ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА

Факультет прикладної математики та інформатики

Програмна інженерія

Лабораторна робота №4

**Тема: «Розробка та затвердження архітектури проекту»**

Виконали:

студентки групи ПМІ-32

Бандурист Юліана

Бурдяк Олена

Середня Ірина

Стасишин Юлія

Чушак Христина

Прийняв:

ас. Галамага Л. Б.

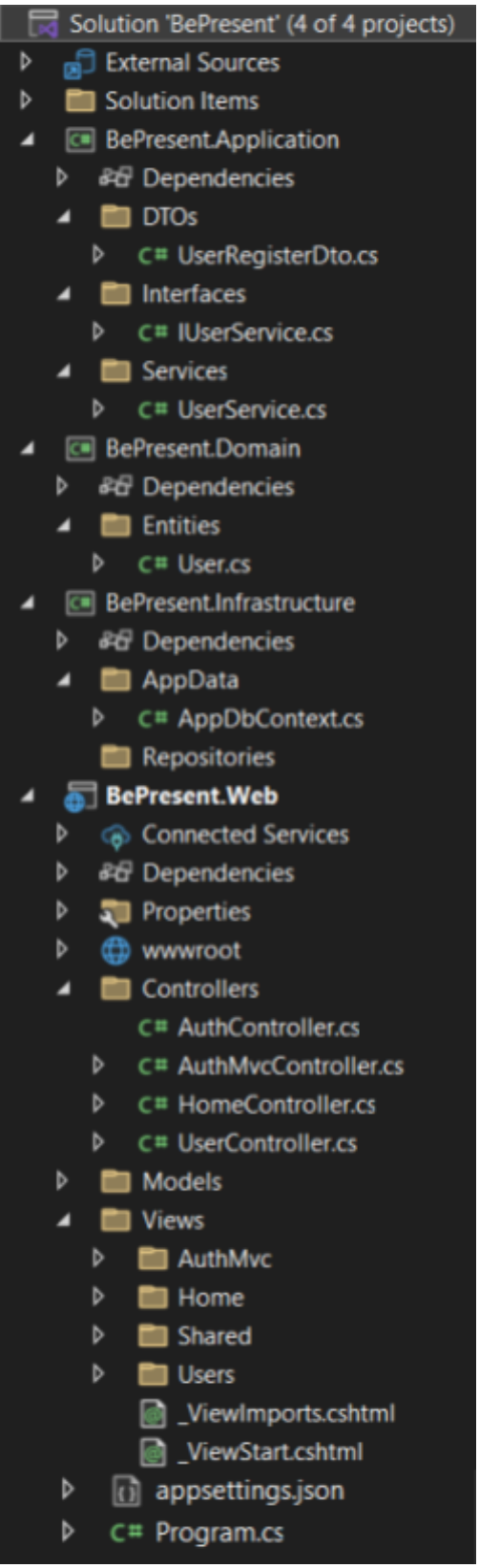
Львів 2025

**Тема :** Розробка та затвердження архітектури проекту

1. **Знання та вибір підходу до побудови архітектури проекту**

Для реалізації нашого проекту ми обрали саме Layered architecture,оскільки вона задовільняє потреби нашого проекту,також у нас вже наявний досвід у створенні застосунків саме такої архітектури,що нам і допоможе у створенні додатку.

2. **Побудова базових архітектурних структур в рамках вибраного архітектурного підходу та платформи ASP .NET Core MVC.**




**1. BePresent.Web (UI - Presentation Layer)**

**Відповідальність:** Шар для взаємодії з користувачем. Містить контролери (MVC), які обробляють HTTP-запити та повертають відповіді у вигляді HTML-сторінок або API-відповідей.

**Основні компоненти:** Контролери, представлення (Views), маршрутизація, налаштування представлення UI.

**2. BePresent.Application (Business Logic - Application Layer)**

**Відповідальність:** Реалізація бізнес-логіки. Сервіси, які обробляють запити, виконують бізнес-правила та взаємодіють із доменними моделями.

**Основні компоненти:** Сервіси, бізнес-логіка, передача даних (DTOs), обробка запитів, що надходять з Web-шару, і координація взаємодії з іншими шарами.

**3. BePresent.Domain (Core - Domain Layer)**

**Відповідальність:** Шар, що містить доменні моделі (сутності), які відображають структуру даних і логіку додатку. Це "серце" вашого проекту.

**Основні компоненти:** Сутності, бізнес-правила, ентиті, які відображають таблиці бази даних.

**4. BePresent.Infrastructure (Data Access - Infrastructure Layer)**

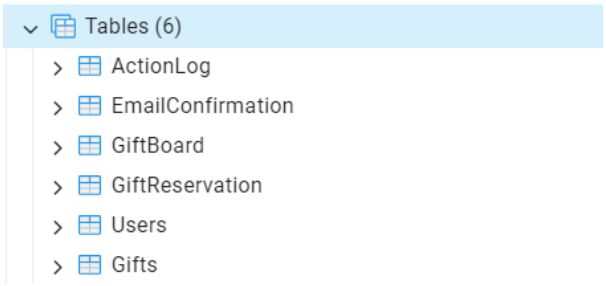
**Відповідальність:** Шар для доступу до даних. Використовує Entity Framework для роботи з базою даних, налаштування з’єднання з БД та реалізація репозиторіїв для доступу до даних.

**Основні компоненти:** DbContext (наприклад, AppDbContext), реалізація репозиторіїв, налаштування з’єднання з базою даних (PostgreSQL або інша БД).

## 3. Створення бази даних

- PostgreSQL

Далі ми відповідно до обраних сутностей створили базу даних BePresent з наступними таблицями. У проекті ми вказуємо connection string у appsettings.json




**Users**: Зберігає дані користувачів — унікальне ім'я, електронну пошту, пароль, дату народження, стать, інтереси та статус авторизації.

**GiftBoards**: Містить інформацію про дошки подарунків, зокрема ім'я, дату святкування, доступність (публічний/лише для друзів/приватний), опис та авторів.

**Gifts**: Зберігає інформацію про подарунки на дошках, включаючи назву, опис, посилання, зображення та статус резервування.

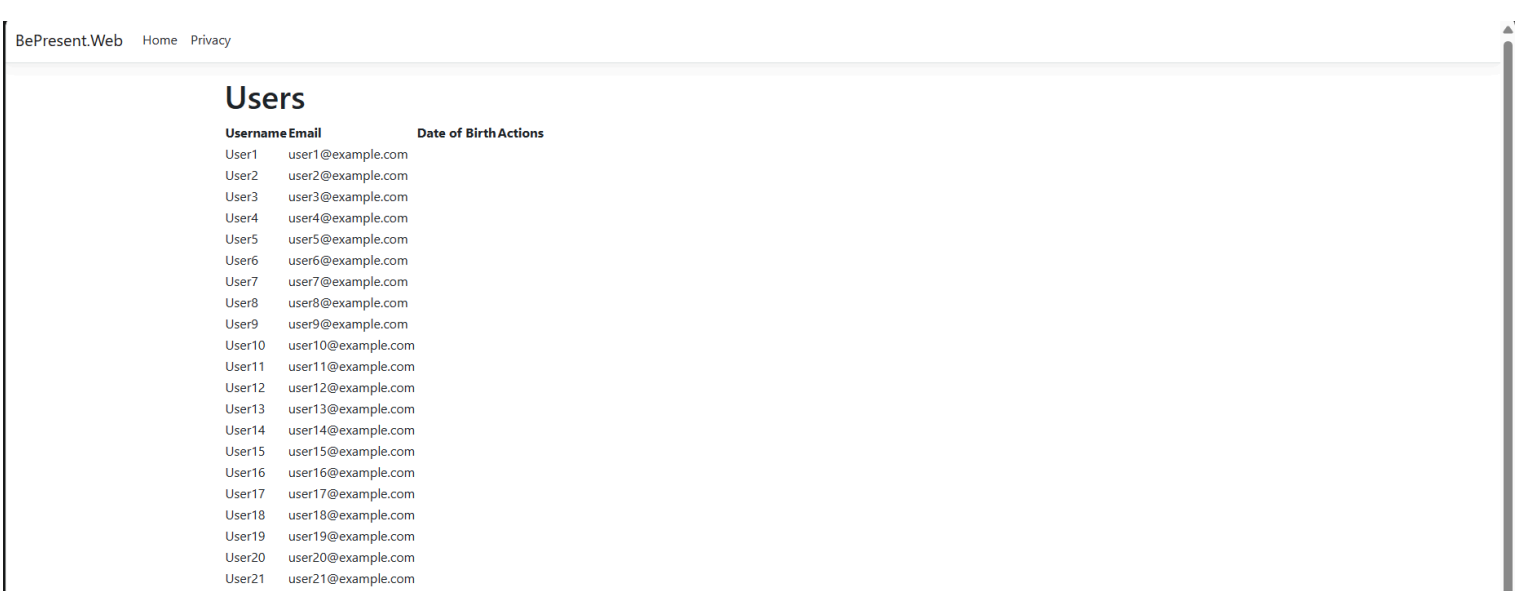
**GiftReservations**: Реєструє резервування подарунків користувачами, вказуючи подарунок, користувача та час резервування.

**ActionLogs**: Логує дії користувачів для аудиту, зберігаючи інформацію про користувача, тип дії та час.

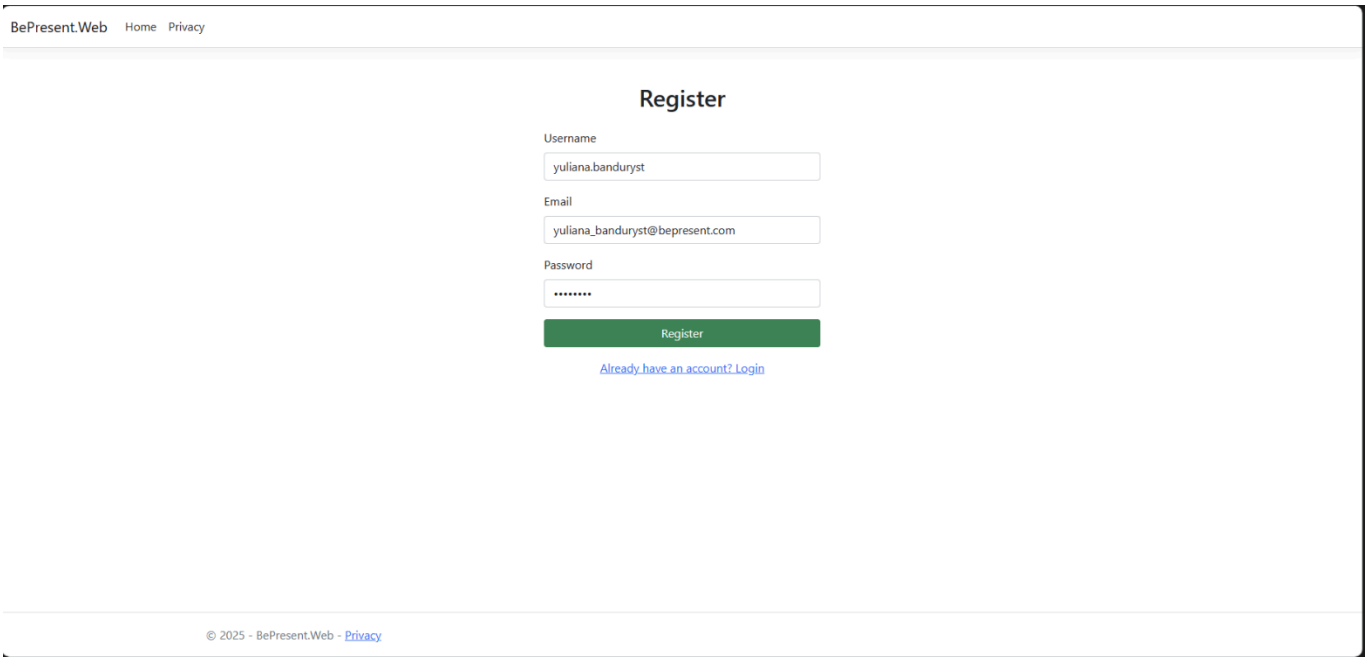
**EmailConfirmations**: Зберігає підтвердження електронної пошти для користувачів з токеном підтвердження та часом, коли підтвердження має закінчитися.

## 4. Підключення Entity Framework для роботи з базою даних

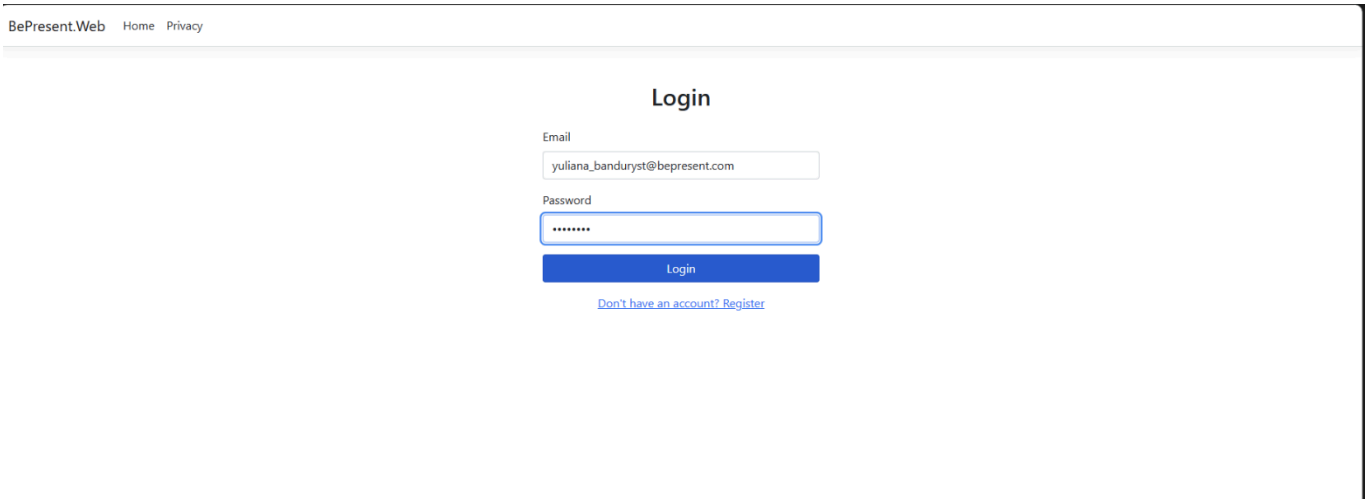
Ось приклад як наш застосунок взаємодіє з базою даних через EntityFramework



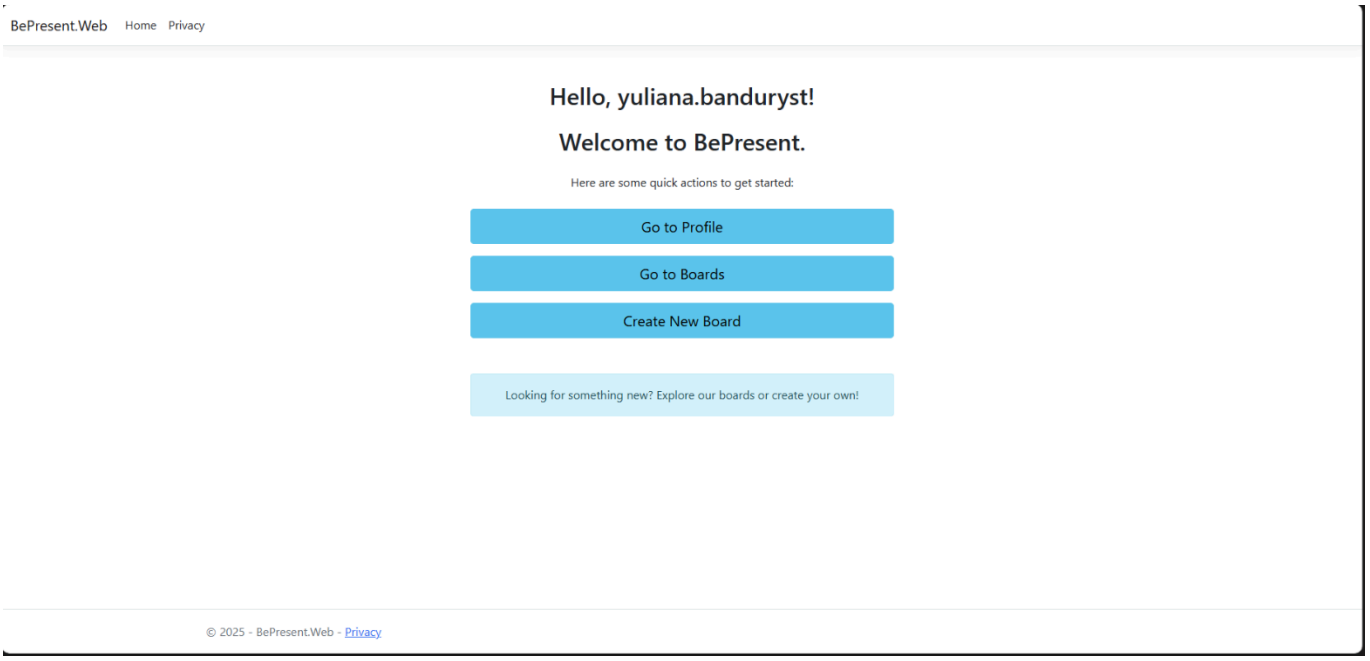
Далі ми реалізували логін/реєстрацію для користувачів і основну сторінку




(початкову версію)



Далі реалізували основну сторінку,де ми вітаємо користувача який залогінився і презентуємо кнопки з основними функціями




5. **Налаштування одного чи декількох аналізаторів коду**
    - **Roslynator**

**У рамках покращення якості коду було вирішено використати статичний аналізатор Roslynator. Цей інструмент є набором C#-аналізаторів і кодогенераторів, побудованих на основі платформи Roslyn — офіційного рушія компіляції в .NET. Roslynator дозволяє автоматично виявляти помилки, стилістичні недоліки, а також пропонує можливості рефакторингу.**

**Переваги Roslynator:**

- **Простота інтеграції з проєктами .NET через NuGet;**
- **Підтримка конфігурації через .editorconfig;**
- **Висока швидкість аналізу коду;**
- **Підказки для оптимізації синтаксису, видалення зайвого коду, покращення читабельності тощо.**

**Кроки налаштування:**

1. **Додавання пакета через NuGet:**

**Аналізатор було встановлено за допомогою команди:**

**dotnet add package Roslynator.Analyzers**

**Ця команда додає посилання на бібліотеку аналізатора у вибраний .csproj файл проєкту.**

**2.Створення та налаштування .editorconfig:**

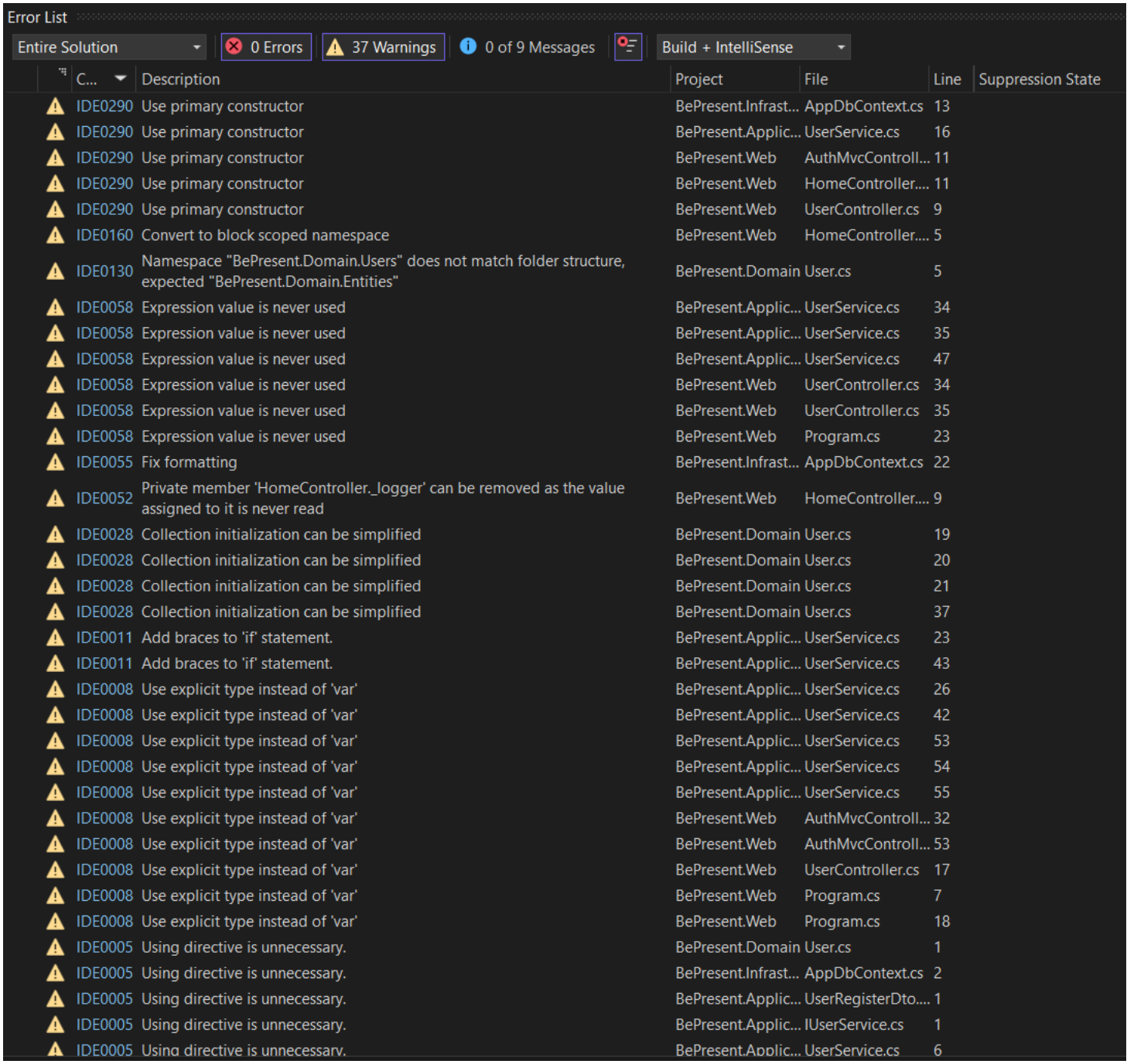
**Для тонкого налаштування правил був створений файл .editorconfig у кореневій директорії рішення. Приклад налаштувань:**

```
root = true

[*.cs]

dotnet_diagnostic.IDE0005.severity = warning

dotnet_analyzer_diagnostic.category-Style.severity = suggestion
```

**Ці правила визначають рівень серйозності окремих аналізаторів і загальний стиль перевірок для C#-файлів.**

**Перевірка роботи аналізатора:**

**Після успішного налаштування, Roslynator автоматично аналізує код під час компіляції або збереження файлів. Повідомлення про проблеми відображаються у Visual Studio (вікно “Error List”) або в терміналі при компіляції.**




## 6. Налаштування структурного логування з використанням бібліотеки SeriLog та інструменту Seq (або інших).

### Виконані кроки

### 1. Інсталяція бібліотек

### До проєкту були додані такі пакети:

- **Serilog.AspNetCore**
- **Serilog.Sinks.Console**

- **Serilog.Sinks.File**
- **Serilog.Sinks.Seq**

  Це дозволило зберігати лог-файли у файл, виводити їх у консоль, а також передавати у Seq для подальшого аналізу.

**2. Налаштування логування у програмі**

У файл Program.cs було додано конфігурацію Serilog з вказаними "sink"-ами (точками призначення):

```
Log.Logger = new LoggerConfiguration()
    .MinimumLevel.Debug()
    .Enrich.FromLogContext()
    .WriteTo.Console()
    .WriteTo.File("Logs/log-.txt", rollingInterval: RollingInterval.Day)
    .WriteTo.Seq("http://localhost:5341")
    .CreateLogger();
```

**3. Встановлення та запуск Seq**

Для локального аналізу логів було встановлено Seq. Його веб-інтерфейс доступний за адресою http://localhost:5341. У разі використання Docker, було використано відповідний контейнер.

**4. Додавання логів у код**

В окремих контролерах було реалізовано приклади логування подій:

Наприклад :

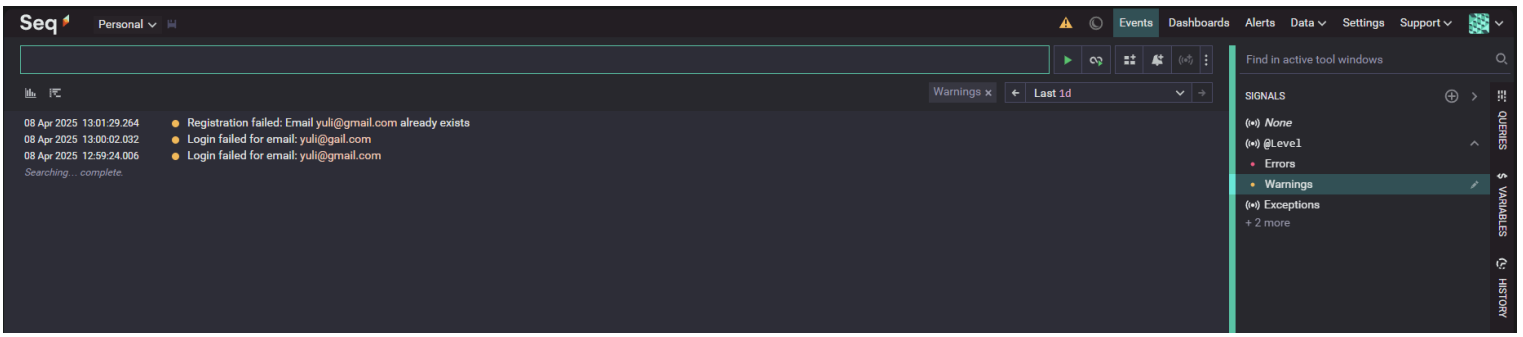
```
Log.Error(ex, "An error occurred during user registration for email: {Email}", dto.Email);

Log.Warning("Registration failed: Email {Email} already exists", dto.Email);
```

Такі структуровані повідомлення легко фільтрувати та аналізувати в Seq.

### 5. Перевірка роботи

Після запуску проєкту, лог-повідомлення відображались у:

- консолі розробника;
- текстових лог-файлах;
- веб-інтерфейсі Seq, де їх можна фільтрувати за рівнем, текстом чи параметрами (наприклад, UserId).




**Висновок :** Ми обрали **Layered architecture** для організації проекту, оскільки цей підхід забезпечує чітке розділення логіки на шари, що покращує підтримку та тестування.Ми побудували базові архітектурні структури проекту, застосовуючи **ASP.NET Core MVC.** Створили базу даних і підключились до неї через Entity Framework. Налаштували аналізатор коду StyleCop, що дозволяє дотримуватися кодових стандартів і покращує якість коду.

Для забезпечення зручного та структурованого логування було використано бібліотеку SeriLog у поєднанні з інструментом Seq.