EPQ Project Documentation

# 1. Introduction

This project is a web-based system developed using ASP.NET Core MVC with C# , Repository Pattern , Entity Framework Core, and SQL Server as the primary database. The system also utilizes front-end technologies including jQuery, JavaScript, and HTML/CSS for the user interface. The application was designed following the Clean Architecture principles to ensure scalability, maintainability, and separation of concerns.

# 2. Technologies Used

- C# & .NET 8

- ASP.NET Core MVC (C#) – for backend and controllers

- Entity Framework Core – for database interactions (ORM)

- SQL Server – relational database management system

- Repository Pattern

- LINQ – querying data from collections and EF

- jQuery – simplifying DOM manipulations and AJAX calls

- JavaScript – client-side logic

- HTML5 / CSS3 – front-end presentation

# 3. Important Problems and Solutions

## 3.1 Large Data Retrieval (Account History)

Problem: When retrieving account history with a large dataset, the system was facing performance issues and slow response times due to Entity Framework tracking all entities.

Solution: The issue was solved by using AsNoTracking to prevent EF Core from tracking the entities, which improved performance significantly.

Code Example:

var history = \_context.AccountHistories  
 .AsNoTracking()  
 .Where(a => a.AccountId == accountId)  
 .ToList();

## 3.2 Handling Large Result Sets with Pagination

Problem: Loading all records at once in the front-end (tables, grids) caused UI freezing and long rendering times.

Solution: Pagination was implemented on both backend and frontend, ensuring only a limited subset of data is fetched and displayed at a time.

Code Example:

var pageSize = 20;  
var data = \_context.AccountHistories  
 .AsNoTracking()  
 .OrderByDescending(a => a.Date)  
 .Skip((pageNumber - 1) \* pageSize)  
 .Take(pageSize)  
 .ToList();

## 3.3 Handling Extra Spaces in Strings

Problem: Data retrieved from the database contained unnecessary spaces before and after text, causing UI inconsistencies.

Solution: The .trim() function was used in JavaScript/jQuery, and (.Trim() & Regex) was used in C# to clean up the strings.

Examples:

C#:

var text = Regex.Replace(row.Description.ToString(), @"^(<br\s\*/?>\s\*)+","");

text = Regex.Replace(text, @"(\s\*<br\s\*/?>)+$", "");

row.Description = text;

JavaScript: var cleanText = text.trim();

**Best regards,**

# Mohamed Hammad