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 одразу 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**)**.

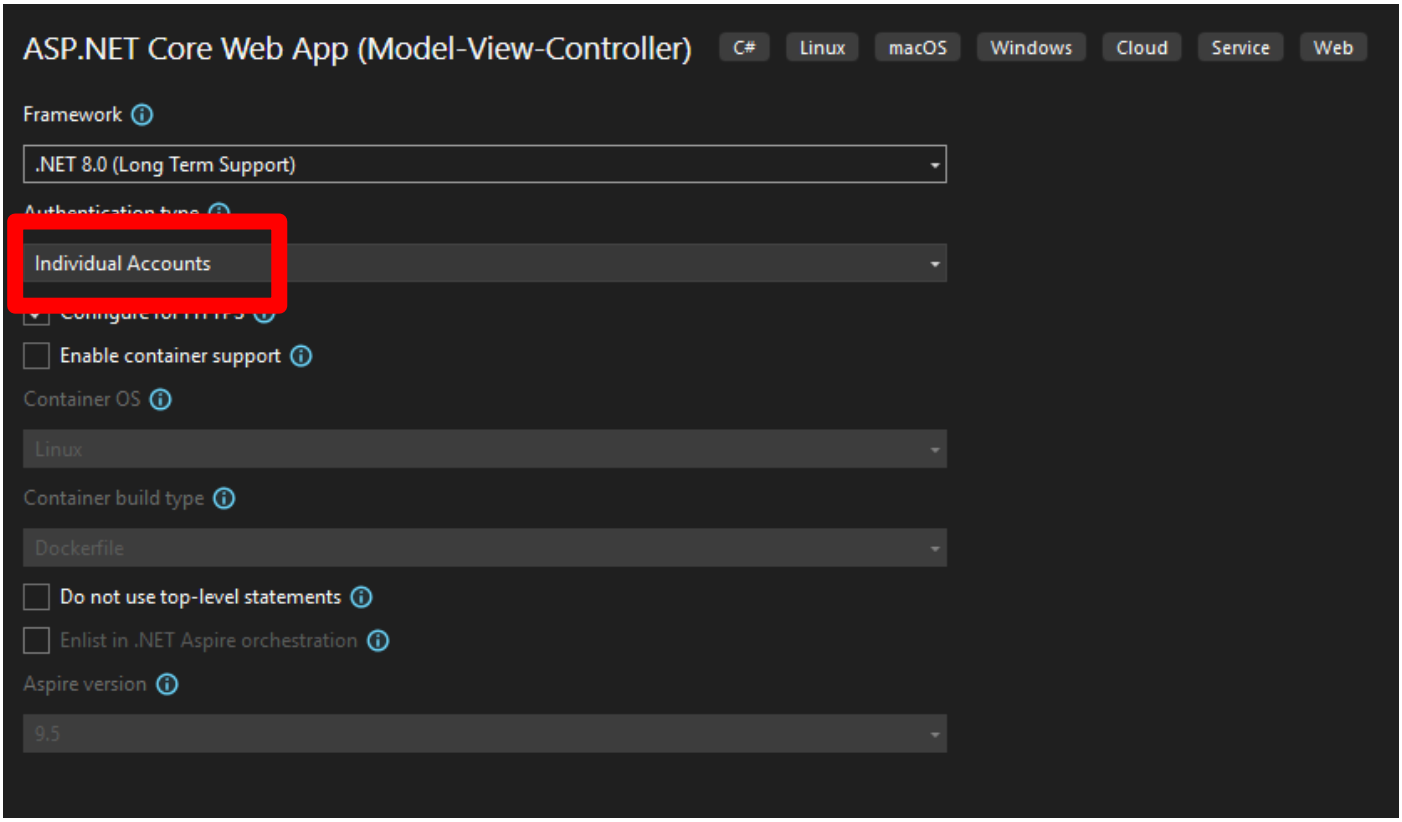

Рис. 1 – ASP.NET Core Web App (Model-View-Controller) з Individual Accounts

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:

1. **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**)**.

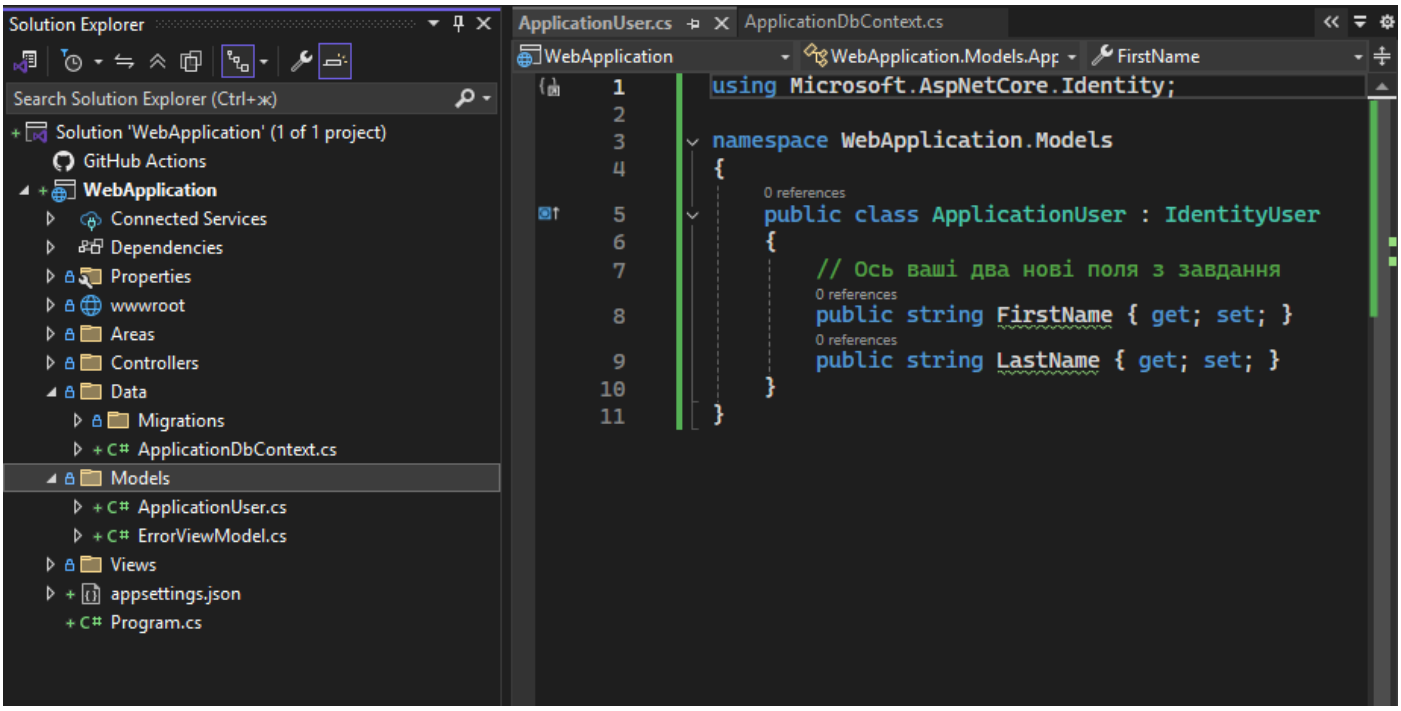

Рис. 2 – Створення ApplicationUser.

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

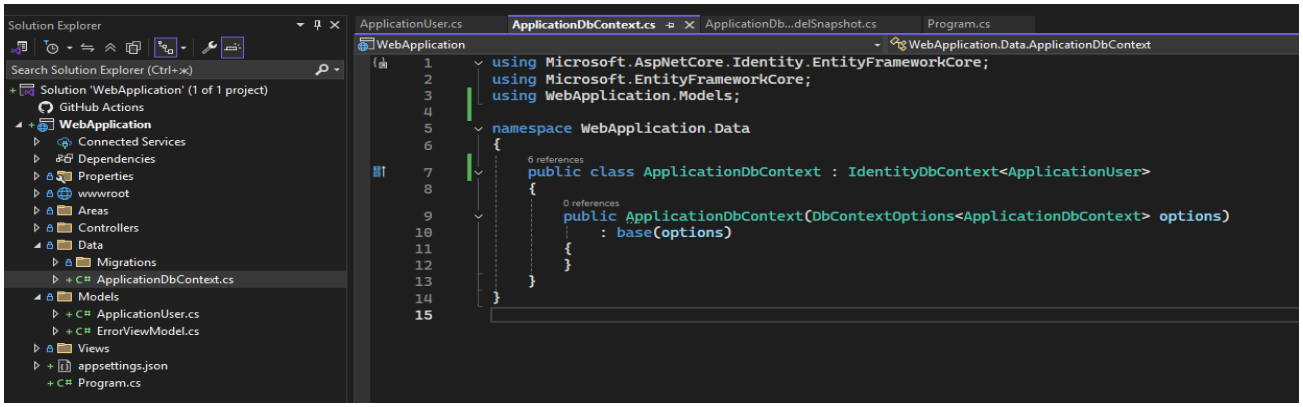

Рис. 3 – Оновлення ApplicationDbContext.

Ці кроки забезпечили повну інтеграцію нової моделі користувача в систему

автентифікації та в шар доступу до даних Individual Accounts.

Розділення моноліту на модулі

Початковий проєкт 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 проєкт типу **Class Library** (Бібліотека класів) і названо його WebApplication.Data. Цей проєкт призначений для інкапсуляції всієї логіки, пов'язаної з даними. Папки Data (з ApplicationDbContext) та Models (з ApplicationUser та ErrorViewModel) були фізично переміщені з основного вебпроєкту до WebApplication.Data (рис. 4).

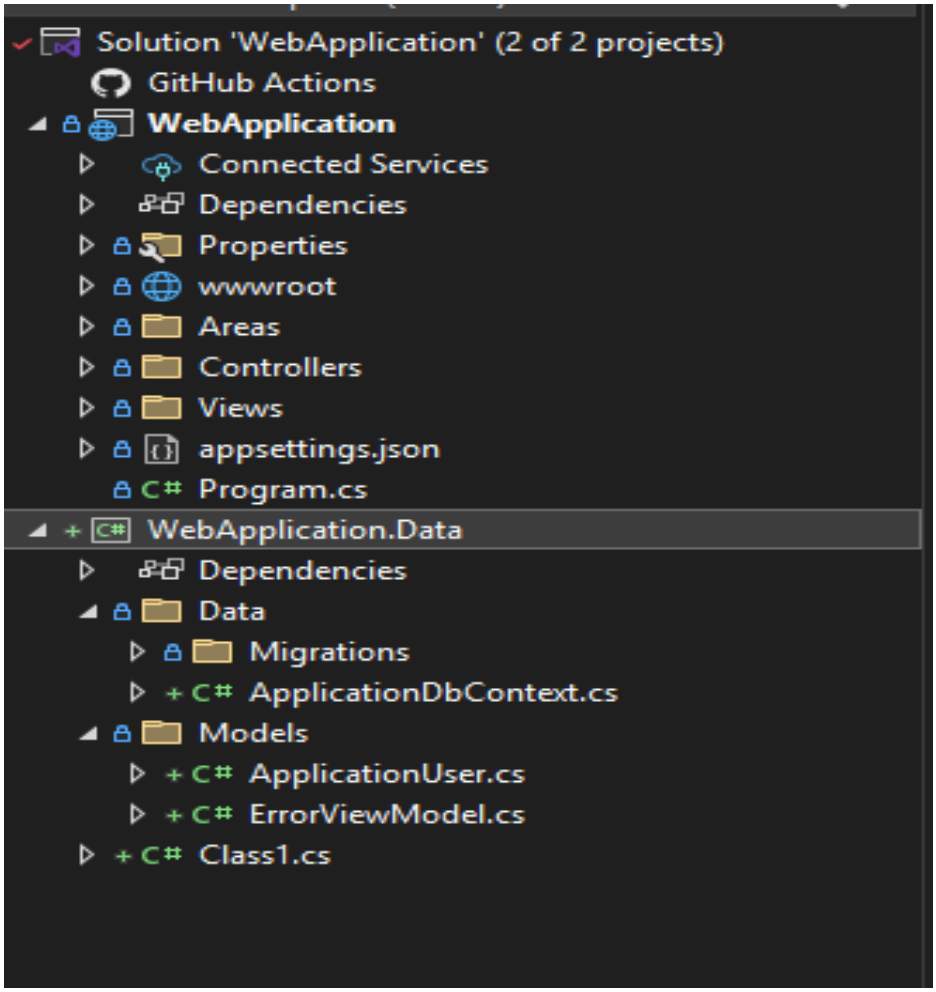

Рис. 4 – Створення Шару Даних Class Library.

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

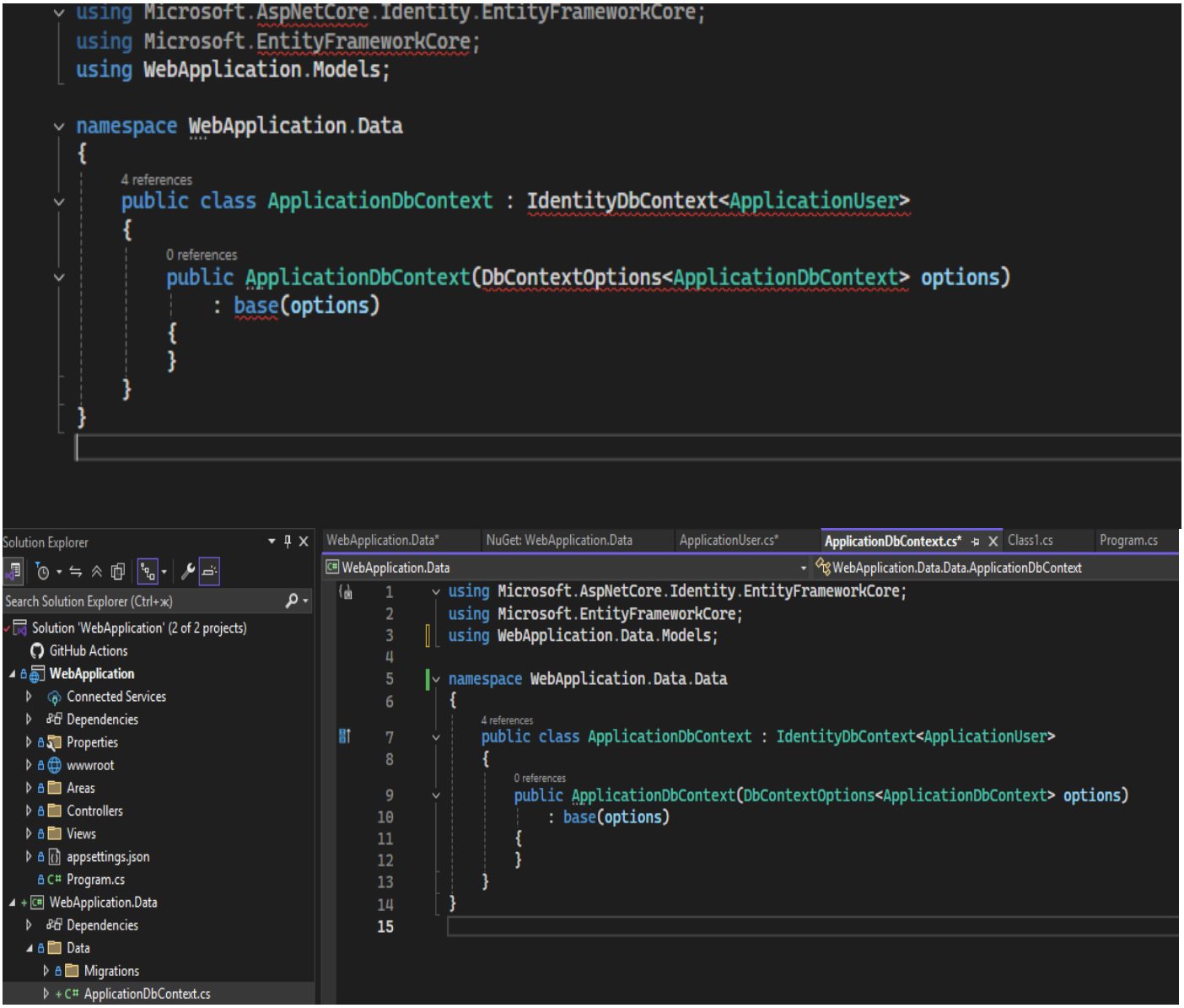

Рис. 5 – Оновлення Просторів Імен.

Виконання міграції бази даних

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

1. У **Package Manager Console** було обрано основний вебпроєкт (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 знаходиться 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.

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).

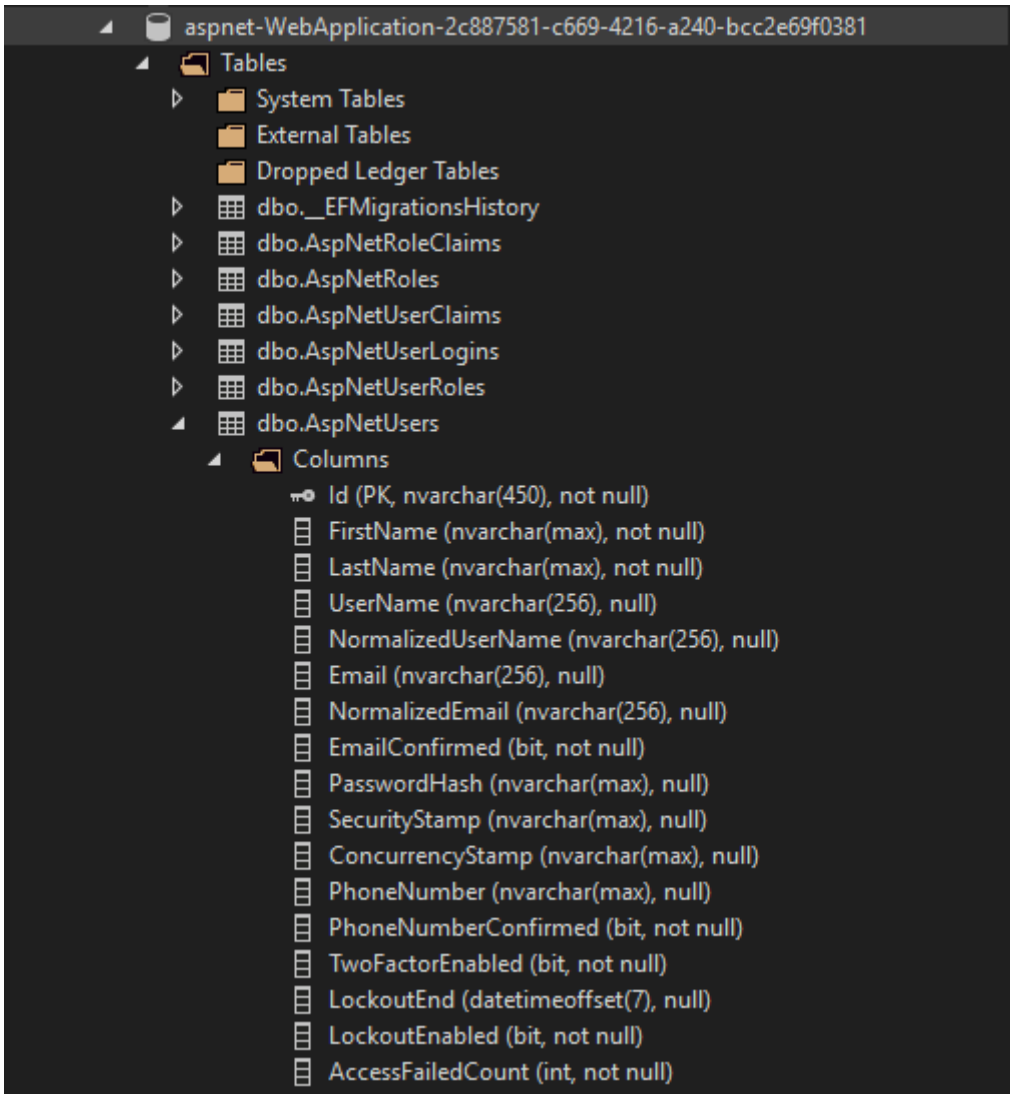

Рис. 7 – Превірка Міграції за допомогою SQL Server Object Explorer

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 проєктів.
Найважливішим результатом роботи стало розуміння та практична реалізація архітектурного принципу "Розділення відповідальності" (Separation of Concerns). Шляхом рефакторингу монолітного проєкту в модульний моноліт (винесення логіки даних в окрему бібліотеку класів), я покращив структуру проєкту, 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 багатопроєктному рішенні.