

Лабораторна робота №1

Звіт

З дисципліни “Сучасні інтернет технології”
на тему: “Структура проекту ASP.NET Core Веб-застосунку.
Розділення моноліту на модулі”.

Студента 4 курсу: Групи МІТ-41
Демиденко Андрій

Київ - 2024р.

Хід виконання роботи

Тема: Структура проекту ASP.NET Core Веб-застосунку. Розділення моноліту на модулі (шари).

Мета: Набути практичних навичок зі створення вебзастосунків ASP.NET Core з використанням архітектурного шаблону MVC. Навчитися реалізовувати автентифікацію, розширювати стандартні моделі даних та проводити рефакторинг проєкту шляхом розділення моноліту на окремі логічні модулі (шари). Опанувати механізм міграцій Entity Framework Core.

Створення проєкту ASP.NET Core

Робота розпочалася зі створення нового рішення у Visual Studio. Було обрано шаблон ASP.NET Core Web App (Model-View-Controller), який сразу створює базову структуру проєкту з папками для Models, Views та Controllers.

Під час налаштування проєкту ключовим кроком був вибір типу автентифікації. Було обрано Individual Accounts (Облікові записи окремих користувачів). Цей вибір є фундаментальним, оскільки він автоматично інтегрує в проєкт фреймворк ASP.NET Core Identity, додаючи необхідні класи (IdentityUser, ApplicationDbContext) та налаштування для реєстрації, входу та керування користувачами (рис.1).

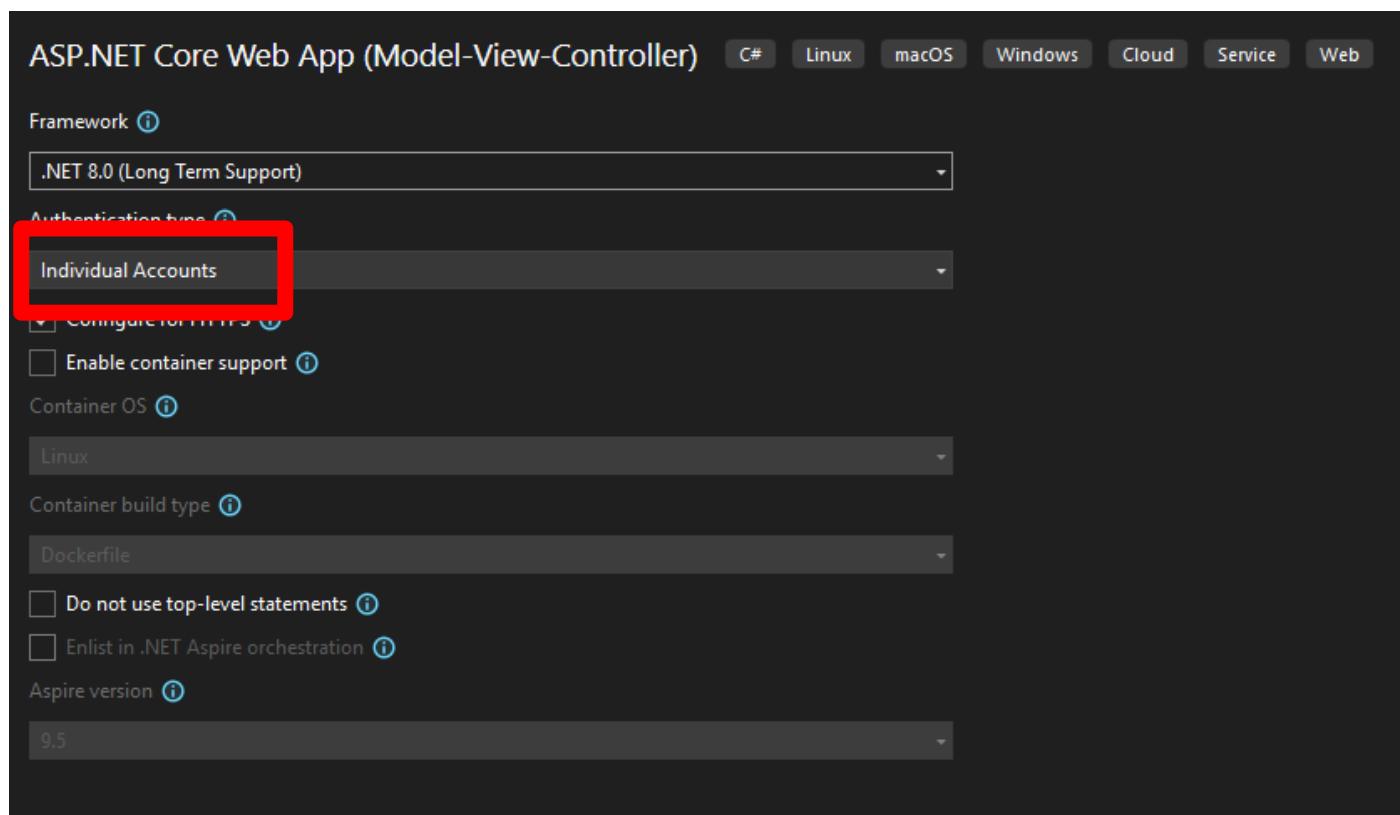


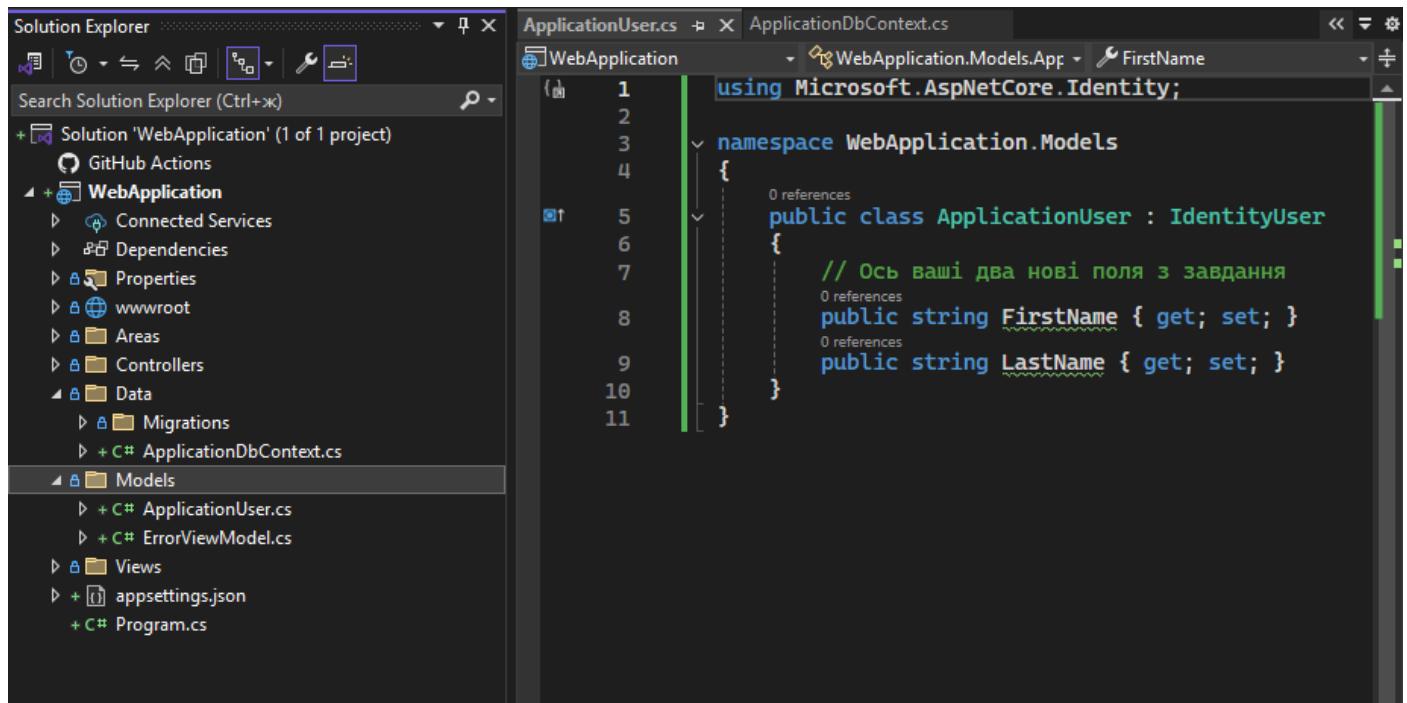
Рис. 1 – ASP.NET Core Web App (Model-View-Controller) з Individual Accounts

Розширення класу автентифікованого користувача

Стандартний клас `IdentityUser`, який надається системою, містить лише базову інформацію (email, ім'я користувача, хеш пароля). Для реального застосунку цього недостатньо. Завдання полягало в розширенні цієї моделі.

Процес складався з трьох частин:

- Створення `ApplicationUser`:** У папці `Models` було створено новий клас `ApplicationUser`, який успадковує `IdentityUser`. До цього класу було додано дві нові властивості: `FirstName` та `LastName` (рис.2).

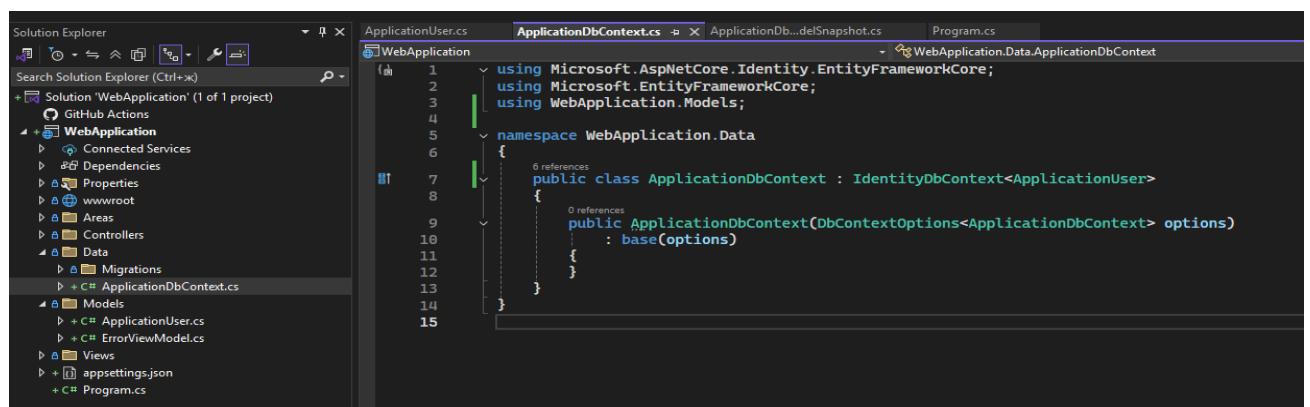


The screenshot shows the Visual Studio interface with the Solution Explorer on the left and the code editor on the right. In the Solution Explorer, under the 'WebApplication' project, the 'Models' folder is selected. In the code editor, the 'ApplicationUser.cs' file is open, showing the following C# code:

```
using Microsoft.AspNetCore.Identity;
namespace WebApplication.Models
{
    public class ApplicationUser : IdentityUser
    {
        // Ось ваші два нові поля з завдання
        public string FirstName { get; set; }
        public string LastName { get; set; }
    }
}
```

Рис. 2 – Створення `ApplicationUser`.

- Оновлення `ApplicationDbContext`:** Стандартний `DbContext` було модифіковано, щоб він успадковував `IdentityDbContext< ApplicationUser>`. Це повідомило Entity Framework про те, що тепер він має працювати з нашою розшиrenoю моделлю користувача (рис.3).



The screenshot shows the Visual Studio interface with the Solution Explorer on the left and the code editor on the right. In the Solution Explorer, under the 'WebApplication' project, the 'Data' folder is selected. In the code editor, the 'ApplicationDbContext.cs' file is open, showing the following C# code:

```
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using WebApplication.Data;
namespace WebApplication.Data
{
    public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
    {
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
            : base(options)
        {
        }
    }
}
```

Рис. 3 – Оновлення `ApplicationDbContext`.

Ці кроки забезпечили повну інтеграцію нової моделі користувача в систему

автентифікації та в шар доступу до даних Individual Accounts.

Розділення моноліту на модулі

Початковий проект мав монолітну структуру, де логіка UI (Views, Controllers) була змішана з логікою даних (Models, Data). Для покращення архітектури було проведено рефакторинг до **модульного моноліту**.

Створення Шару Даних: У поточному рішенні (Solution) було створено новий проект типу **Class Library** (Бібліотека класів) і названо його WebApplication.Data. Цей проект призначений для інкапсуляції всієї логіки, пов'язаної з даними. Папки Data (з ApplicationDbContext) та Models (з ApplicationUser та ErrorViewModel) були фізично переміщені з основного вебпроекту до WebApplication.Data (рис. 4).

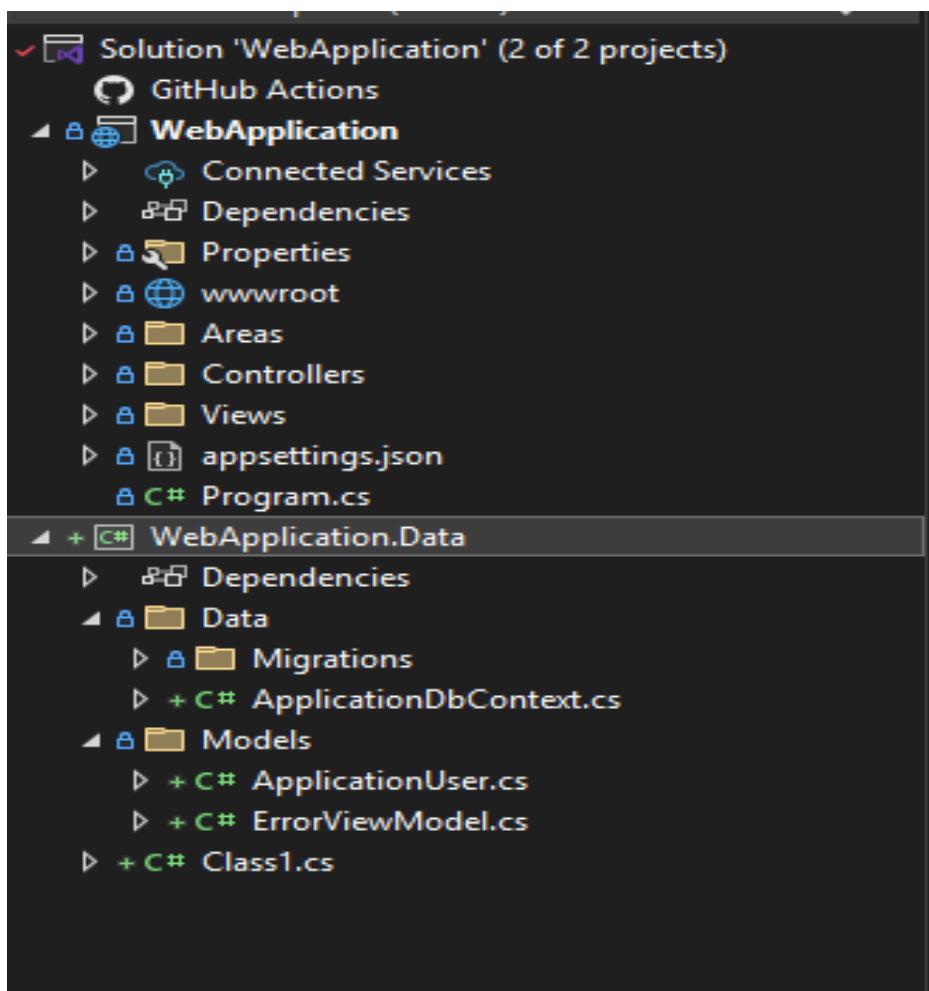


Рис. 4 – Створення Шару Даних Class Library.

Оновлення Просторів Імен: У всіх переміщених файлах (.cs) та у файлах основного проекту (Program.cs, _ViewImports.cshtml), які на них посилялися, було оновлено простори імен (namespace) та директиви using, щоб вони відповідали новій двопроектній структурі (рис. 5).

```

    using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
    using Microsoft.EntityFrameworkCore;
    using WebApplication.Models;

    namespace WebApplication.Data
    {
        4 references
        public class ApplicationDbContext : IdentityDbContext< ApplicationUser >
        {
            0 references
            public ApplicationDbContext(DbContextOptions< ApplicationDbContext > options)
                : base(options)
            {
            }
        }
    }

```

Рис. 5 – Оновлення Просторів Імен.

Виконання міграції бази даних

Після того, як модель користувача була розширенна (Завдання 2) і проект був реструктуризований (Завдання 3), необхідно було синхронізувати ці зміни з фізичною базою даних (рис. 6).

1. У **Package Manager Console** було обрано основний вебпроект (WebApplication) як "Default project", оскільки він містить appsettings.json з рядком підключення.
2. Для коректної роботи міграцій у багатопроєктній архітектурі, конфігурацію DbContext у Program.cs було доповнено, вказавши, де знаходитьться збірка міграцій: `b => b.MigrationsAssembly("WebApplication")`.
3. Було виконано команду `Add-Migration AddedFirstNameLastNameToUser`. Entity Framework успішно проаналізував різницю між кодом (де вже є FirstName) та базою даних (де їх ще не було) і згенерував скрипт міграції.
4. Командою `Update-Database` згенерований скрипт було застосовано до SQL Server.

```

PM> Add-Migration AddedFirstNameLastNameToUser
Build started...
Build failed.
PM> Add-Migration AddedFirstNameLastNameToUser
Build started...
Build succeeded.
Your target project 'WebApplication' doesn't match your migrations assembly 'WebApplication.Data'. Either change your target project or change your migrations assembly.
Change your migrations assembly by using DbContextOptionsBuilder. E.g. options.UseSqlServer(connectionString, b => b.MigrationsAssembly("WebApplication")). By default, the migrations assembly is the assembly containing the DbContext.
Change your target project to the migrations project by using the Package Manager Console's Default project drop-down list, or by executing "dotnet ef" from the directory containing the migrations project.
PM> Add-Migration AddedFirstNameLastNameToUser
Build started...
Build succeeded.
To undo this action, use Remove-Migration.
PM> Update-Database
Build started...
Build succeeded.
Microsoft.EntityFrameworkCore.Database.Command[20101]
  Created DbCommand (7ms) [Parameters=[], CommandType='Text', CommandTimeout='60']
    CREATE DATABASE [aspnet-WebApplication-2c887581-c669-4216-a240-bcc2e69f0381];
Microsoft.EntityFrameworkCore.Database.Command[20101]
  Executed DbCommand (7ms) [Parameters=[], CommandType='Text', CommandTimeout='60']
    IF SERVERPROPERTY('EngineEdition') <> 5
    BEGIN

```

Рис. 6 – Виконання міграції бази даних.

Превірка Міграції за допомогою SQL Server Object Explorer

Перевірка: За допомогою **SQL Server Object Explorer** було перевірено структуру таблиці dbo.AspNetUsers. Перевірка підтвердила успішне додавання нових стовпців FirstName та LastName. Висновок (рис. 7).

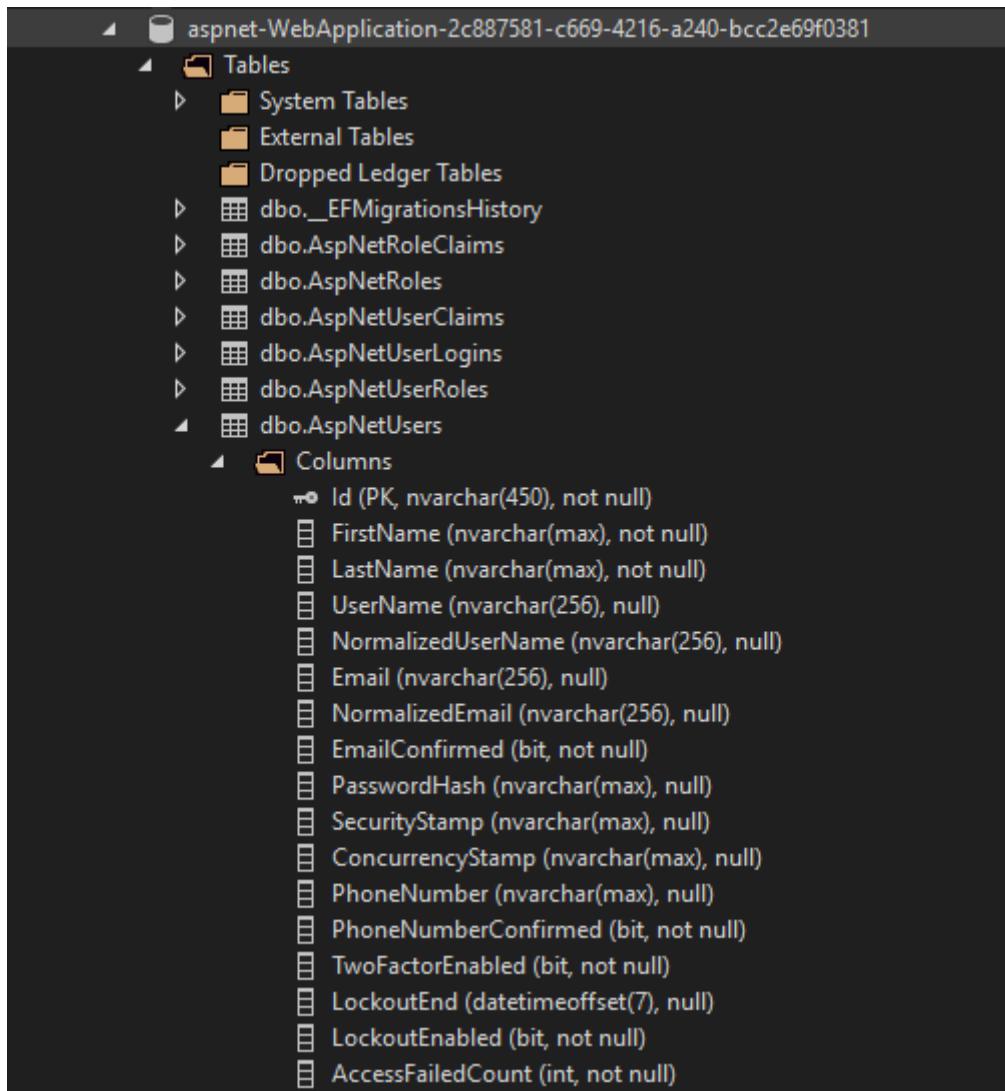


Рис. 7 – Превірка Міграції за допомогою SQL Server Object Explorer

Висновок:

Під час виконання цієї лабораторної роботи я успішно створив вебзастосунок ASP.NET Core MVC з повноцінною системою автентифікації.

Я здобув практичні навички з кастомізації ASP.NET Core Identity шляхом розширення стандартної моделі IdentityUser, що є необхідною вимогою для більшості реальних проектів.

Найважливішим результатом роботи стало розуміння та практична реалізація архітектурного принципу "Розділення відповідальності" (Separation of Concerns). Шляхом рефакторингу монолітного проекту в модульний моноліт (винесення логіки даних в окрему бібліотеку класів), я покращив структуру проекту, зробивши її більш організованою, легкою для підтримки та масштабування. Такий підхід дозволяє в майбутньому легко підключити цей шар даних до інших типів застосунків (наприклад, мобільного клієнта), не дублюючи код.

Також я на практиці опанував механізм Entity Framework Core Migrations як інструмент для синхронізації C# моделей ("креслень") та реальної структури бази даних ("будівлі"), і навчився керувати цим процесом у багатопроектному рішенні.