МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМ. ТАРАСА ШЕВЧЕНКА

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА МЕРЕЖЕВИХ ТА ІНТЕРНЕТ ТЕХНОЛОГІЙ

Лабораторне заняття №3

Тема: КОНФІГУРАЦІЯ ПРОЄКТУ ЗАСТОСУНКУ ASP.NET CORE

Виконав студент групи: МІТ-41

Кухарчук Богдан Петрович

Київ - 2025

Хід роботи

1. Налаштування файлів конфігурації для різних середовищ

Було створено ієрархію конфігураційних файлів для розділення налаштувань між середовищами. У корінь проєкту додано файли sharedsettings.json (спільні налаштування) та appsettings.Production.json. Також відредаговано appsettings.Development.json.

Згідно із завданням, рядок підключення до бази даних (ConnectionStrings) та параметр ApplicationName були рознесені по різних файлах залежно від середовища.

Рис.3.1 Структура файлу appsettings.Development..json

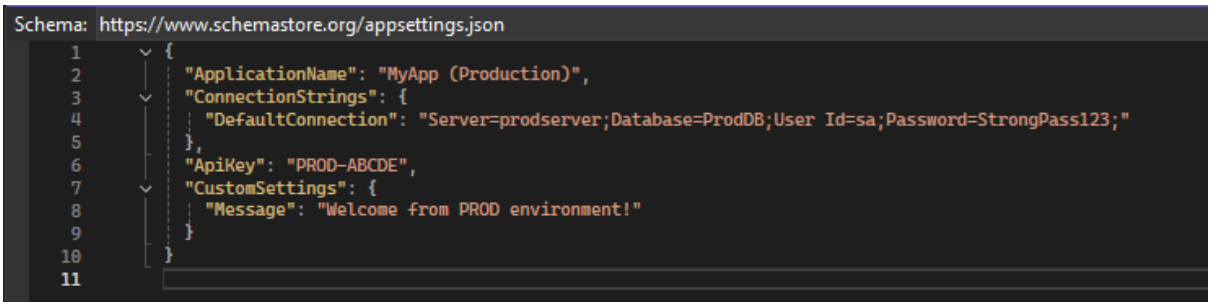


Рис.3.2 Структура файлу appsettings.Production.json

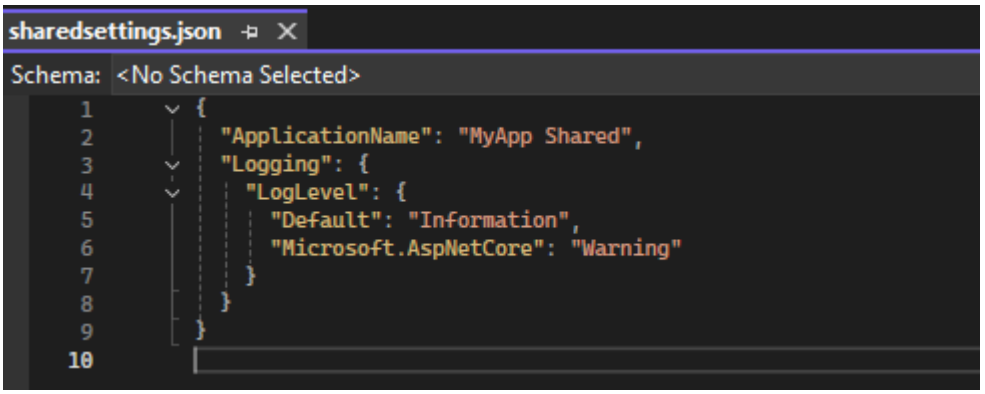


Рис.3.3 Структура файлу sharedsettings.json

2. Реалізація безпечного зберігання секретів (User Secrets)

Для зберігання конфіденційних даних, таких як ApiKey, у середовищі розробки було використано механізм User Secrets (Менеджер секретів), щоб уникнути їх потрапляння у репозиторій.

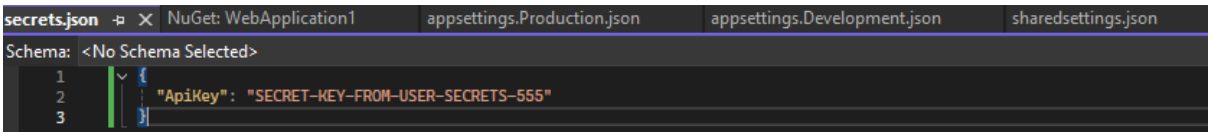


Рис. 3.4 Структура файлу secret.json

3. Реалізація строго типізованої конфігурації (Options Pattern)

Було створено клас MyConfiguration у проєкті WebApplicationData, який точно відображає структуру JSON-файлів налаштувань. Це дозволяє уникнути помилок, пов'язаних з ручним введенням ключів рядкового типу

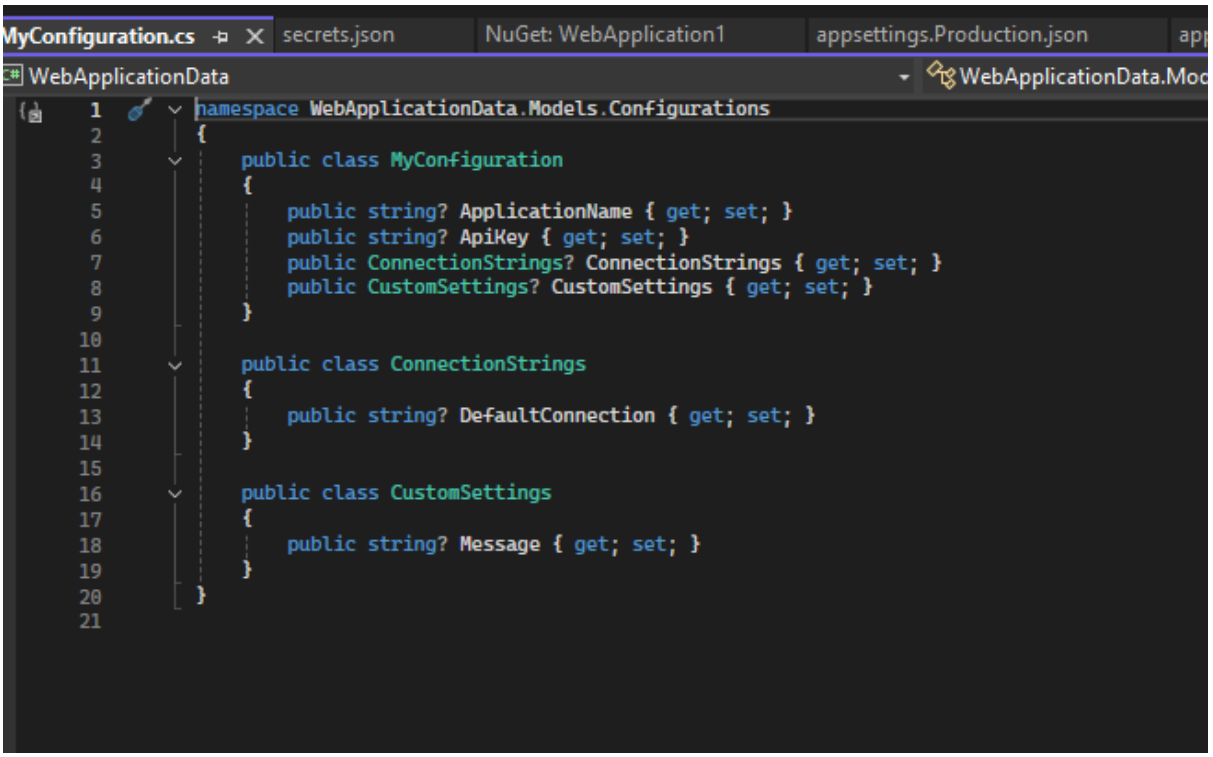


Рис.3.5 Структура файлу MyConfiguration.cs

У класі Program.cs було налаштовано завантаження конфігурації з пріоритетом джерел та зареєстровано сервіс конфігурації як Singleton у контейнері DI.

```
// --- 1. НАЛАШТУВАННЯ КОНФІГУРАЦІЇ (ЗАВДАННЯ 1) ---
builder.Configuration.Sources.Clear();
builder.Configuration
    .AddJsonFile("sharedsettings.json", optional: true, reloadOnChange: true)
    .AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)
    .AddJsonFile($"appsettings.{builder.Environment.EnvironmentName}.json", optional: true, reloadOnChange: true)
    .AddEnvironmentVariables();
```

Рис.3.6 Налаштування конфігурацій

```
// Завантаження секретів лише в середовищі розробки (Завдання 4)
if (builder.Environment.IsDevelopment())
{
    builder.Configuration.AddUserSecrets<Program>();
}
```

Рис.3.7 Завантаження секретів

4. Використання конфігурації у контролері та відображення

Сервіс конфігурації було інжектовано через конструктор у HomeController. Отримані значення (ApplicationName, ApiKey) передаються на представлення.

У файлі _Layout.cshtml було налаштовано відображення цих параметрів у футері сайту.

Результат роботи: На сторінці веб-застосунку у нижній частині відображаються дані, завантажені з конфігураційних файлів та User Secrets.

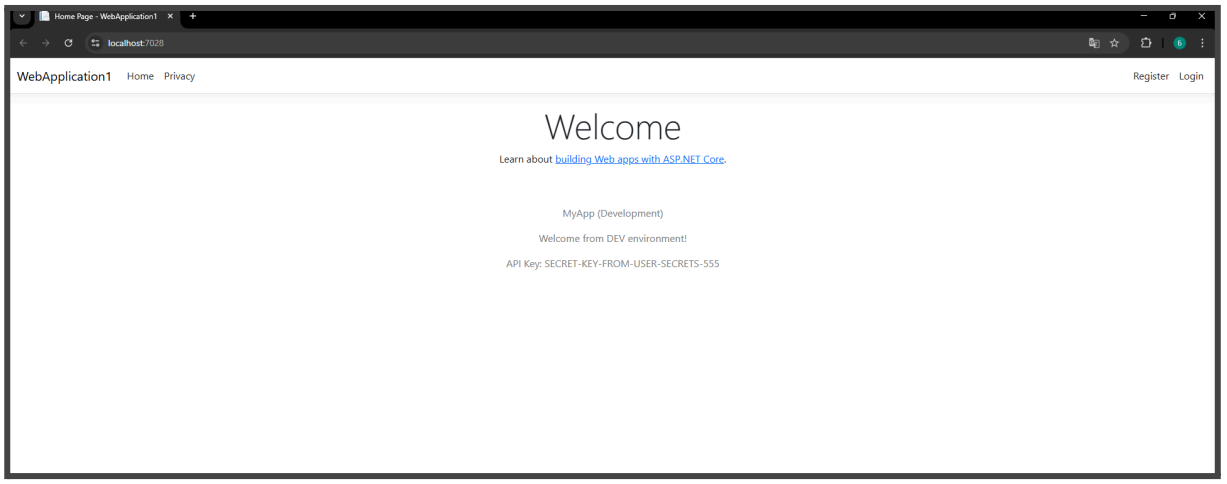


Рис.3.8 Демонстрація футера на сайті

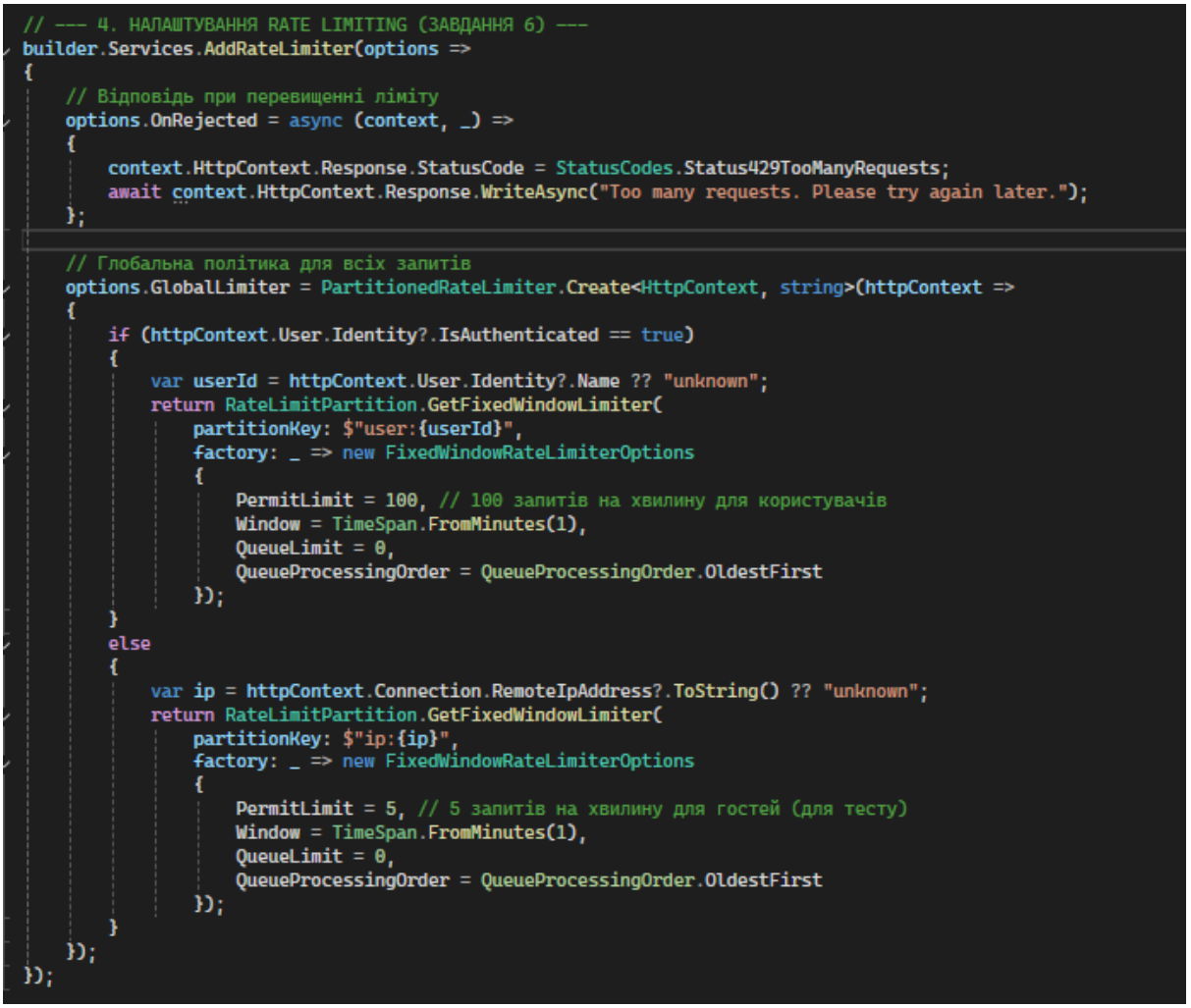
5. Реалізація Rate Limiting Middleware

Було додано та налаштовано Partitioned Rate Limiter для обмеження кількості запитів. Згідно із завданням, реалізовано різні політики для автентифікованих користувачів (100 запитів/хв) та гостей (20 запитів/хв, або 3 запити/10сек для тестування).

```
// --- 4. НАЛАШТУВАННЯ RATE LIMITING (ЗАВДАННЯ 6) ---

builder.Services.AddRateLimiter(options =>
{
    // Додано OnRejected, щоб повертати статус 429
    options.OnRejected = (context, _) =>
    {
        context.HttpContext.Response.StatusCode = StatusCodes.Status429TooManyRequests;
        return new ValueTask();
    };
```

Рис.3.9 Реалізація Rate Limitting



```
// --- 4. НАЛАШТУВАННЯ RATE LIMITING (ЗАВДАННЯ 6) ---
builder.Services.AddRateLimiter(options =>
{
    // Відповідь при перевищенні ліміту
    options.OnRejected = async (context, _) =>
    {
        context.HttpContext.Response.StatusCode = StatusCodes.Status429TooManyRequests;
        await context.HttpContext.Response.WriteAsync("Too many requests. Please try again later.");
    };

    // Глобальна політика для всіх запитів
    options.GlobalLimiter = PartitionedRateLimiter.Create<HttpContext, string>(httpContext =>
    {
        if (httpContext.User.Identity?.IsAuthenticated == true)
        {
            var userId = httpContext.User.Identity?.Name ?? "unknown";
            return RateLimitPartition.GetFixedWindowLimiter(
                partitionKey: $"user:{userId}",
                factory: _ => new FixedWindowRateLimiterOptions
                {
                    PermitLimit = 100, // 100 запитів на хвилину для користувачів
                    Window = TimeSpan.FromMinutes(1),
                    QueueLimit = 0,
                    QueueProcessingOrder = QueueProcessingOrder.OldestFirst
                });
        }
        else
        {
            var ip = httpContext.Connection.RemoteIpAddress?.ToString() ?? "unknown";
            return RateLimitPartition.GetFixedWindowLimiter(
                partitionKey: $"ip:{ip}",
                factory: _ => new FixedWindowRateLimiterOptions
                {
                    PermitLimit = 5, // 5 запитів на хвилину для гостей (для тесту)
                    Window = TimeSpan.FromMinutes(1),
                    QueueLimit = 0,
                    QueueProcessingOrder = QueueProcessingOrder.OldestFirst
                });
        }
    });
});
```

Рис.3.10 Реалізація обмеження запиту

Рис.3.11 Перевірка правильності обмеження запитів

Висновок: Під час лабораторної роботи було налаштовано конфігурацію застосунку ASP.NET Core для роботи з кількома середовищами. Реалізовано безпечне зберігання секретів через User Secrets та доступ до налаштувань через строго типізовані класи (Options Pattern).

Додатково було інтегровано Middleware для обмеження частоти запитів (Rate Limiting), налаштоване на різні політики для авторизованих та неавторизованих користувачів. Проєкт структуровано та збережено на GitHub.