NET Core Web API Assignment

1. Overview & Objectives

In this extension assignment, you will transform the console-based Patient Visit Manager into a full-fledged .NET Core Web API application with HTML front-end pages. You will implement:

- User registration and login with role-based access
- All core flows from the console app exposed as RESTful endpoints
- Repository Pattern, Dependency Injection, SOLID principles, and design patterns
- ADO.NET for database communication using the previously designed schema
- HTML/CSS pages to interact with your Web API

By completing this assignment, you will gain hands-on experience building a layered, maintainable web application using modern .NET practices.

2. Assignment Requirements

Part 1: Architecture & Design

- Identify and document all API endpoints required to support:
  - Patient CRUD operations
  - Doctor CRUD operations
  - Visit scheduling and querying
  - Fee schedule retrieval
  - Activity log retrieval
  - User registration, login, and role management
- Design Controller classes mapping each endpoint to action methods.
- Define domain models and DTOs for request and response payloads.
- Create repository interfaces and implementations for each aggregate using the Repository Pattern.
- Outline dependency injection configuration for services and repositories.
- Apply SOLID principles and use at least one design pattern (e.g., Factory, Strategy, Adapter) where appropriate.

Part 2: Implementation

- Initialize a .NET Core Web API project (use the latest LTS version).

- Configure ADO.NET data access:
    - Connection string in appsettings.json
    - DbConnection and DbCommand usage in repositories
- Implement each repository method for CRUD operations and custom queries.
- Build Controllers with appropriate HTTP verbs (GET, POST, PUT, DELETE) and status codes.
- Implement authentication:
    - Secure endpoints with JWT or cookie-based auth
    - Enforce role-based authorization (e.g., Admin vs. User)
- Develop HTML/CSS pages:
    - Login and registration forms
    - Views for listing and managing Patients, Doctors, Visits
    - Navigation menu and basic styling

Part 3: Testing & Documentation

- Provide a Postman collection (or equivalent) demonstrating all API calls and authentication flow. Don't know what postman is? Don't worry you can learn about it from here: https://www.youtube.com/watch?v=A36VQFdIAkI
- Write a README with:
    - Setup instructions (database creation, running migrations)
    - How to launch the Web API and HTML front-end
    - Description of design patterns used and DI setup

3. File & Branch Organization

- Git Branch: **project-01-webapi**
- Solution Structure:
    - Controllers/
    - Models/ (Domain + DTO)
    - Repositories/Interfaces and Implementations
    - Services/ (business logic if separated)
    - Data/ (ADO.NET setup / Simply your DbContext)
    - wwwroot/ (HTML, CSS files)
- Include comments at the top of each file with your name and date.

4. Step-by-Step Guidelines

1. Plan your API surface by mapping console app commands to REST endpoints.
2. Scaffold the Web API project and configure DI in Startup/Program.
3. Implement ADO.NET repositories and test data access methods.
4. Build Controllers.
5. Create HTML views and AJAX calls or form posts to your API.
6. Test all flows with Postman and through the UI pages.
7. Document your setup and design decisions in the README.

5. Submission Guidelines

- Deadline: **Friday, August 15th** (deliver when you reach the office before 11:00 AM)
- Deliverables:
    - Code on Git branch
    - Postman collection JSON
    - README and Front-End files included in wwwroot
- Ensure your solution runs without errors and UI pages can perform all operations.

6. Grading Criteria

- API Design & Architecture 35%
- ADO.NET Implementation & DI 25%
- HTML UI & Integration 25%
- Documentation & Testing (Exception Handling) 15%

Extension Tasks

1. Authentication and Authorization

- Add persistence for users and roles (tables or reuse existing) with unique email constraint.
- Store passwords as hashed + salted values (no plaintext).
- Implement endpoint POST **/api/auth/register** to create a user.
- Implement endpoint POST **/api/auth/login** to return a JWT access token containing user role/claims.
- Implement endpoint POST **/api/auth/change-password** for authenticated users.

- Enforce token expiry and deny access with expired/invalid tokens.
- Define roles: at minimum **Admin, Doctor, Receptionist**.
- Create authorization policies (**e.g., RequireAdmin, RequireDoctorOrAdmin, RequireReceptionistOrAdmin**).
- Protect endpoints using [Authorize] + policies
- Return **401** for unauthenticated and **403** for unauthorized requests.

2. Middlewares

- Implement Exception Handling Middleware that returns JSON ProblemDetails (type, title, status, traceId, optional detail).
- Implement Request Logging + Timing Middleware that records method, path, user (if any), status code, and elapsed ms.

3. Validation (FluentValidation) & Filters:

- Add FluentValidation validators for DTOs used by your endpoints
- Create a Validation Filter that returns 400 with a unified error payload when validation fails.

4. Logging:

- Use Serilog to log error response that are being caught by your exception handling middleware and your general logs on a text file.

NOTE: Deadline of this assignment is **Monday 18th of August, 2025 (Before 10.30 am)**