# SmartClinic - Appointment & Health Record Management System

## Project Overview

You will build a backend API for a fictional multi-branch medical clinic system called SmartClinic. This system manages doctors, patients, appointments, and medical records across different branches.

## Core Requirements

### 1. Domain Entities & Relationships (SQL & EF Core)

Model the following with foreign key relationships:

- Patient (Id, Name, DOB, Email, etc.)

- Doctor (Id, Name, Specialty, BranchId)

- Branch (Id, Location, Contact Info)

- Appointment (Id, PatientId, DoctorId, Date, Status)

- MedicalRecord (Id, AppointmentId, Notes, Diagnosis, CreatedAt)

Use Entity Framework Core Code First with SQL Server, and apply migrations.

### 2. Dependency Injection & Interfaces

- Use the repository pattern for data access with interfaces.

- Register and inject services via .NET's built-in DI container.

### 3. Controllers & API Layer (Web API)

- Create RESTful endpoints for managing:

- Patients (CRUD)

- Doctors (CRUD)

- Appointments (Create, Reschedule, Cancel)

- Medical Records (Create, View)

- Include input validation using data annotations.

### 4. Aggregator API Logic

- Doctor Availability endpoint:

- Input: DoctorId + Date

- Output: List of available time slots

- Patient History endpoint:

- Input: PatientId

- Output: All past appointments and diagnoses

## 5. Business Logic Layer (Services)

- Abstract core logic from controllers.

- Validate business rules (e.g., no double bookings).

## 6. Unit Testing (xUnit or NUnit)

- Test services and business logic using mocks.

- Cover edge cases like overlapping appointments or invalid patient IDs.

## 7. Configuration and Logging

- Use appsettings.json for configurations.

- Set up Serilog for file and console logging.

## 8. Exception Handling & Middleware

- Implement a global exception handling middleware.

- Return meaningful error messages via custom exception types.

## 9. Swagger & Documentation

- Add Swagger for interactive API docs.

- Document example payloads and responses.

## 10. Extra Stretch Goals (Optional but Valuable)

- Authentication/Authorization (JWT or cookie-based)

- Rate Limiting (e.g., 10 appointments per day per doctor)

- Caching (e.g., memory cache for branch list)

- Background Services (e.g., notify doctors of upcoming appointments)

## 11. Raw SQL Integration (Optional Advanced Features)

To strengthen your SQL skills, implement the following features using raw SQL queries via FromSqlRaw():

1. Doctor Appointment Load Summary:

- Endpoint: GET /reports/doctor-appointment-summary

- Query: Count appointments per doctor in a date range, using GROUP BY and HAVING to filter those with more than 5.

2. Branch Performance Report:

- Endpoint: GET /reports/branch-performance

- Query: Calculate average appointments per doctor in each branch using nested GROUP BY and HAVING.

3. Top Diagnoses:

- Endpoint: GET /analytics/top-diagnoses

- Query: Return diagnoses with more than 10 occurrences, sorted by count using GROUP BY and HAVING.

Use FromSqlRaw() in a dedicated ReportsService or AnalyticsService and map results to DTOs for clean output.