# **FRACTO**

-Parth Tinna

## **Table of Contents**

1. Introduction and Project Goals
2. System Architecture
3. Workflow and Design Diagrams
4. Application Modules & API Overview
5. Database Schema and Data Handling
6. Testing Strategy
7. Conclusion

## **1. Introduction and Project Goals**

### **Problem Context**

In many hospitals and clinics, patients still depend on outdated methods to book appointments, such as phone calls or in-person scheduling. These approaches are time-consuming, prone to errors, and offer little to no transparency about doctor availability. Administrators, meanwhile, must manually track appointments, often resulting in double-bookings or missed slots.

### **Project Goals**

The Fracto system was developed with the following intentions:

* Deliver a **web-based platform** that simplifies appointment booking for patients.
* Allow **administrators to efficiently manage** doctors, patients, and schedules.
* Guarantee **security** using JWT authentication and role-based permissions.
* Provide **real-time updates** through SignalR to improve communication.
* Enhance user engagement with a **doctor rating and feedback system**.
* Support **profile image uploads** to maintain professional and user identities.

## **2. System Architecture**

### **Chosen Technologies**

* **Frontend**: Angular 16 with Angular Material for responsive, interactive UI.
* **Backend**: ASP.NET Core Web API (.NET 9) with EF Core ORM for smooth data operations.
* **Database**: SQL Server for production; SQLite during development for simplicity.
* **Security**: JWT-based authentication tokens with embedded claims.
* **File Storage**: Local file system (wwwroot/uploads).

### **Interaction Model**

1. **Patient/Admin** accesses Angular frontend.
2. Requests are routed to **ASP.NET Core API endpoints**.
3. The backend uses **EF Core** to query and update the database.
4. Uploaded files are stored and retrieved from the server directory.

## **3. Workflow and Design Diagrams**

### **Application Workflow**

* **Patient Workflow**: Register → Login → Select City → Search Doctor → Choose Slot → Book Appointment → Receive Confirmation .
* **Admin Workflow**: Login → Manage Users → Add/Edit Doctors → Approve/Reject Appointments

### **Design Flow**

* **Frontend** handles navigation, form validation, and API requests.
* **Backend** provides endpoints for each feature (Auth, Users, Doctors, Appointments, Ratings).
* **Database** maintains relationships between users, doctors, and appointments.
* **SignalR Hub** ensures real-time synchronization.

## **4. Application Modules & API Overview**

### **Angular Modules**

* **Authentication**: Components for Login and Register with Guards for route protection.
* **User Section**: DoctorSearchComponent, AppointmentBookComponent, UserAppointmentsComponent.
* **Admin Section**: Components for User Management, Doctor Management, Specialization Management, and Appointment Oversight.
* **Shared Utilities**: Navbar, Snackbar notifications, and FileUpload component.

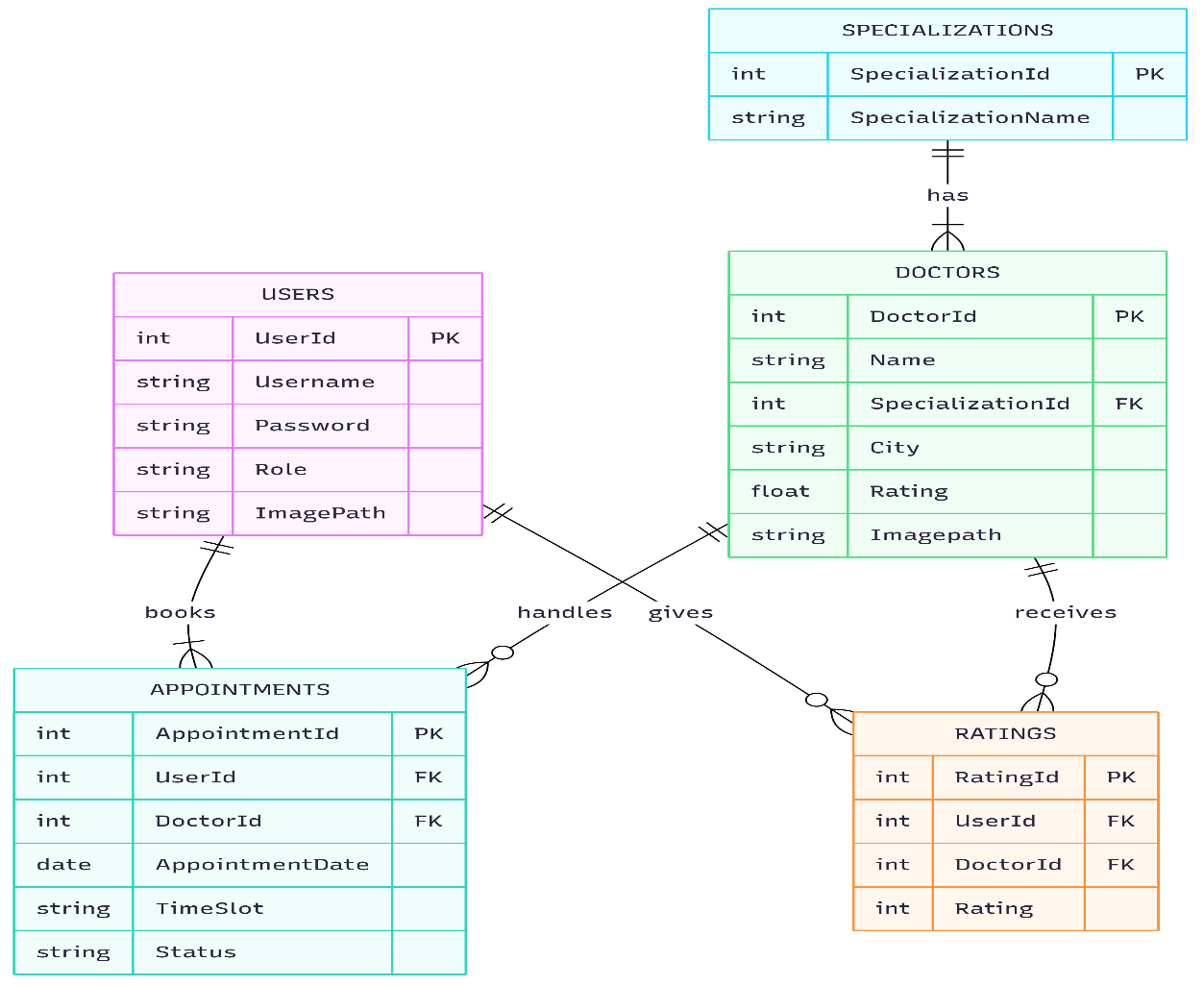
### **Web API Controllers**

* **AuthController** → Register & Login.
* **UsersController** → Manage patients (CRUD, Admin only).
* **DoctorsController** → CRUD, search by filters, and timeslot availability.
* **AppointmentsController** → Book, reschedule, cancel, approve, reject.

## **5. Database Schema and Data Handling**

### **Core Tables**

* **Users**: UserId, Username, Password, Role, ProfileImagePath.
* **Doctors**: DoctorId, Name, SpecializationId, City, Rating, ProfileImagePath.
* **Specializations**: SpecializationId, Name.
* **Appointments**: AppointmentId, UserId, DoctorId, AppointmentDate, TimeSlot, Status.
* **Ratings**: RatingId, DoctorId, UserId, Rating, Comment.

**ER Diagram :**

### **Data Handling Strategies**

* Referential integrity enforced with **foreign keys**.
* Indexes on **UserId, DoctorId, SpecializationId** for faster queries.
* Frontend caches data like specializations to reduce repeated API calls.

## **6. Testing Strategy**

### **Backend**

* **Frameworks**: xUnit with Moq.
* **Approach**: EF Core InMemory database to test controllers without affecting production.
* **Scenarios Tested**:  
  + User registration and login produce valid JWTs.
  + Booking an appointment creates a record.
  + Admin approval updates appointment status correctly.

### **Frontend**

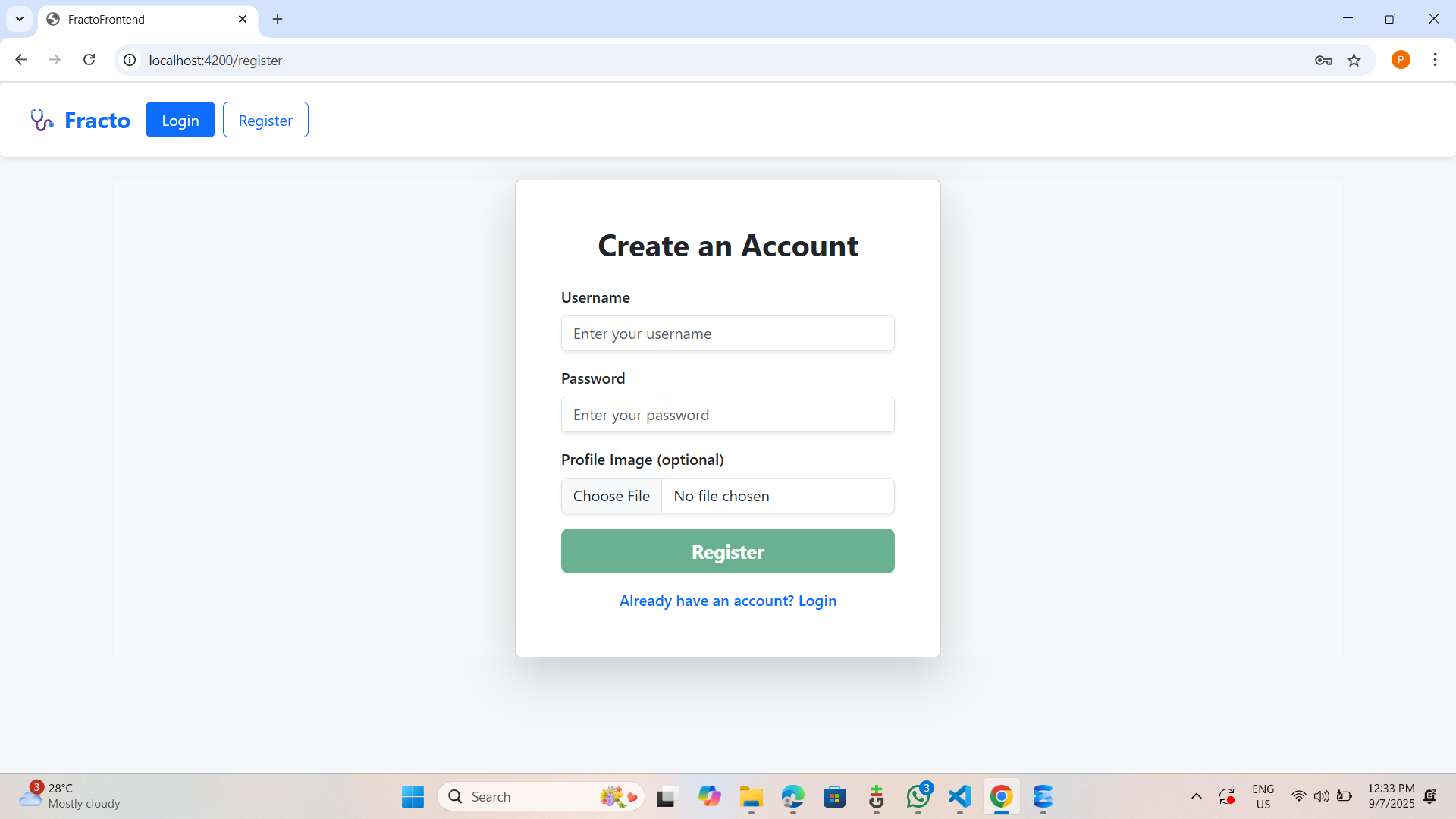
* **Tools**: Jasmine + Karma.
* **Unit Tests**:  
  + Login form validates required fields.
  + Appointment booking triggers the correct API call.
  + Doctor search renders accurate data.

### **Integration**

* **Postman** used for testing REST APIs.
* Manual linking tests with Angular frontend and ASP.NET backend.
* Future E2E scope with Cypress.

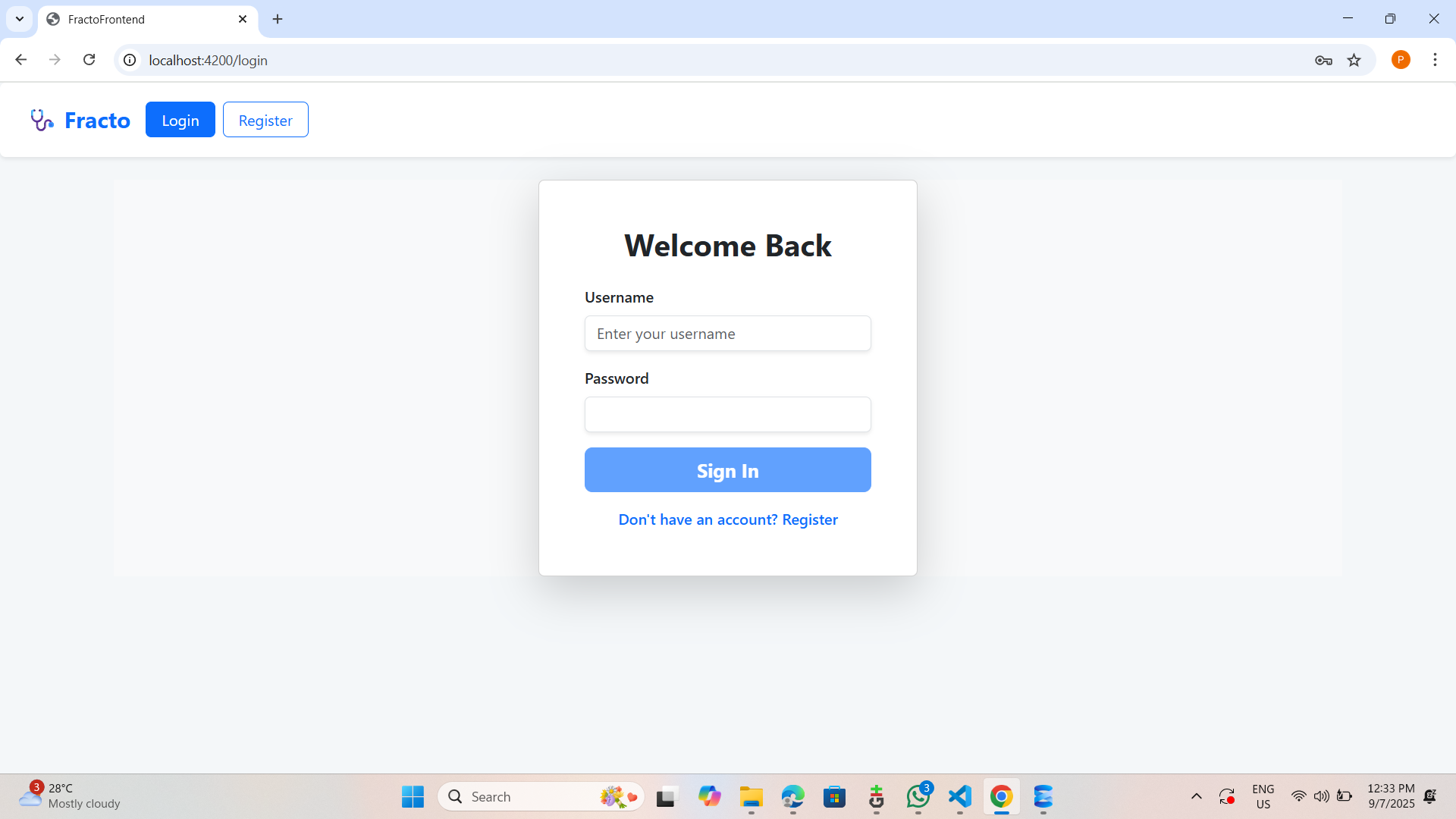
## **7. Implementation Screenshots**

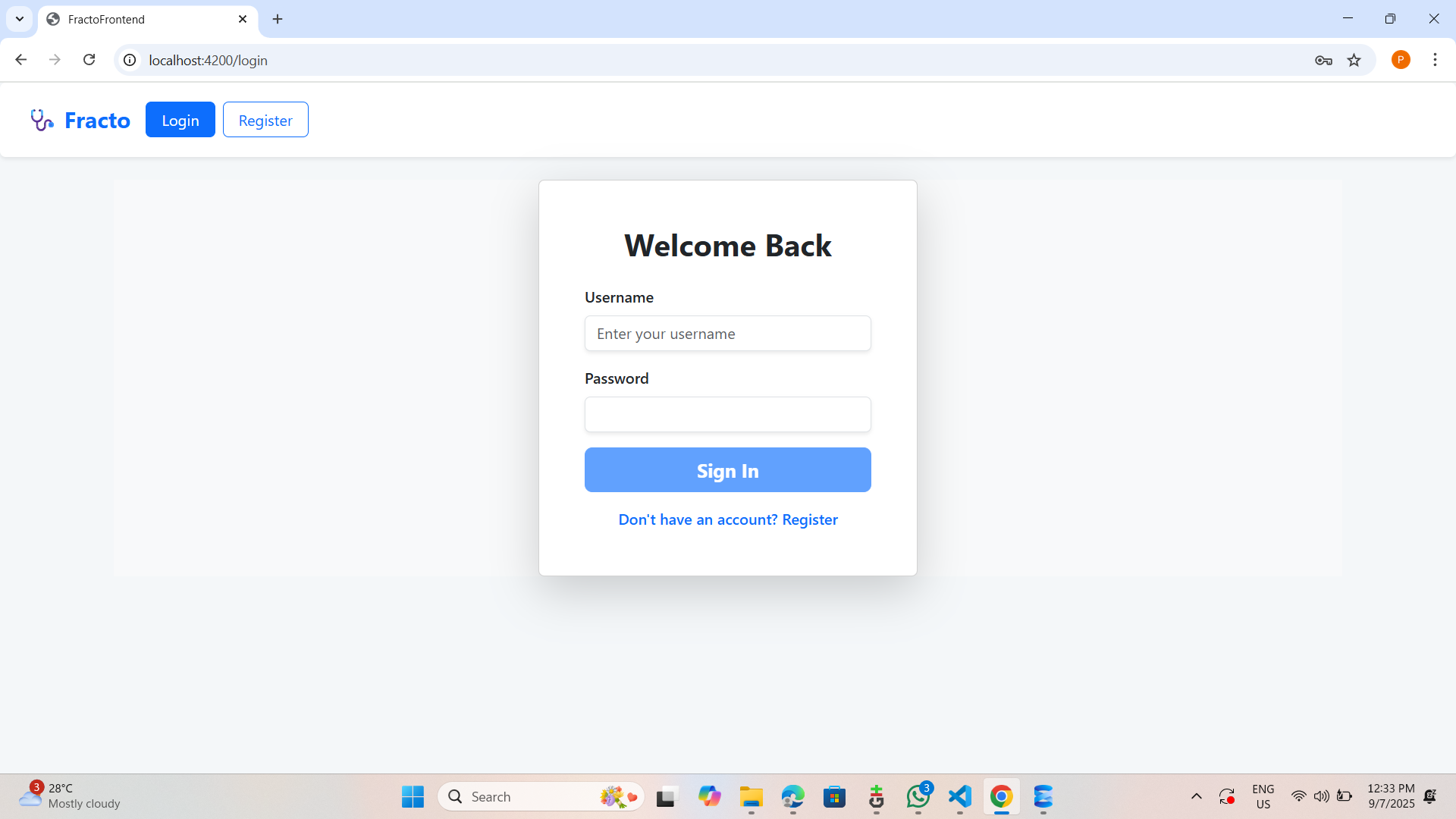
User Register:



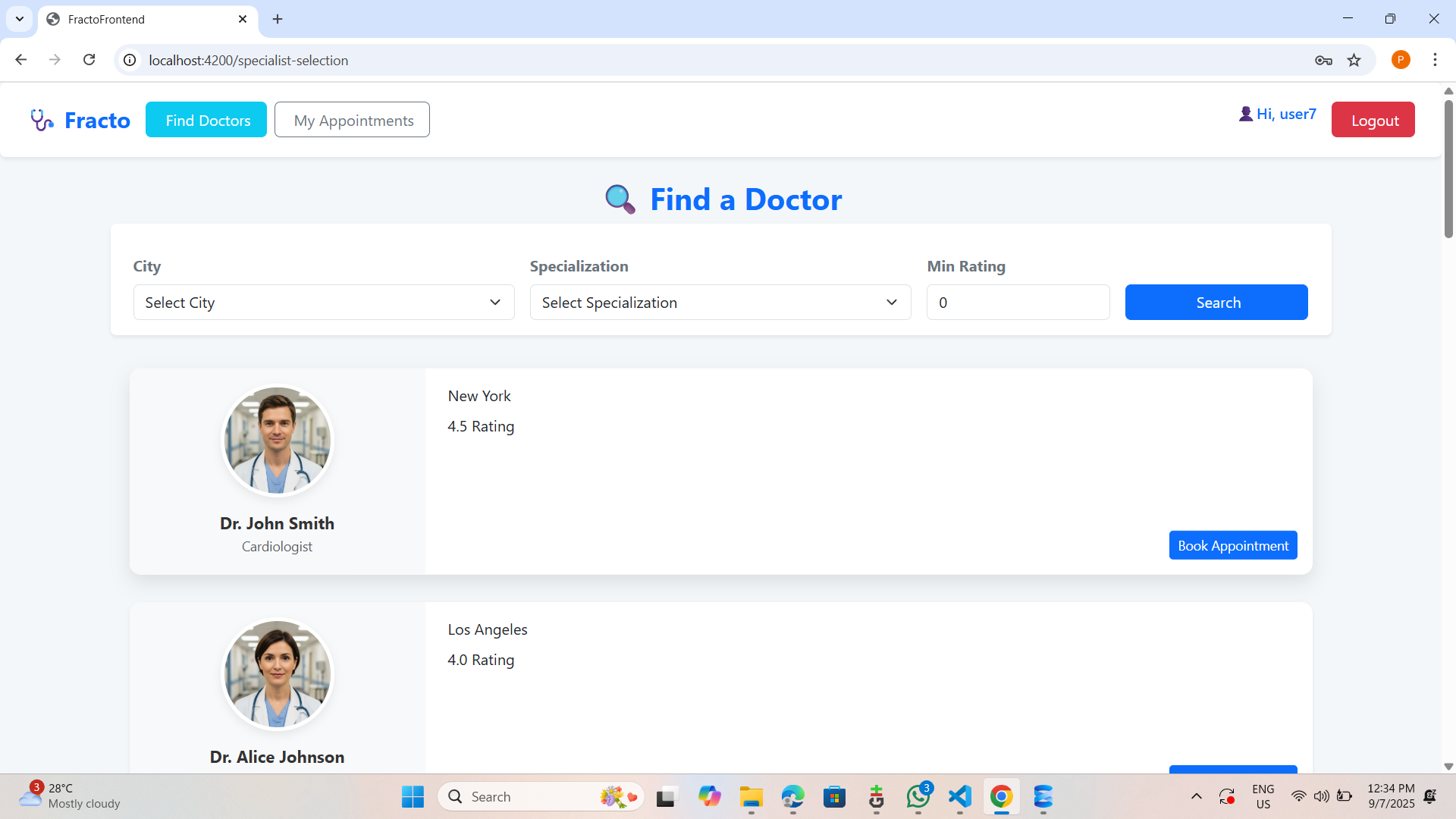
:

User Login:



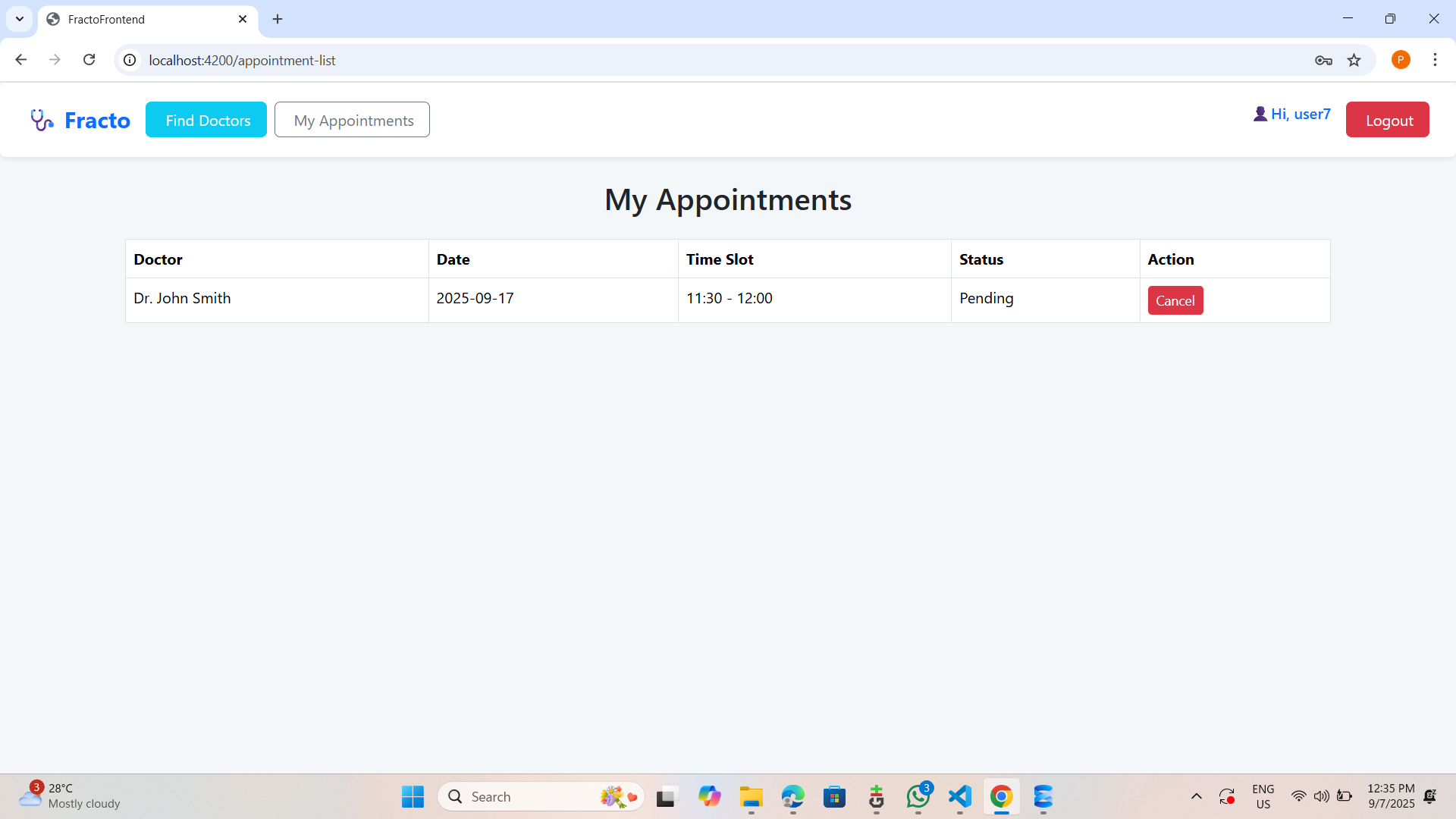


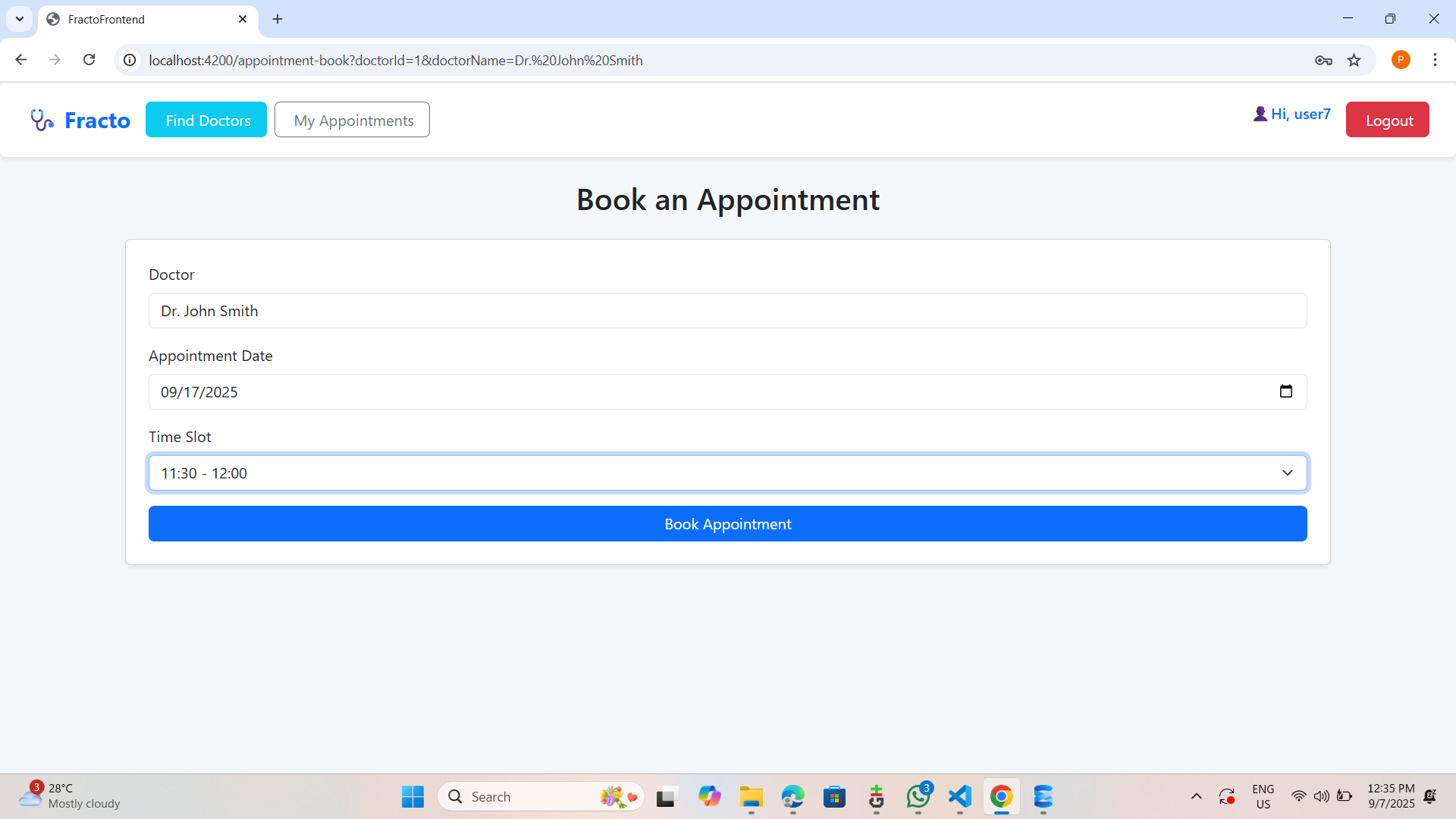
**Doctor’s List:**



**Booking Appointment On Basis of Slot:**

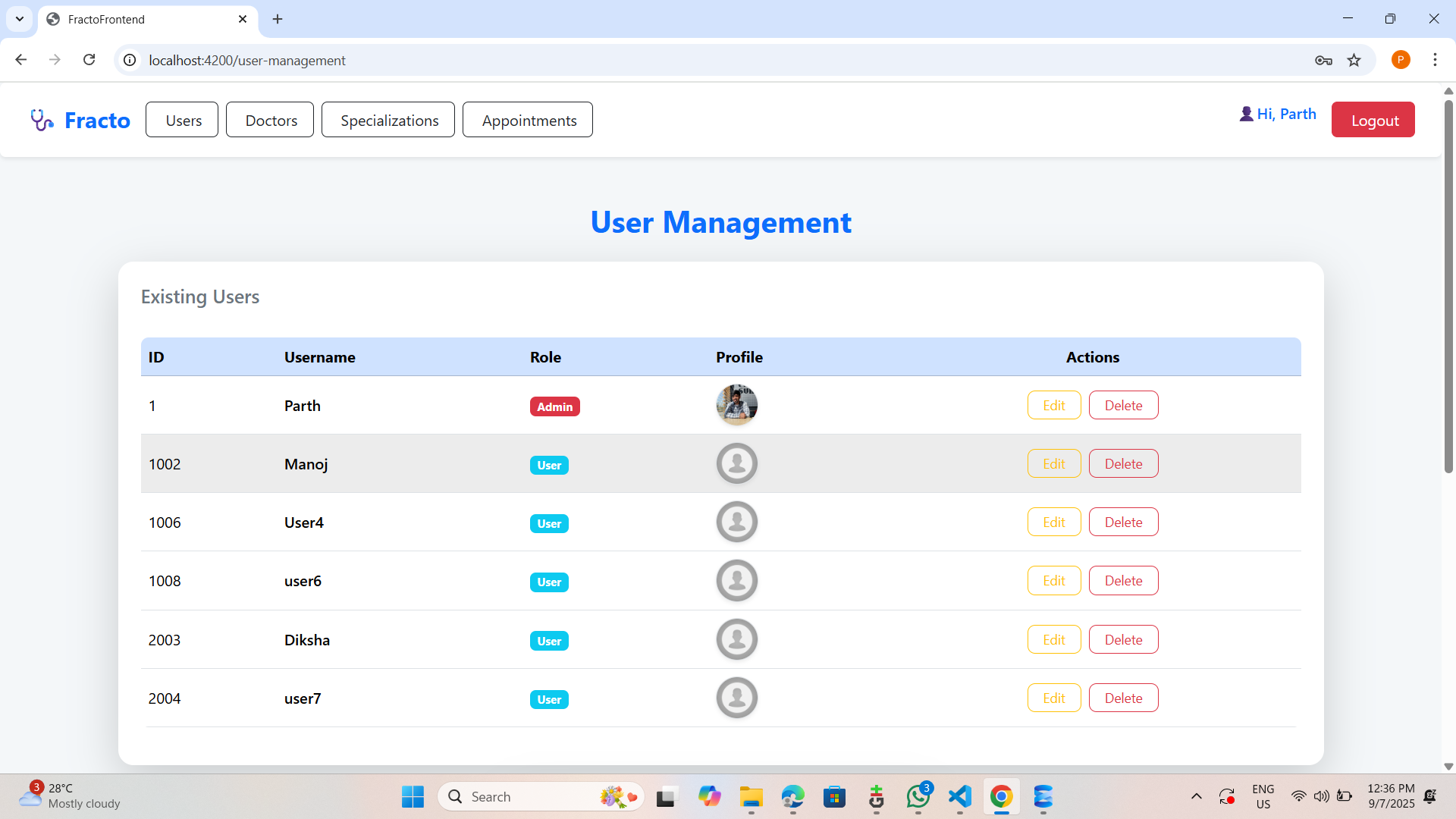
**Checking Appointments:**

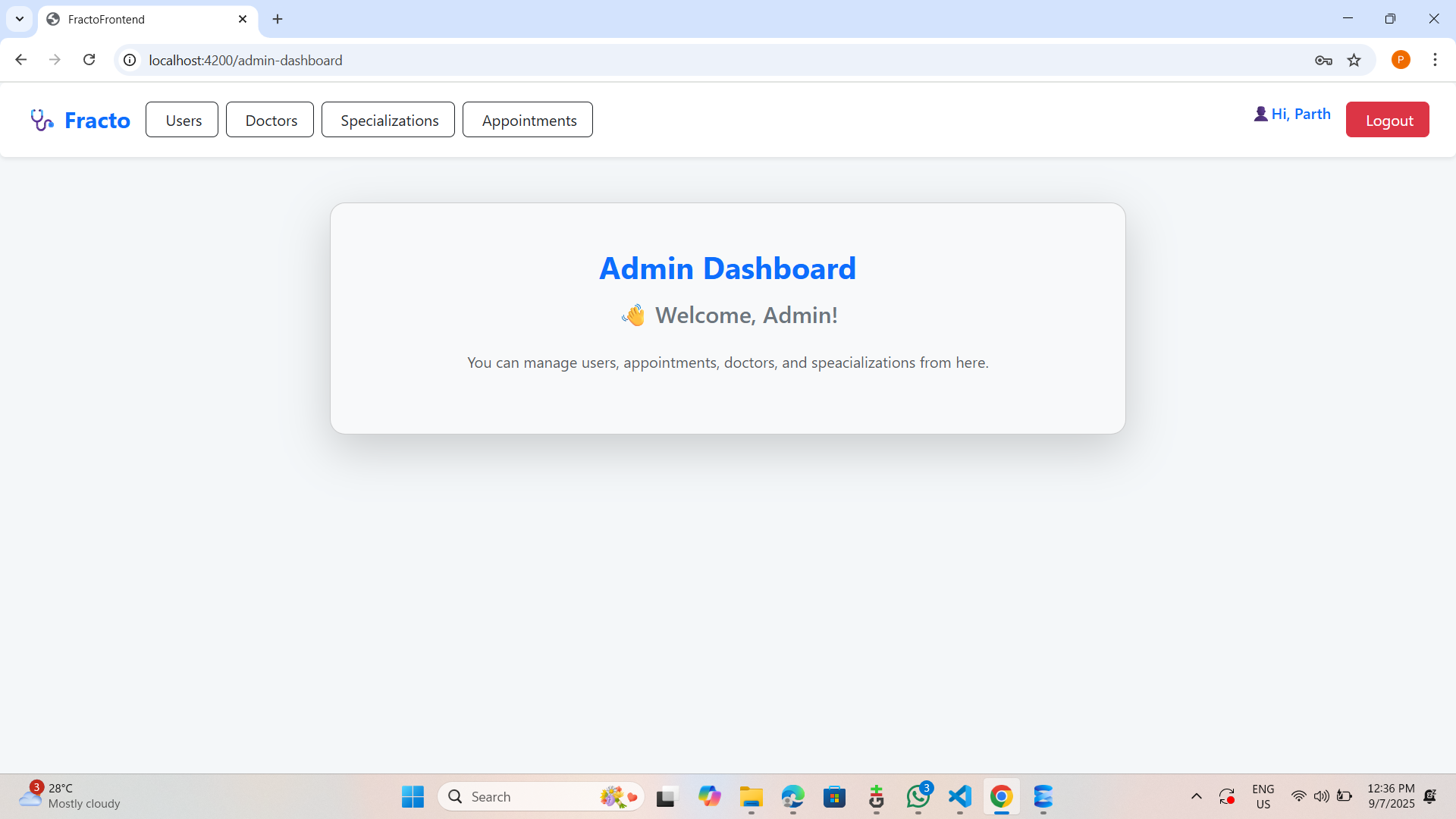




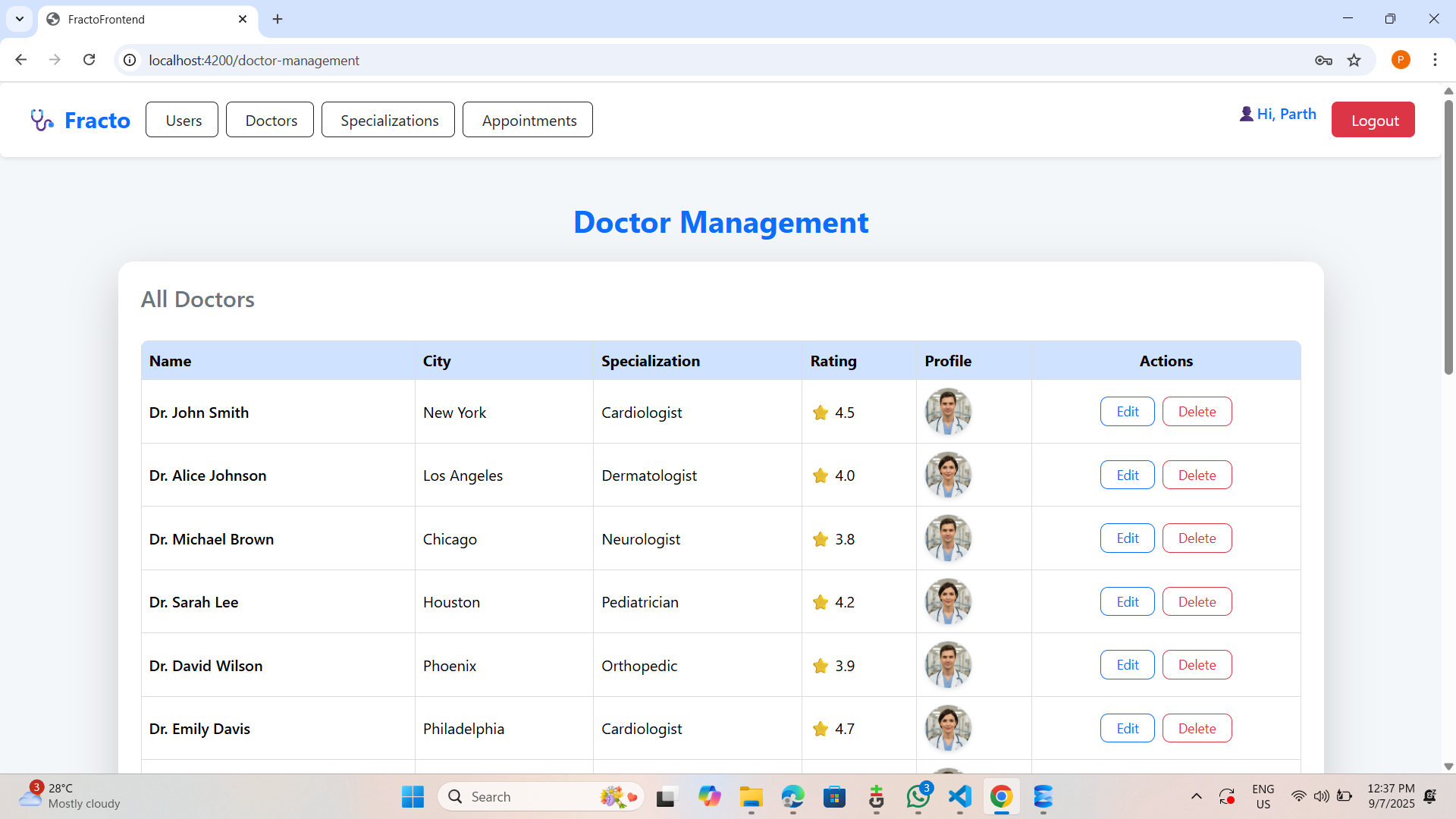
**Admin Side**

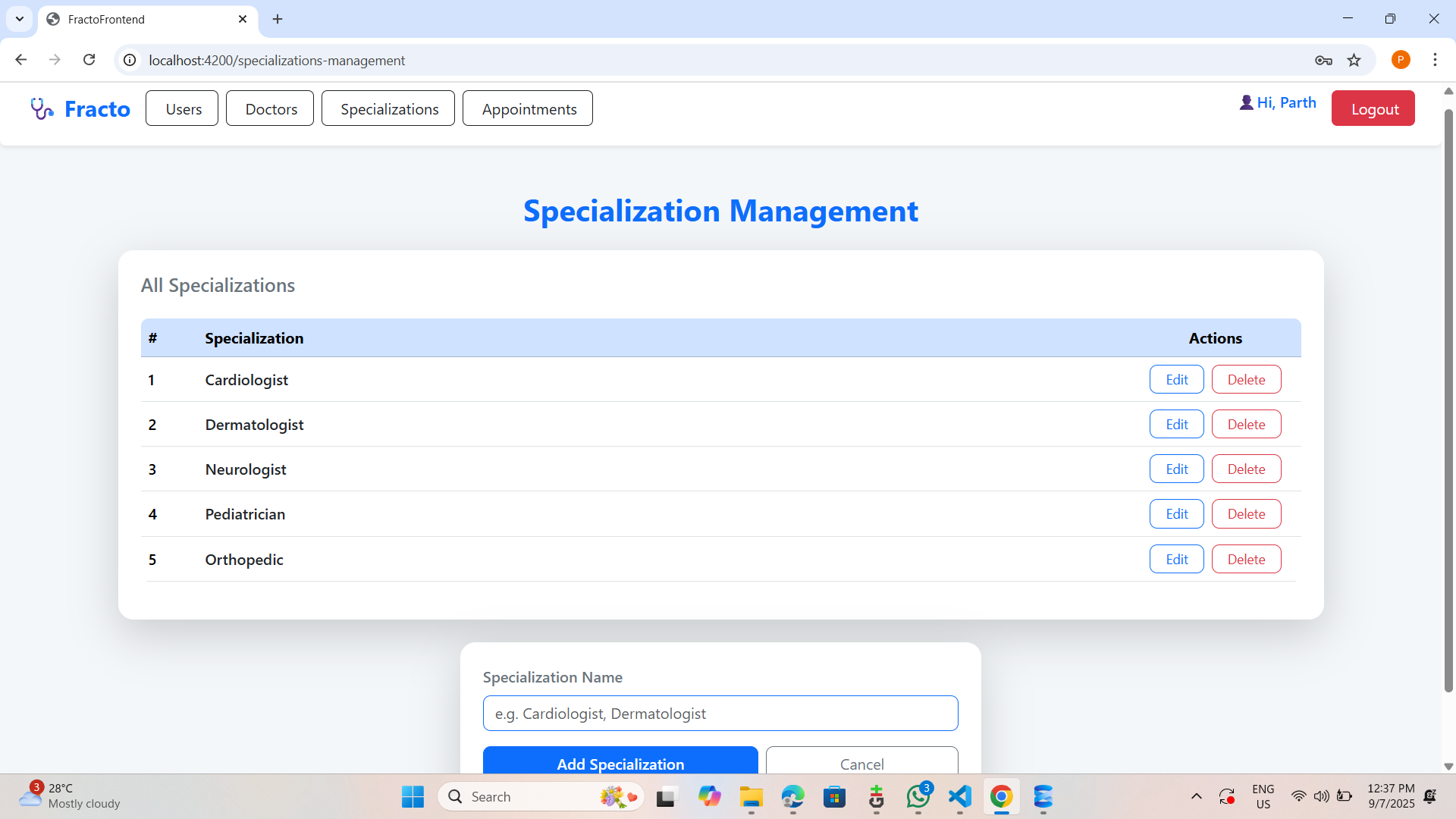
**Admins Dashboard:**

**Admin User Management:**



**Admin Doctors Management:**



**Admin Specializations Management**

**Admin Appointment Management:**

## **Conclusion**

Fracto has been built to streamline healthcare appointment management through a digital-first solution. It bridges the gap between patients and healthcare providers with **real-time visibility, secure access, and user-friendly interfaces**.

The project successfully delivers:

* **Ease for patients** in booking and managing appointments.
* **Efficiency for administrators** in overseeing users, doctors, and schedules.
* **Security** with robust authentication and role-based permissions.
* **Engagement** through ratings and instant notifications.

