1. The Project

Inside the project folder you will find the folders **PatientManagerWebApi** and **patient-manager-app**. The first will have the Web API source code and the last will have the Front End source code.

The Web API was developed with .NET, ASP.NET for the HTTP Rest API and the Entity Framework to manipulate data in a Relational Database. Inside the Web API folder, you will find the following folders:

- **Controller**: This folder will contain classes responsible to handle the HTTP requests.
- **Request**: This folder will contain the classes used to restrict what data will enter and leave the system through the controllers.
- **Model**: This folder will contain classes that represent entities related to business logic.
- **Error**: This folder will contain classes that will deal with errors in the API.
- **Data**: This folder will have classes responsible to connect to the database and execute CRUD operations. There are also interfaces in it to ensure different implementations offering a similar functionality can be provided, which is an important feature because it allows you to replace, for example, the Entity Framework used in the current implementations with ADO.NET easily. To do that, you only have to provide an implementation that uses ADO.NET instead of the Entity Framework and register it as a dependency to be satisfied, check Program.cs to see how you can do that.
- **Service**: This folder will contain classes that will provide the API functionalities related to business logic. There are also interfaces in it, allowing different implementations to be provided, again offering some change flexibility.

The Front End was developed with Angular. Inside the Front End folder, there is an **src** folder containing the Front End source code. In the **src** folder you will find the folders:

- **Model:** This folder will contain classes that represent entities related to business logic.
- **Util**: This folder will contain classes that has some utility functionality.
- **Service**: This folder will contain classes that will provide the API functionalities related to business logic.
- **App**: This folder will contain the Angular Components used to build the user interface.

2. Project Goals

1. Authentication & Authorization:

- Implement secure login functionality. (Couldn't finish it on time)
- Role-based access control (Roles are there, but without authentication)
  (e.g., Admin, Healthcare Professional).

2. Patient Management:

- Display a list of patients with pagination and filtering options. (Done)
- Search functionality by name or ID. (Done)
- View detailed patient information. (Done)

3. Recommendations:

- Display a list of recommendations (Allergy check, screenings, follow-ups) associated with a patient. (Done)
- Allow users to mark recommendations as completed. (Done)

4. Security Considerations:

- Implement OWASP security best practices (CSP, anti-CSRF, secure headers). (Couldn't finish authentication on time)
- Protect against SQL Injection, XSS. (Done)
- Protected authentication-related attacks. (Couldn't finish authentication on time)

5. Technical Requirements:

- Backend: .NET Core 6+ Web API. (Done)
- Frontend: Angular 17+ (Done)
- Use Angular Material Components. (Couldn't finish it on time)
- Database: MS SQL Server. (I was using Supabase Cloud Postgre only for testing because of the free plan, I couldn't switch it to MS SQL Server on time)
- Entity Framework is optional (Done)
- API should follow RESTful principles. (Done)

6. Infrastructure:

- Deployment-ready configuration (Docker support is a plus). (Couldn't finish it on time)
- Local development setup instructions. (Done)

Improvements To Be Done

1. Web API:

- Improve the Entities, some like patient and user are collecting just symbolic data.
- Make the user's role a table in the database instead of a hardcoded Enum.
- Save user password with encryption in the database.
- Better error handling and better error responses to the user.
- Code user authentication and authorization.
- Unit testing.

2. Front End:

- Change Path Parameters to Query Parameters ???
- Better error handling and better error response to the user.
- Code user authentication.
- Lock some features based on the role of the logged user.
- Unit testing.