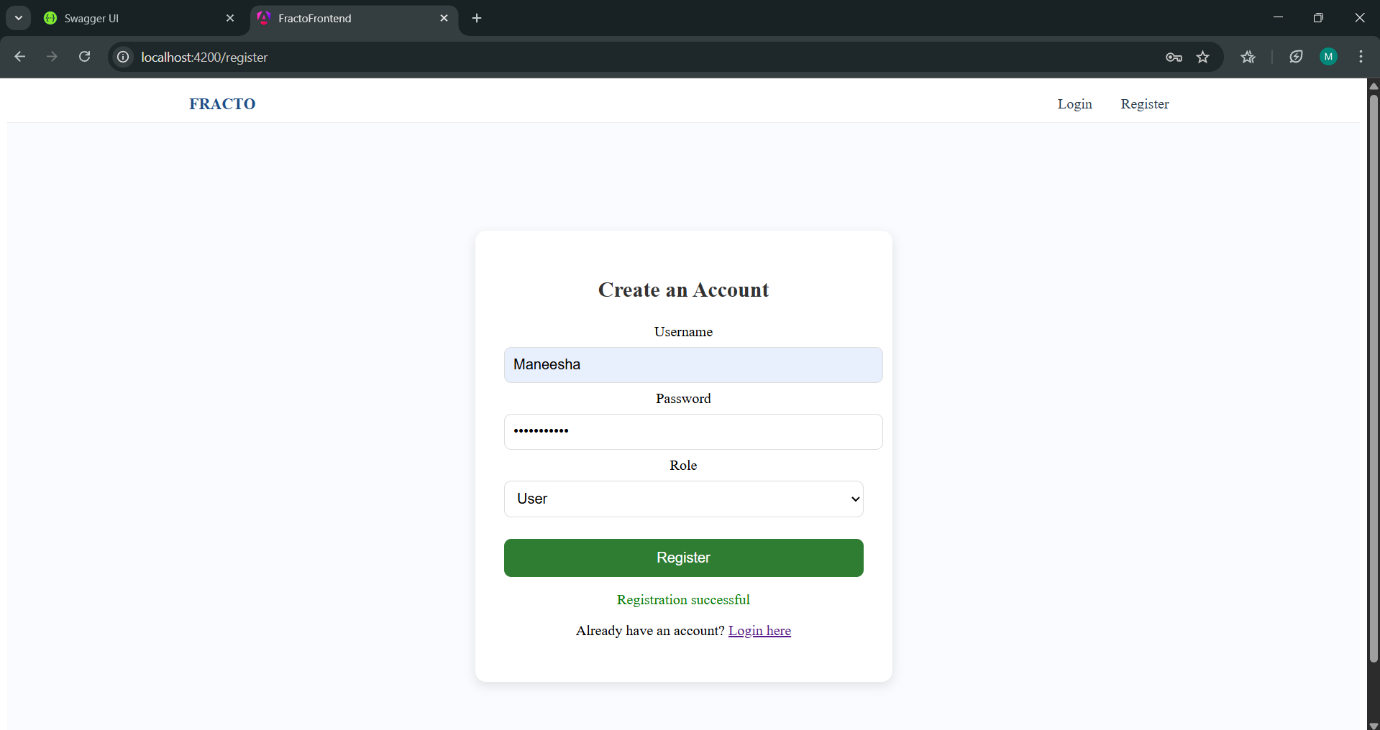
End-to-end workflows (e.g., register → login → book appointment → rate doctor) were tested to ensure system cohesion and accurate database updates.

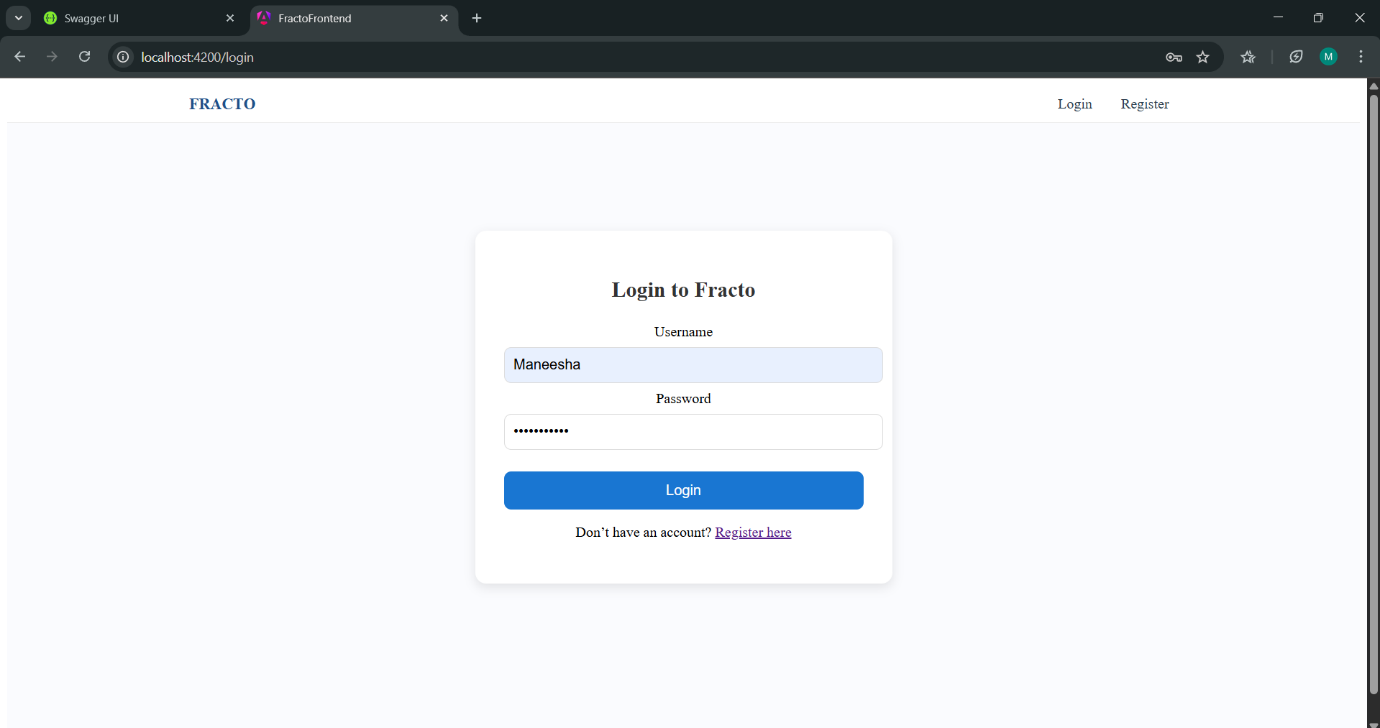
**6.4 Security**:

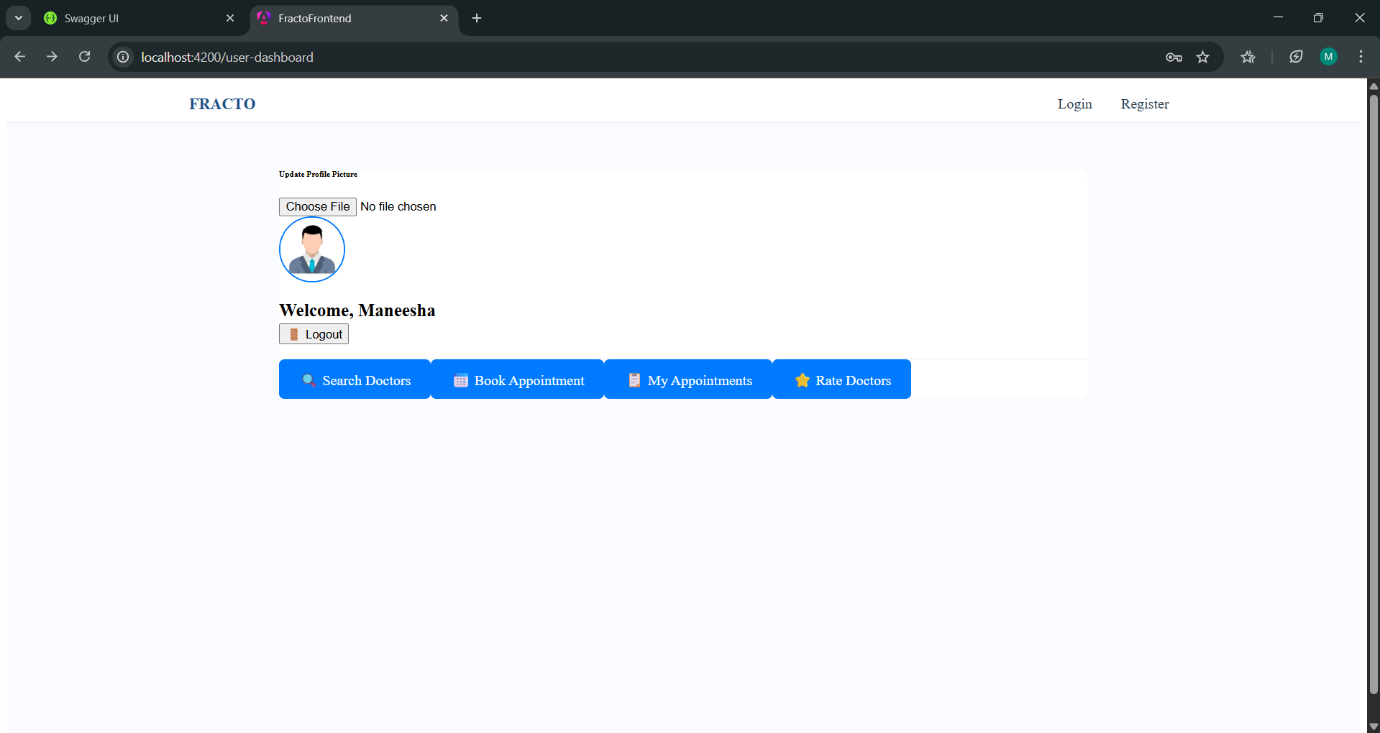
Implemented JWT authentication and role-based authorization (User/Admin). Validated and sanitized all inputs. Secured file uploads for user avatars.

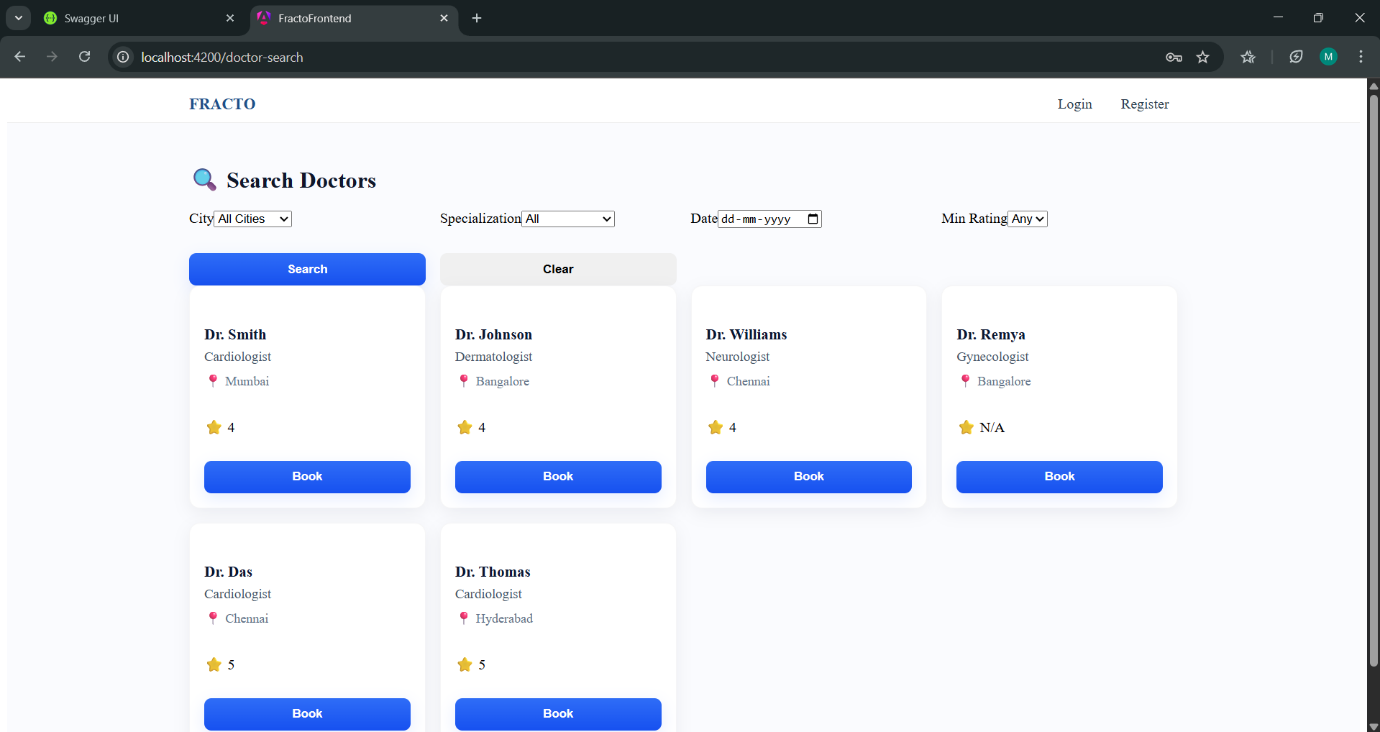
**7. OUTPUT SCREENSHOTS**

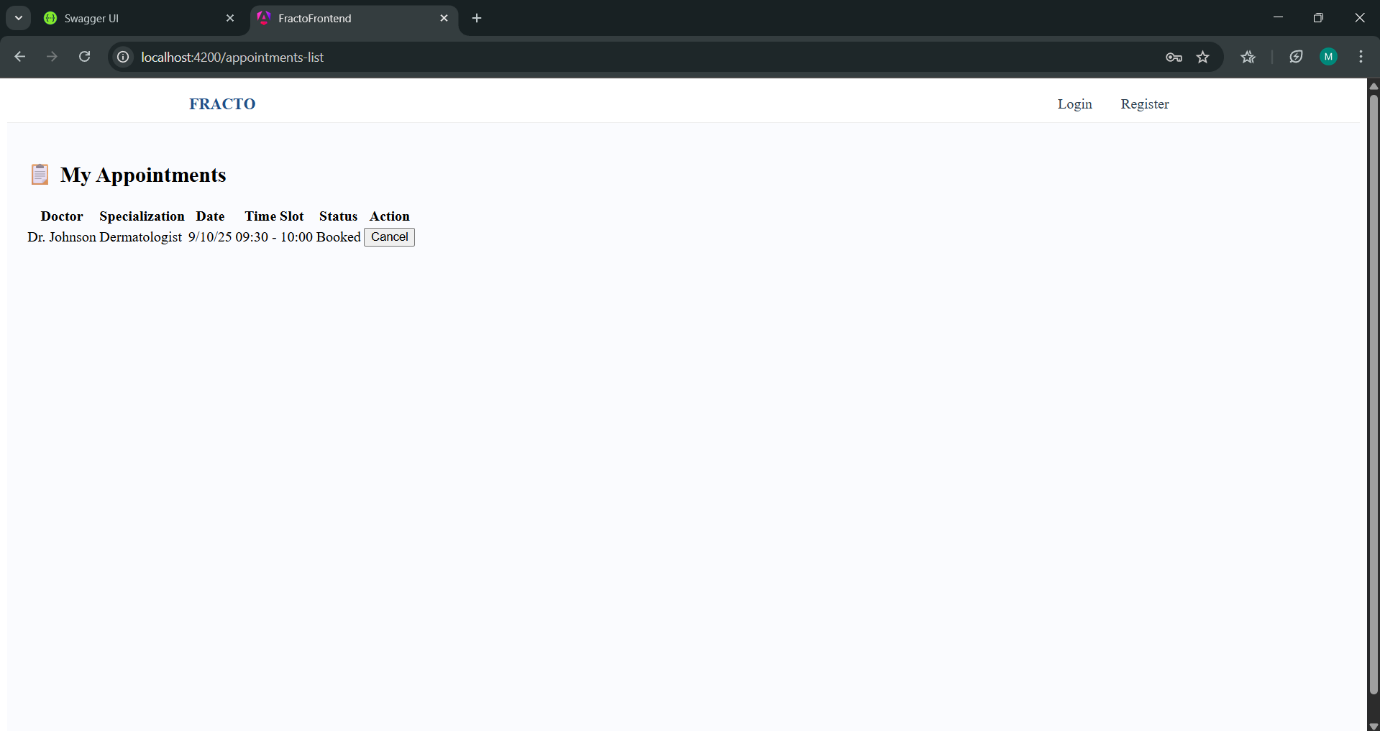
****

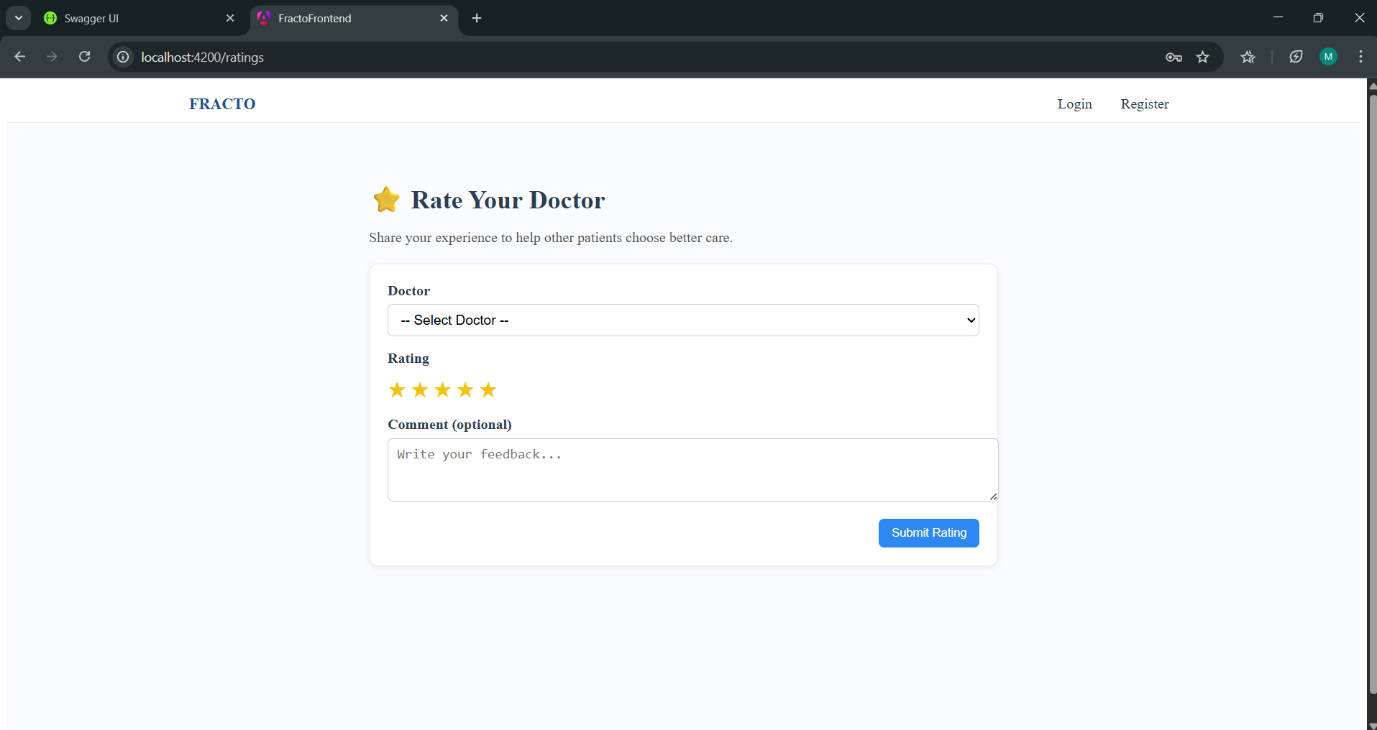
****

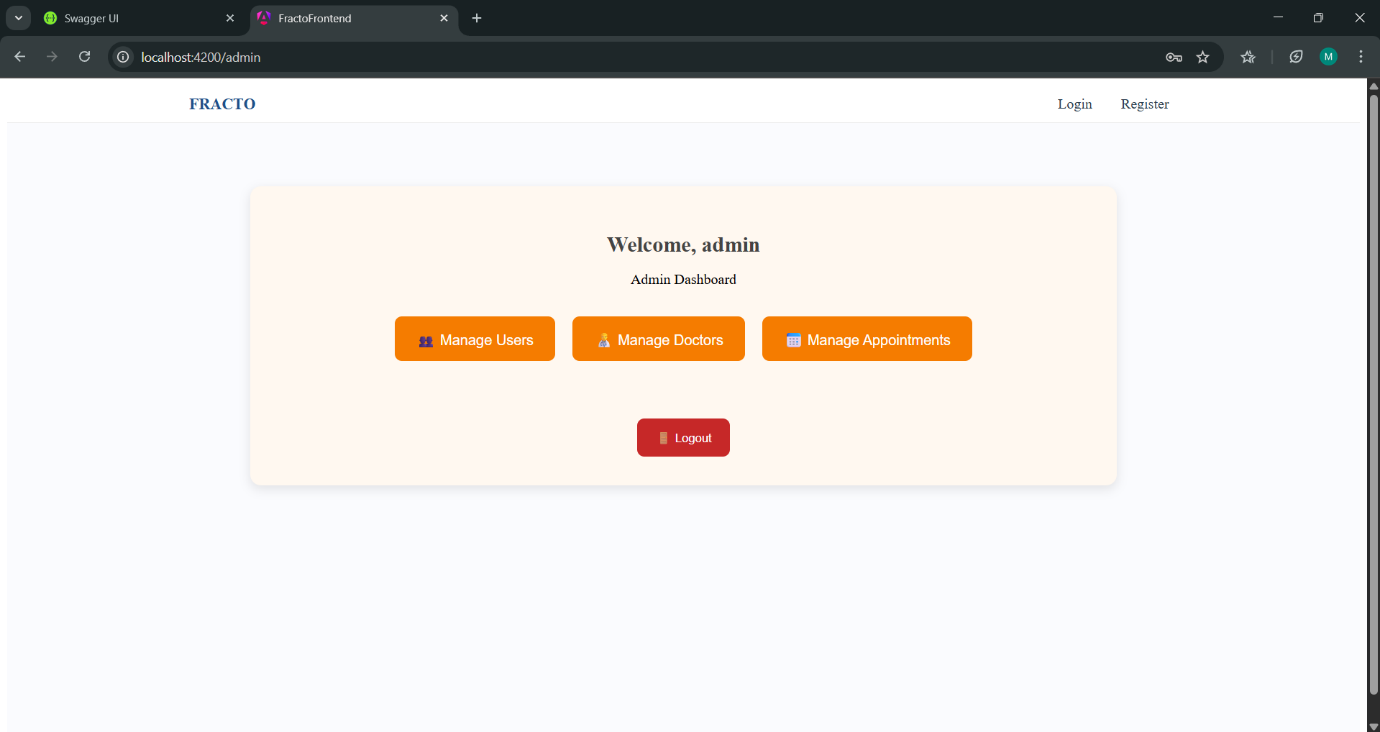
****

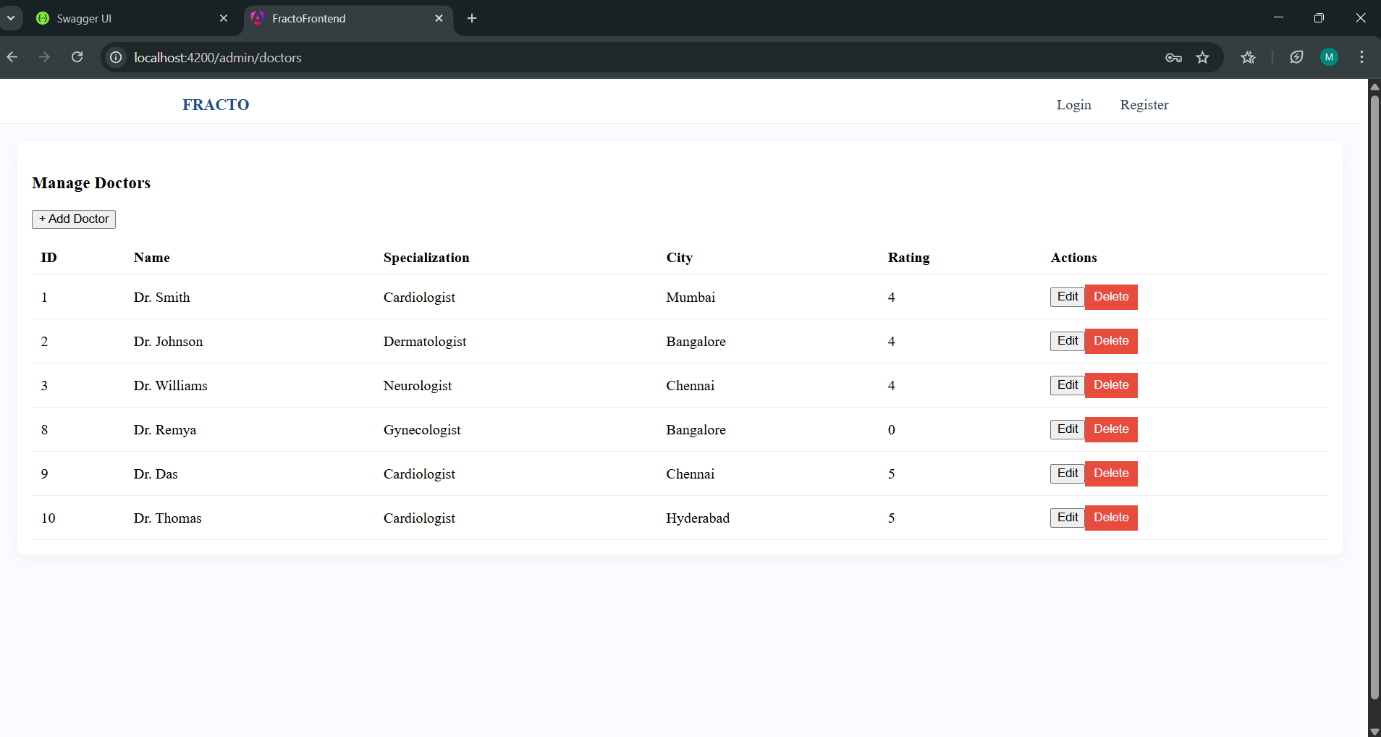
****

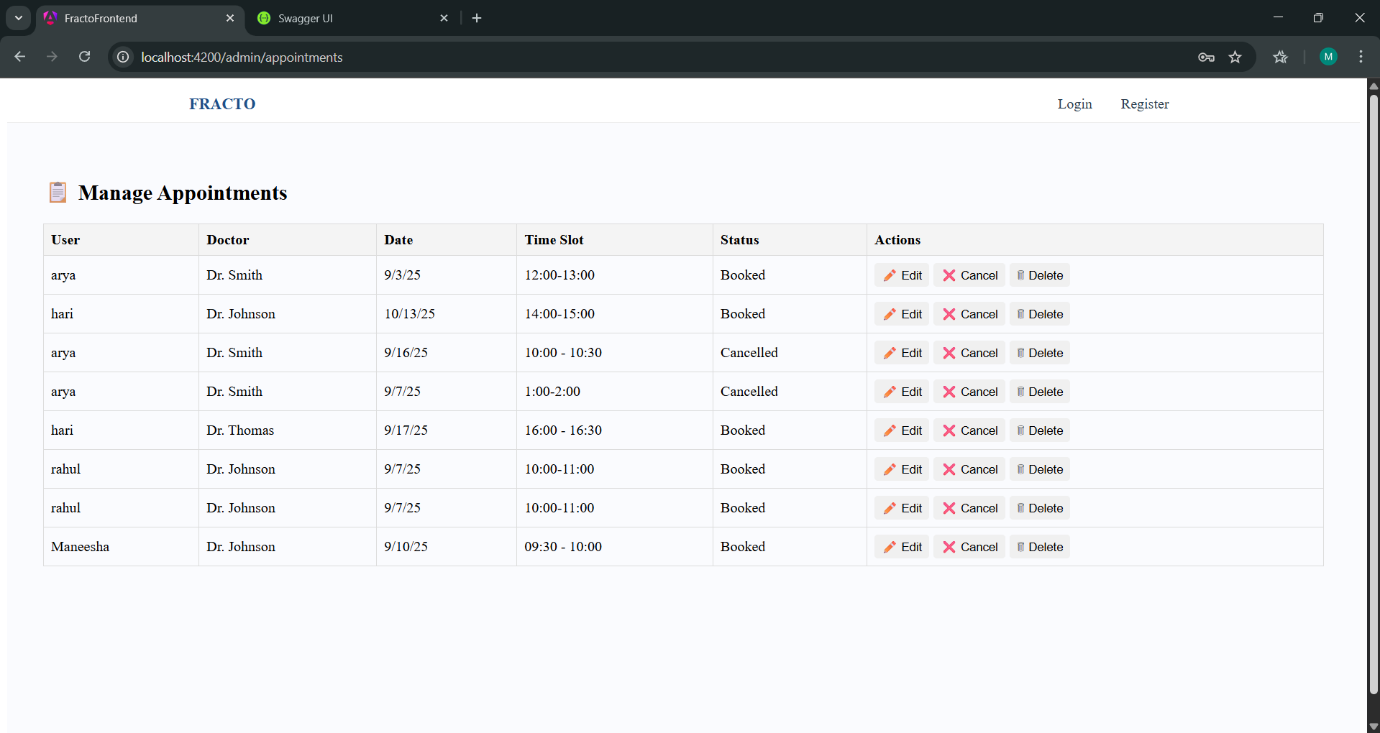
****

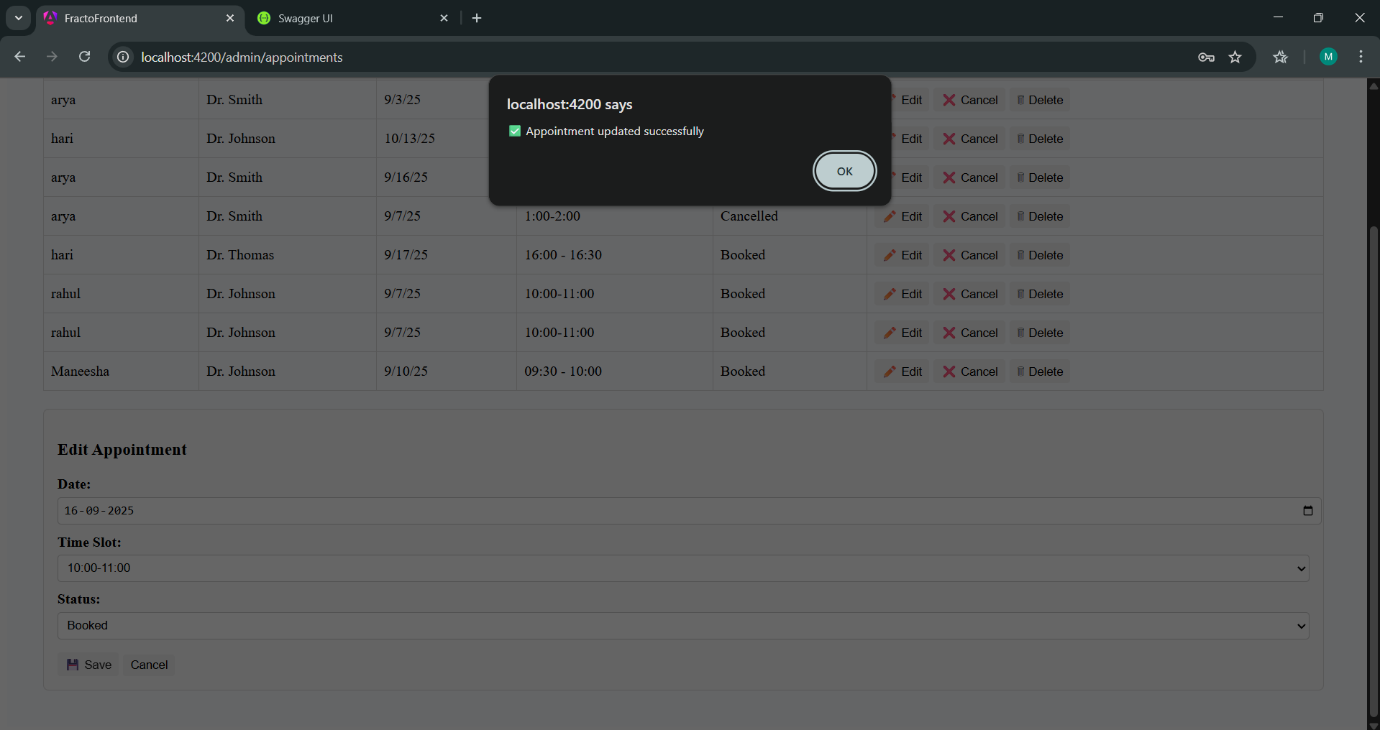
****

****

****

****

****

****

**8. CONCLUSION & FUTURE WORK**

**8.1 Conclusion**

The Fracto project successfully delivers a fully functional online doctor appointment booking system. The platform enables users to search for doctors by specialization, city, and availability, book and manage appointments, and submit ratings. Administrators have full CRUD capabilities to manage users, doctors, and appointments. The system is secured with JWT authentication and role-based access control, ensuring data integrity and a seamless user experience across the Angular frontend and [ASP.NET](https://asp.net/) Core backend.

**8.2 Future Work**

* **Real-Time Notifications:** Integrate **SignalR** to provide real-time updates for appointment confirmations, cancellations, and admin alerts.
* **Payment Integration:** Incorporate a secure payment gateway to handle booking fees or consultation payments.
* **Advanced Filtering:** Enhance search with additional filters such as availability by time window, languages spoken, or insurance acceptance.
* **Patient Medical Profiles:** Allow users to maintain basic medical history and profile information for faster booking.
* **Mobile Application:** Develop a cross-platform mobile app using a framework like **Ionic** or **React Native** for greater accessibility.
* **Automated Reminders:** Implement email or SMS reminders for upcoming appointments to reduce no-shows.