FRACTO

By: VISHAKHA PRASAD

Table of Contents:

---

1. Problem Definition and Objectives

Problem Statement:

Booking doctor appointments is often inefficient. Patients struggle with visibility of available slots, manual booking processes, and difficulties in rescheduling. Doctors and administrators face challenges in record-keeping, managing cancellations, and providing real-time updates.

Objectives:

● Provide a digital platform for users to register, search doctors, and book/cancel

appointments.

● Develop an admin interface for managing doctors, users, and appointments.

● Implement JWT-based authentication for security.

● Support file uploads (profile images for doctors and users).

● Add real-time notifications using (SignalR).

● Enable a rating system for users to rate doctors.

---

2. Frontend & Backend Architecture

Technology Stack:

● Frontend: Angular 16, Angular Material, RxJS, Forms, SignalR Client

● Backend: ASP.NET Core Web API (.NET 9), Entity Framework Core, SignalR

● Database: SQL Server / SQLite (development)

● Authentication: JWT Tokens (with UserId and Role claims)

● File Storage: Local server (wwwroot/uploads)

Architecture Overview

- **Frontend:**

○ Angular components for user and admin roles.

○ Routing for login, register, search, booking, and admin dashboards.

○ Services for API communication (Auth, Users, Doctors, Appointments, Ratings, Files).

- **Backend:**

○ REST APIs with controllers for each module (Auth, Users, Doctors, Appointments, Ratings, Specializations, Files).

○ EF Core handles database operations.

○ JWT Authentication middleware secures APIs.

○ SignalR hub broadcasts real-time events.

- **Database:**

○ Normalized schema with foreign key relationships. ○ Stores references to uploaded files.

---

3. System Design Diagram

Flow:

- Users/Admins interact with Angular Frontend.
- Angular sends requests to ASP.NET Core Web API controllers.
- Controllers use Entity Framework Core to access SQL Server/SQLite.
- SignalR Hub provides real-time notifications (appointment confirmations, cancellations).
- File Upload Controller handles storing and retrieving profile images.

---

4. Component Breakdown & API Design

Frontend Components

Core Module

1. Guards: admin.guard.ts, auth.guard.ts
2. Interceptors: jwt.interceptor.ts
3. Services: auth.service.ts (likely manages user authentication across the app)
4. Models: user.ts (the fundamental user data model)

Shared/Common Module

1. Components: navbar.component.ts (a shared navigation bar)

Appointment Module

2. Pages: book-appointment.component.ts, my-appointments.component.ts
3. Services: appointment.service.ts

4. Models: appointment.ts

Doctor Module

1. Pages: doctor-list.component.ts
2. Services: doctor.service.ts, specialization.service.ts
3. Models: doctor.ts
4. Ratings Module

Admin Module

1. Pages: ratings-page.component.ts
2. Services: rating.service.ts

Authentication Module

1. Pages: login.component.ts, register.component.ts

Admin Module

1. Pages: admin-appointments.component.ts, manage-doctors.component.ts

API Design

AdminUsers

- POST /api/Appointments
- GET /api/Appointments
- PUT /api/Appointments/cancel/{id}
- GET /api/Appointments/user/{userId}
- GET /api/Appointments/pending
- PUT /api/Appointment/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/doctors/{doctorsId}/avg
- GET /api/Ratings/doctors/{doctorsId}

Specialisations

- GET /api/Specializations
- POST /api/Specializations
- GET /api/Specializations/{id}
- PUT /api/Specialization/{id}

- DELETE /api/Specialization/{id}

Users

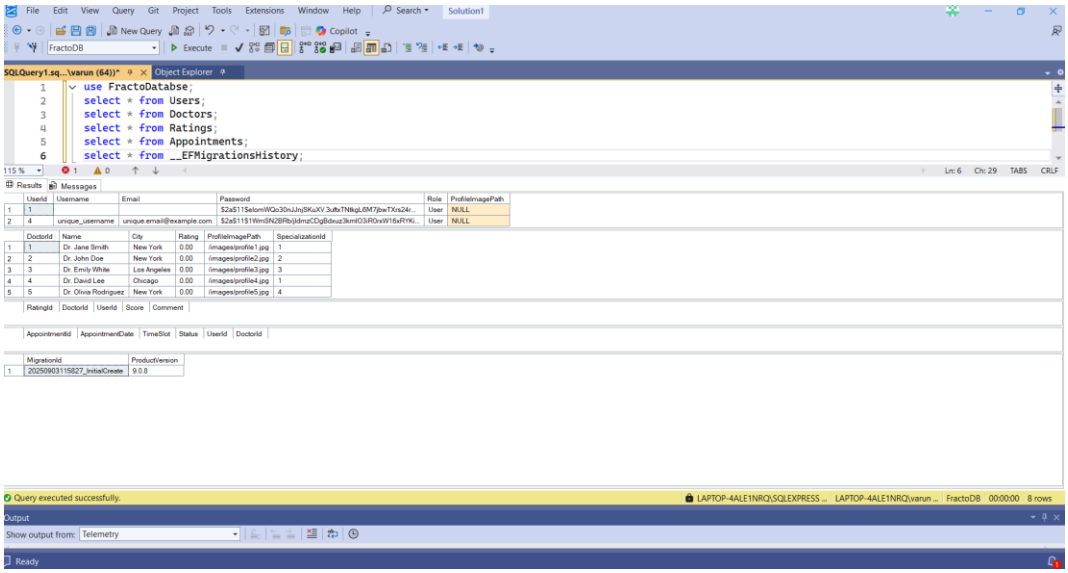
- GET /api/Users/{id}
- POST /api/Users/{id}/avatar

---

5. Database Design & Storage Optimization

Entities

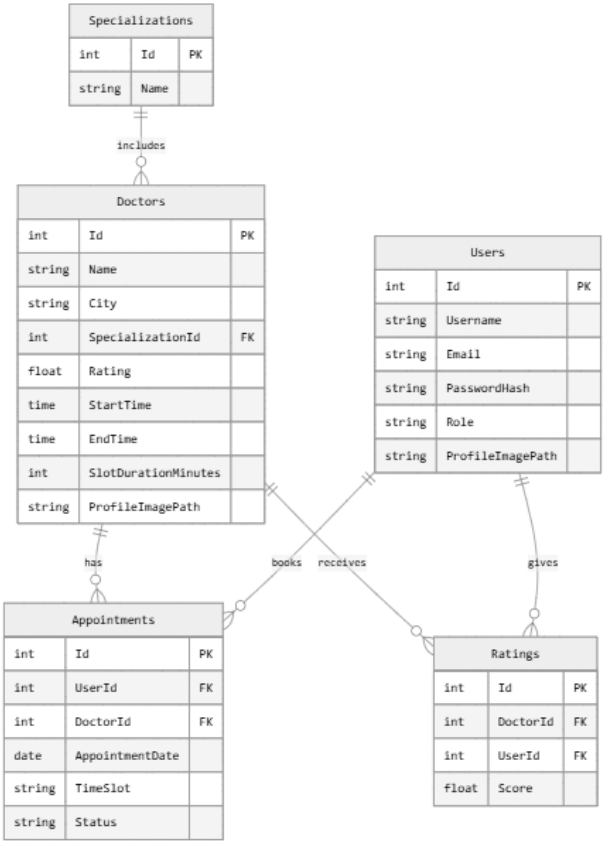
- Users(UserId, Username, Email, PasswordHash, Role, City, ProfileImagePath, CreatedAt)
- Doctors(DoctorId, Name, SpecializationId, City, Rating, ProfileImagePath, CreatedAt)
- Specializations(SpecializationId, SpecializationName)
- Appointments(AppointmentId, UserId, DoctorId, AppointmentDate, TimeSlot, Status)
- Ratings(RatingId, DoctorId, UserId, Rating, Comment, CreatedAt)



Relationships:

- One-to-Many relationship between Doctors and Appointments (one doctor can have multiple appointments).
- One-to-Many relationship between Users and Appointments (one user can have multiple appointments).
- One-to-Many relationship between Doctors and Ratings (one doctor can have multiple ratings).
- One-to-Many relationship between Users and Ratings (one user can rate multiple doctors).
- One-to-Many relationship between Specializations and Doctors (one specialization can have multiple doctors).

ER Diagram:-

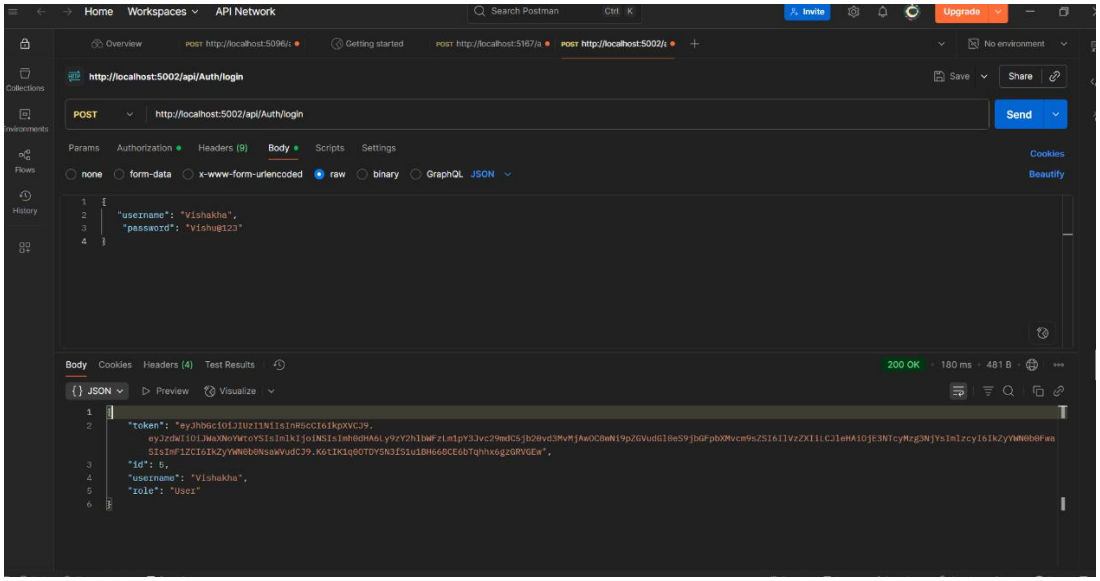


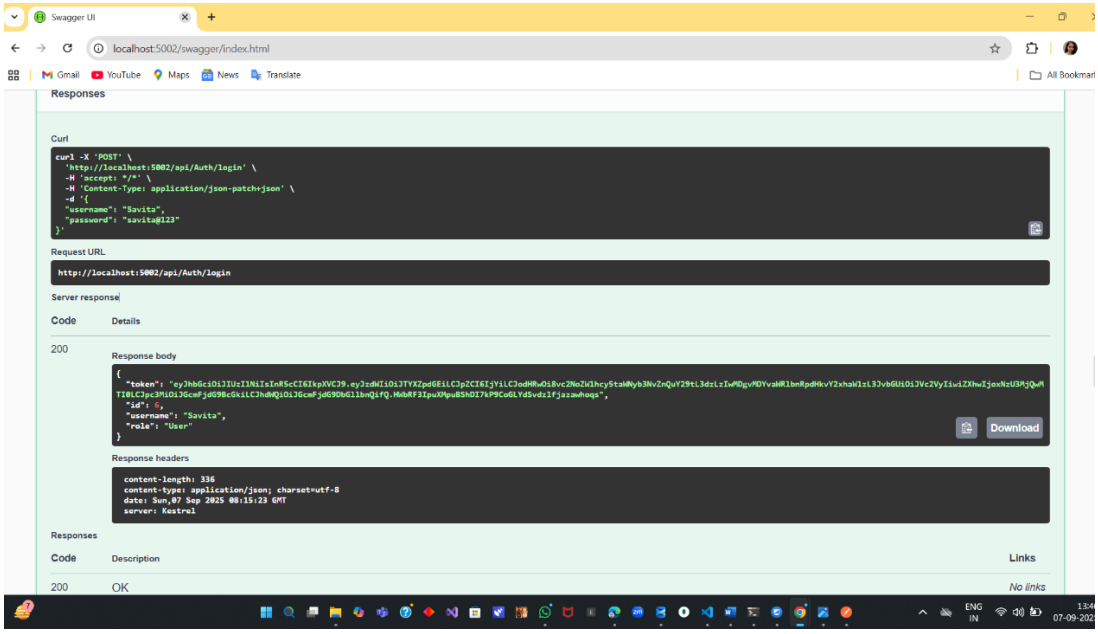
6. Testing Backend Testing

- xUnit + Moq used for unit testing controllers and services.
- EF Core InMemory provider simulates database.

End-to-End Testing

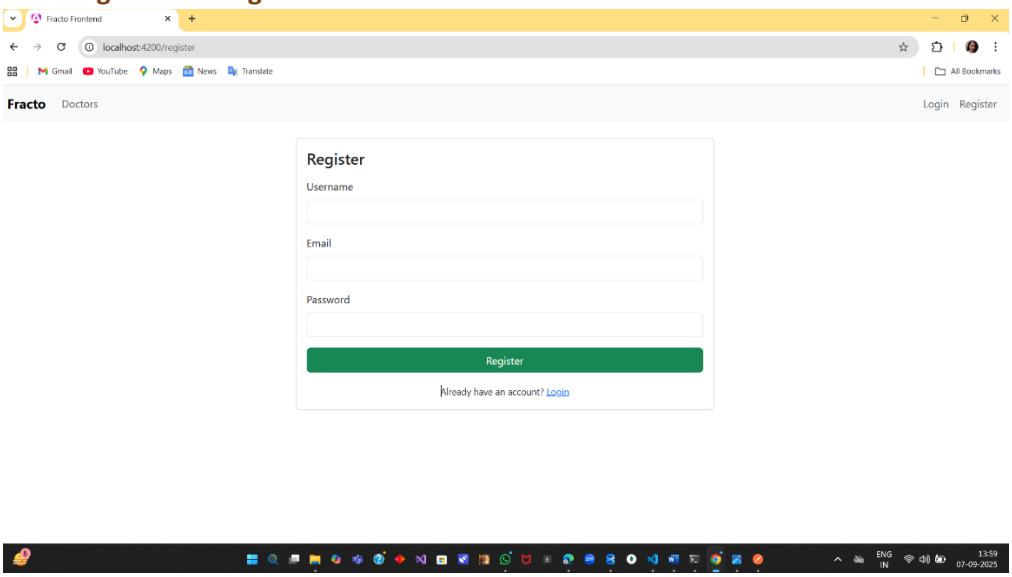
- Postman collections used to test APIs



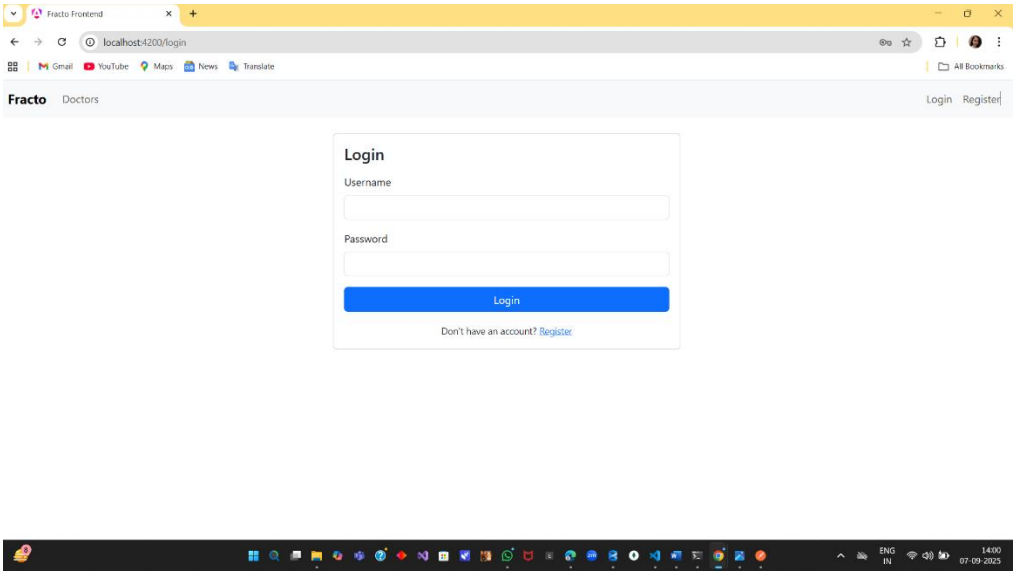


7. Implementation Screenshots

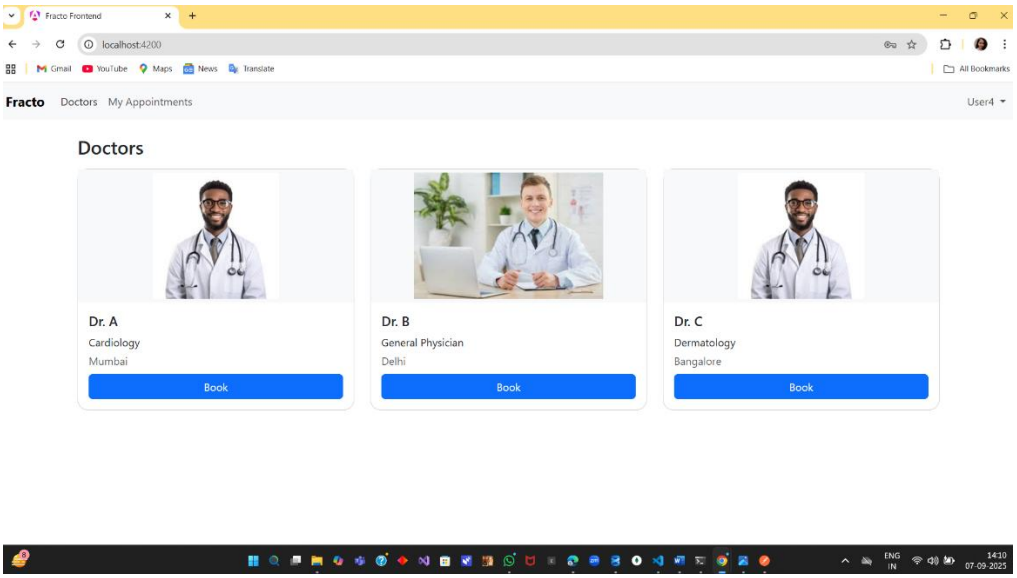
User Registration Page :-



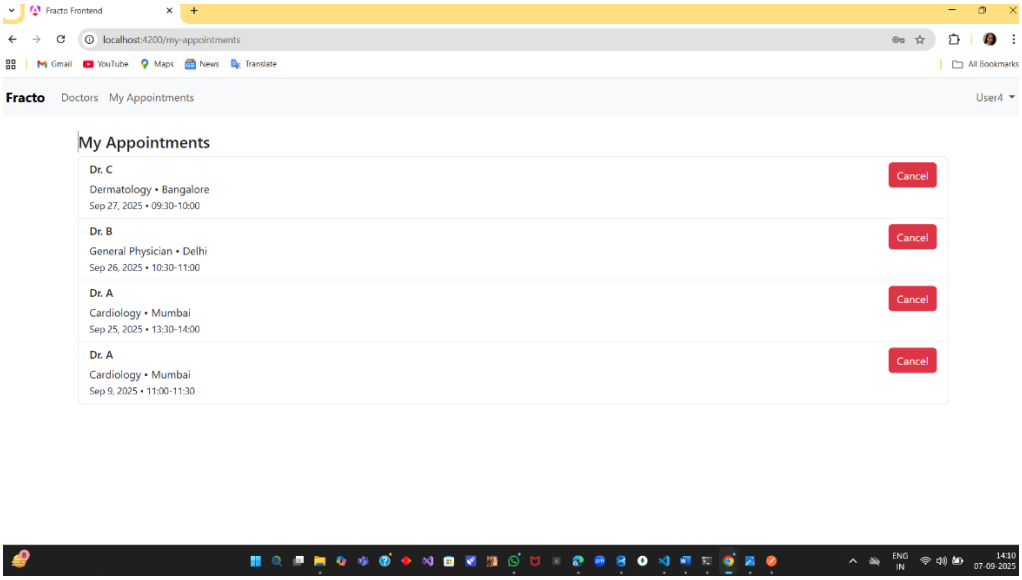
User Login Page:-



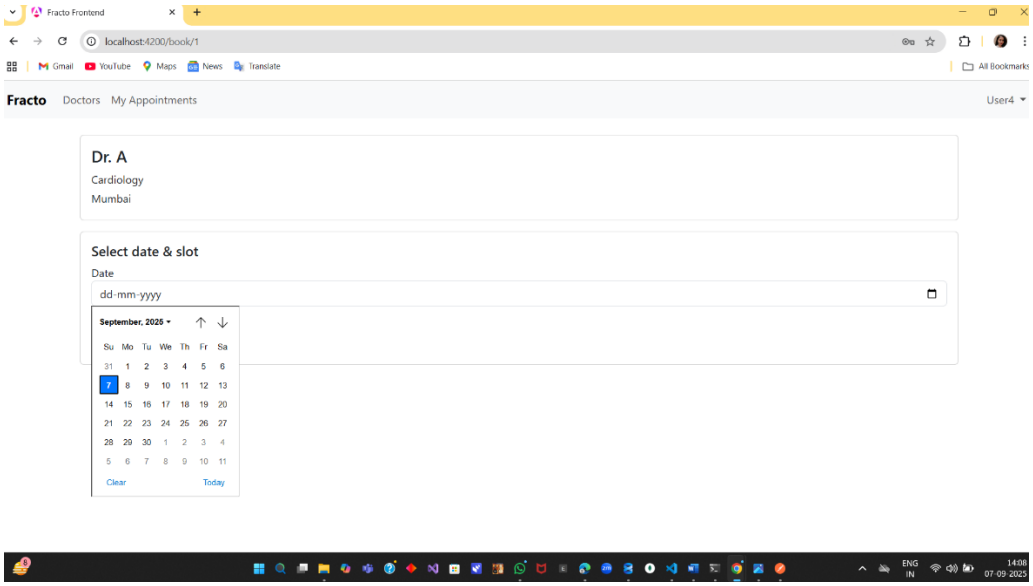
User Home Page:-

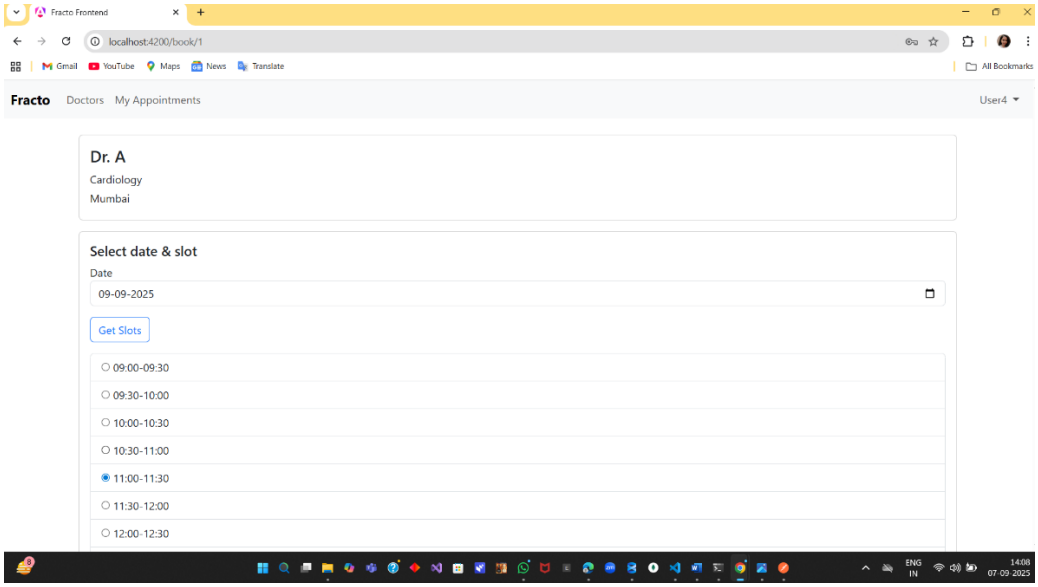


My Appointment Page:-

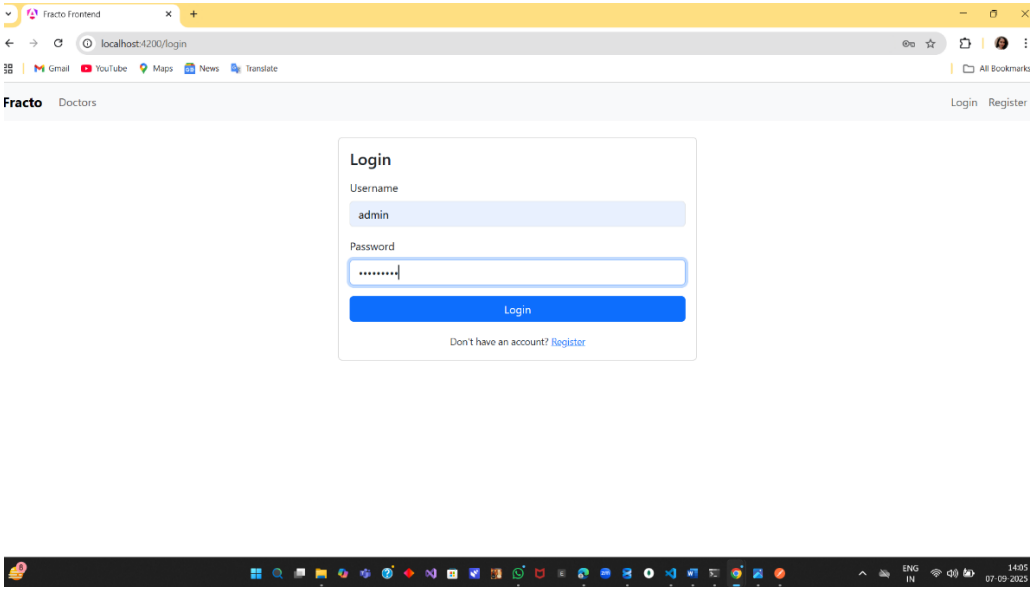


User Can Book Appointment According To Date and Slot:-

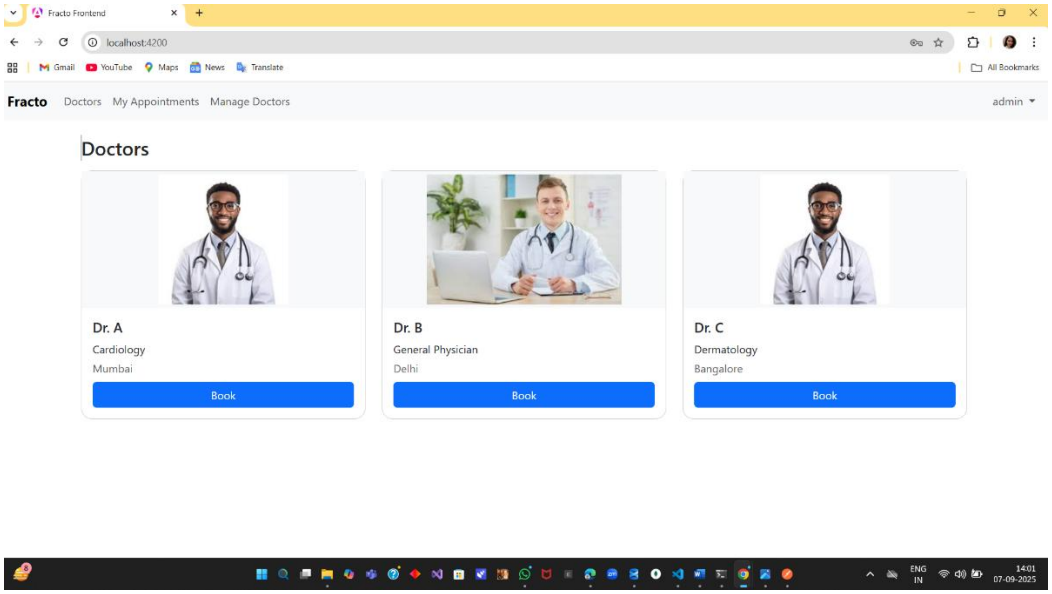




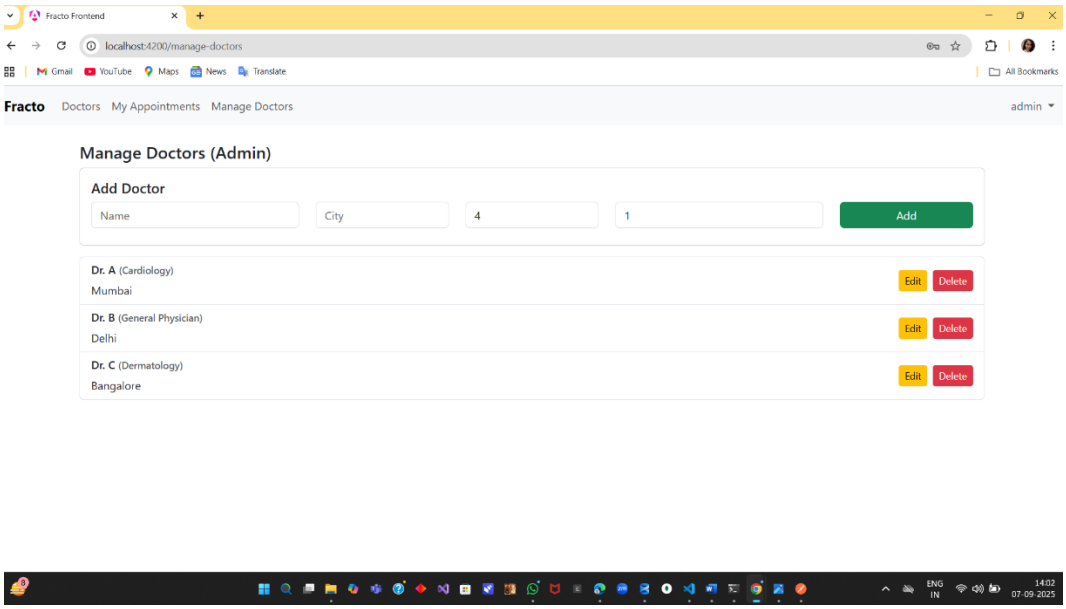
Admin Login Page:-



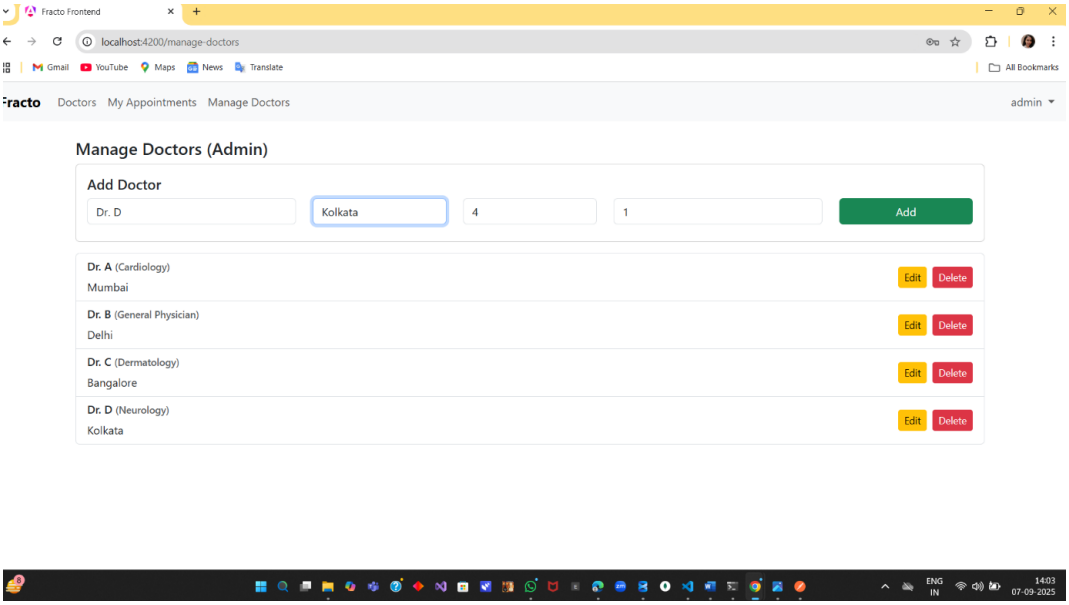
Admin Home Page:-



Admin Can Manage Doctors(using CRUD operations):-



Adding Doctor:-



Managing Appointments:-