Project Name: Fracto (Frontend: React, Backend: ASP.NET Core MVC)

Fracto is an online doctor appointment booking platform where users can search for doctors based on their city, specialization, and available time slots, book appointments, and cancel them if needed. The application has two types of users:

- **User**
- **Admin**

Tech Stack:

- **Frontend:** React
- **Backend:** ASP.NET Core MVC with Web API
- **Database:** SQL Server (using Entity Framework Core as ORM)
- **File Storage:** File upload functionality on the server for profile images.
- **API Documentation:** Swagger for Web API
- **Notifications:** Optional feature (SignalR) for appointment confirmation.

User Stories:

User Stories (User)

1. **User Authentication:**
   As a user, I should be able to log in, log out, and register into the application.
2. **City Selection:**
   As a user, I should be able to select the city for the doctor appointment.
3. **Appointment Date:**
   As a user, I should be able to select the desired date for the appointment.
4. **Select Specialist:**
   As a user, I should be able to select the required specialist from a list of specializations.
5. **View Available Doctors:**
   As a user, I should be able to view available doctors in the selected specialization for the selected date.
6. **Filter by Ratings:**
   As a user, I should be able to filter doctors based on their ratings.
7. **Doctor Selection:**
   As a user, I should be able to select a doctor from the list.
8. **View Time Slots:**
   As a user, I should be able to view available time slots for the selected doctor.
9. **Book Appointment:**
   As a user, I should be able to book an appointment with the selected doctor.
10. **Receive Confirmation:**
    As a user, I should receive a confirmation message upon successful booking.

11. **Cancel Appointment:**
    As a user, I should be able to cancel my appointment if needed.

Admin Stories

1. **Admin Authentication:**
   As an admin, I should be able to log in, log out, and register into the application.
2. **CRUD on Users:**
   As an admin, I should be able to create, read, update, and delete user records.
3. **Manage Appointments:**
   As an admin, I should be able to allow users to book appointments.
4. **Send Confirmation:**
   As an admin, I should be able to send confirmation messages for appointments.
5. **Cancel Appointments:**
   As an admin, I should be able to cancel appointments.

System Requirements:

- **Frontend:** React-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:

1. **Frontend (React):**
   - User interface for Users and Admin.
   - Consumes Web API for login, registration, booking, and cancellation of appointments.
   - Routing for User and Admin pages (React Router).
   - Manage state using Context API or Redux.
2. **Backend (ASP.NET Core MVC):**
   - ASP.NET Core MVC with Web API endpoints for data exchange.
   - Entity Framework Core to handle database CRUD operations.
   - JWT Token Authentication for secure login/logout.
   - Web API endpoints to support user authentication, appointment booking, and management.
3. **Database (SQL Server):**
   - Tables for `Users`, `Doctors`, `Appointments`, `Ratings`, and `Specializations`.
   - Relationships: One-to-Many between Users and Appointments, Doctors and Specializations.

4. **Image Upload:**
    - Users can upload profile images, and these images are stored in a server directory.
    - File path references are stored in the database.
5. **API Documentation:**
    - Swagger UI for testing and documenting the Web API.

Sprint Plan:

Sprint I:

Objectives:

1. **Use Case Document:**
   Prepare use case documentation for all functionalities (login, registration, doctor selection, appointment management, etc.).
2. **Database Schema Design:**
   Define tables and relationships:
    - Users (UserId, Username, Password, Role)
    - Doctors (DoctorId, Name, SpecializationId, City, Rating)
    - Specializations (SpecializationId, SpecializationName)
    - Appointments (AppointmentId, UserId, DoctorId, AppointmentDate, TimeSlot, Status)
    - Ratings (RatingId, DoctorId, UserId, Rating)
3. **Select ORM Tool:**
   Use **Entity Framework Core (EF Core)** for database interactions.
4. **Backend Setup:**
    - Identify necessary controllers (UserController, DoctorController, AppointmentController).
    - Create **ASP.NET Core MVC Views** (HTML templates) as placeholders for the frontend.
5. **Static Frontend in React:**
    - Build static page templates for login, doctor search, and appointment booking in React.

Deliverables:

- Use case document
- Database schema (ERD and .sql script)
- Static views in MVC for doctor search, appointment booking, and user authentication

---

Sprint II:

Objectives:

1. **Frontend Setup (React):**
    - Initialize a React app for the user and admin interfaces.
    - Implement routing for login, register, city selection, doctor search, and appointment booking pages.
2. **Backend Implementation (ASP.NET Core MVC):**
    - Implement **CRUD operations** for users (register, login, logout).
    - Implement user authentication with **JWT Tokens**.
    - Create the **DBContext** object using EF Core.
    - Implement appointment booking feature.
3. **Doctor and Specialization Management:**
    - Create controllers and API endpoints to handle doctor and specialization data.
    - CRUD operations for doctor management (admin-side).
4. **Admin and User Pages:**
    - Design and implement the layout for user pages (React) and admin pages.
    - Connect the React frontend to the Web API (Axios or Fetch).
5. **Component Testing:**
    - Perform unit testing of individual components for frontend and backend.

Deliverables:

- React components for user and admin UI
- Web API endpoints for Users, Doctors, Appointments, and Ratings
- Working authentication for users and admin
- Component testing for React and MVC components

Sprint III:

Objectives:

1. **Search Functionality (API + Frontend):**
    - Develop the **search API** for filtering doctors based on city, specialization, and ratings.
    - Integrate the search functionality into the React frontend.
2. **Appointment Booking:**
    - Complete the appointment booking functionality (API + Frontend) for users.
    - Implement available time slot retrieval for doctors and allow users to select from those slots.
3. **Admin Appointment Module:**
    - Implement admin-side appointment approval, cancellation, and notification module.
4. **Ratings and Reviews Module:**

    - Allow users to rate doctors after their appointments.
    - Implement a doctor rating system that updates based on user feedback.
5. **Swagger API Testing:**
    - Add **Swagger UI** for API documentation and testing.
6. **Notifications (Optional):**
    - Implement real-time notifications using **SignalR** to notify users and admins about appointment bookings and cancellations.
7. **Final Integration:**
    - Ensure that all frontend components are correctly integrated with the backend API.
    - Test the entire system for bugs and performance issues.

Deliverables:

- Search and filter functionality in the API and frontend
- Doctor rating module
- Admin appointment module (approve/cancel appointments)
- Swagger API documentation for all Web API endpoints
- Fully functional React frontend consuming ASP.NET Core MVC backend API

Key Points for Implementation:

1. **Frontend:**
    - Use **React Router** for navigation between different views (login, register, book appointment, etc.).
    - Manage API requests and responses using **Axios**.
    - State management using **Context API** or **Redux**.
2. **Backend:**
    - Implement JWT-based authentication for secure user sessions.
    - Use **Entity Framework Core** for database operations (CRUD).
    - Create Web API endpoints for users, doctors, appointments, and ratings, and consume them in React.
3. **Image Uploads:**
    - Handle image uploads in the backend using **IFormFile** in ASP.NET Core.
    - Store file references in the database and serve them through the API.
4. **Admin Notifications (optional):**
    - Implement **SignalR** for real-time notifications to users and admins.