Київський національний університет ім. Тараса Шевченка
Кафедра мережевих та інтернет технологій

Лабораторна робота № 3

Дисципліна: Сучасні Інтернет Технології
Тема: Конфігурація проєкту застосунку ASP.NET CORE
Підготувала студентка МІТ-41
Пась Олександра

Хід виконання

1. Забезпечте проєкт файлами sharedsettings.json, appsettings.Development.json та appsettings.Production.json. Налаштуйте різні значення параметра ApplicationName та принаймні одного специфічного для свого проєкту параметра для кожного середовища.

   Для виконання цього завдання було створено:
   - sharedsettings.json — спільні налаштування для всіх середовищ;
   - appsettings.Development.json — налаштування для режиму розробки;
   - appsettings.Production.json — налаштування для продакшн-режиму.

   Структура файлів однакова.

рис. 3.1 - вміст файлу appsettings.Development.json

2. Забезпечте належне розташування параметра ConnectionString та коректну обробку різних значень для середовищ Development та Production.

   У файлі appsettings.Development.json я не записувала реальний рядок підключення до бази, а зберегла його у Secret Manager (secrets.json) — щоб приховати пароль.

   Для Production можна вказати рядок прямо у launchSettings.json.

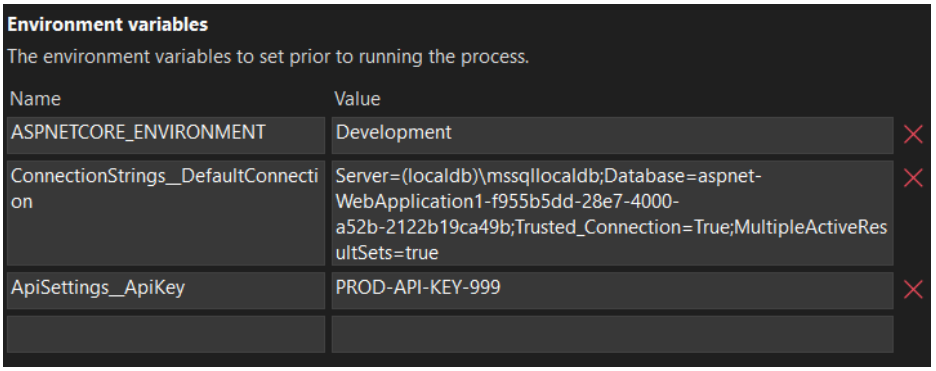


рис. 3.2 - змінні середовища

У Program.cs підключили конфігурацію

```
builder.Configuration.Sources.Clear();
builder.Configuration.AddJsonFile("sharedsettings.json", optional: false, reloadOnChange: true)
    .AddJsonFile("appsettings.json", optional: false, reloadOnChange: true);

if (builder.Environment.IsDevelopment())
{
    builder.Configuration.AddJsonFile("appsettings.Development.json", optional: true);
    builder.Configuration.AddUserSecrets<Program>(optional: true);
}
if (builder.Environment.IsProduction())
{
    builder.Configuration.AddJsonFile("appsettings.Production.json", optional: true);
    builder.Configuration.AddEnvironmentVariables();
}
```

рис. 3.3 - підключення конфігурації

3. Створіть строго типізоване налаштування всієї ієрархії параметрів конфігурації. Додайте у контейнер DI застосунку сервіс конфігурації з життєвим циклом Singleton. Інжектуйте сервіс конфігурації через конструктор у контролері та використайте його для виведення у Footer інтерфейсу параметрів із завдання 1.

Для виконання цього проекту було створено папку Configurations, а в ній файл AppConfiguration.cs у якому створено клас, який відображає структуру конфігураційних параметрів нашого застосунку.

Він містить вкладений клас ApiSettings із параметром ApiKey.

Цей клас використовується для зручного доступу до значень із файлів appsettings.json через Dependency Injection.

```
namespace WebApplication1.Configurations
{
    4 references
    public class AppConfiguration
    {
        0 references
        public string ApplicationName { get; set; } = string.Empty;
        0 references
        public string Theme { get; set; } = string.Empty;
        1 reference
        public ApiSettings ApiSettings { get; set; } = new ApiSettings();
    }

    2 references
    public class ApiSettings
    {
        1 reference
        public string ApiKey { get; set; } = string.Empty;
    }
}
```

рис. 3.4 - вміст файлу

У файлі Program.cs додали реєстрацію цього класу. Це дозволяє будь-якому контролеру або сервісу отримати доступ до налаштувань через конструктор.

```
builder.Services.Configure<AppConfiguration>(builder.Configuration);
builder.Services.AddSingleton(resolver =>
    resolver.GetRequiredService<Microsoft.Extensions.Options.IOptions<AppConfiguration>>().Value);
```

рис. 3.5 - реєстрація класу

У HomeController.cs додали залежність через конструктор. Таким чином, контролер отримує дані напряму з файлів appsettings.*.json.

```
private readonly ILogger<HomeController> _logger;
private readonly AppConfiguration _appSettings; // Додаємо поле для збереження конфігурації

//Додаємо AppSettings у конструктор через Dependency Injection
0 references
public HomeController(ILogger<HomeController> logger, AppConfiguration appSettings)
{
    _logger = logger;
    _appSettings = appSettings;
}

//Використовуємо AppSettings у дії Index
0 references
public IActionResult Index()
{
    // Передаємо дані у View через ViewBag
    ViewBag.AppName = _appSettings.ApplicationName;
    ViewBag.Theme = _appSettings.Theme;
    ViewBag.ApiKey = _appSettings.ApiSettings.ApiKey;

    return View();
}
```

рис. 3.6 - вміст HomeController

У файлі Views/Shared/_Layout.cshtml (у нижній частині сторінки) додали блок із виведенням параметрів. Тепер під час запуску сайту внизу сторінки з’являється інформація про застосунок і конфігураційні параметри, які беруться автоматично з поточного середовища.

```
<footer class="border-top footer text-muted">
    <div class="container">
        <div class="row">
            <div class="col-md-6">
                &copy; 2025 - @(ViewBag.AppName ?? "WebApplication1") -
                <a asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
            </div>
            <div class="col-md-6 text-end">
                <small>
                    <strong>Configuration:</strong><br />
                    Theme: @(ViewBag.Theme ?? "Default") |
                    API Key: @(ViewBag.ApiKey ?? "N/A")
                </small>
            </div>
        </div>
    </div>
</footer>
```

рис. 3.7 - layout.cshtml

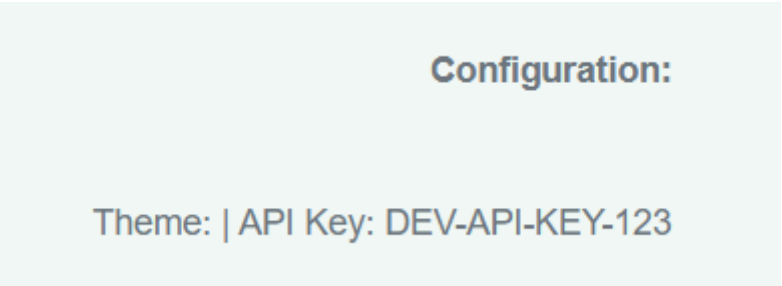
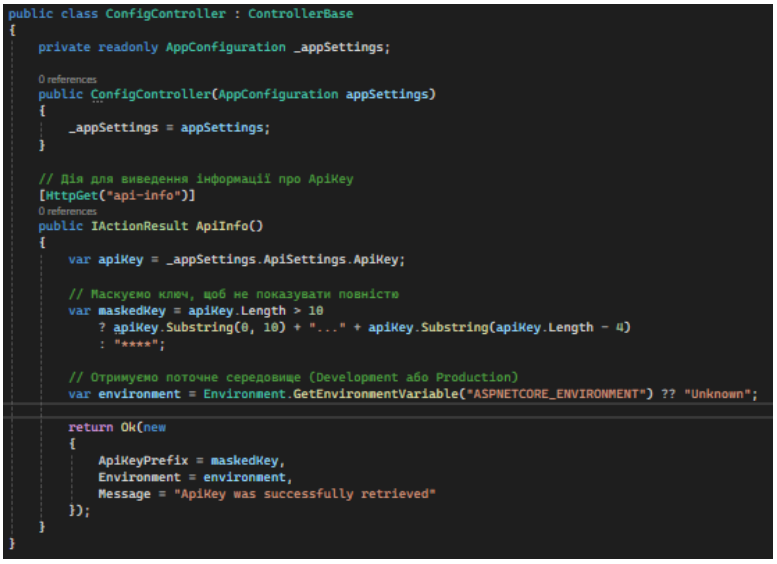


рис. 3.8 - відображення

4. Забезпечте конфігурацію параметром ApiKey. Забезпечте використання різних значень для середовища розробки та промислового середовища.

   Було створено Action для виведення інформації про ключі. Перевірити можна запустивши проект та ввести адресу https://localhost:7131/api/config/api-info



```
public class ConfigController : ControllerBase
{
    private readonly AppConfiguration _appSettings;

    0 references
    public ConfigController(AppConfiguration appSettings)
    {
        _appSettings = appSettings;
    }

    // Дія для виведення інформації про ApiKey
    [HttpGet("api-info")]
    0 references
    public IActionResult ApiInfo()
    {
        var apiKey = _appSettings.ApiSettings.ApiKey;

        // Маскуємо ключ, щоб не показувати повністю
        var maskedKey = apiKey.Length > 10
            ? apiKey.Substring(0, 10) + "..." + apiKey.Substring(apiKey.Length - 4)
            : "****";

        // Отримуємо поточне середовище (Development або Production)
        var environment = Environment.GetEnvironmentVariable("ASPNETCORE_ENVIRONMENT") ?? "Unknown";

        return Ok(new
        {
            ApiKeyPrefix = maskedKey,
            Environment = environment,
            Message = "ApiKey was successfully retrieved"
        });
    }
}
```

рис.3.9 - метод

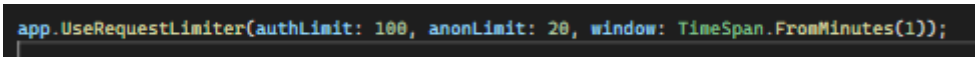

```
{
  "apiKeyPrefix": "DEV-API-KE...-123",
  "environment": "Development",
  "message": "ApiKey was successfully retrieved"
}
```


рис. 3.10 - результат

5. Додати Partitioned Rate Limiting middleware, що надає різні привілеї (кількість запитів за хвилину) для автентифікованих та неавтентифікованих користувачів. В разі обмеження повертати статус 429 – Too Many Requests.

   Було створено окрему папку Middlewares, де додали клас RequestLimiterMiddlewareExtensions.cs та RequestLimiter.cs. middleware обмежує кількість запитів:

   - для авторизованих користувачів — 100 запитів за хвилину;
   - для неавторизованих — 20 запитів за хвилину.



```
app.UseRequestLimiter(authLimit: 100, anonLimit: 20, window: TimeSpan.FromMinutes(1));
```

рис. 3.11 - впровадження Middlewares у додаток

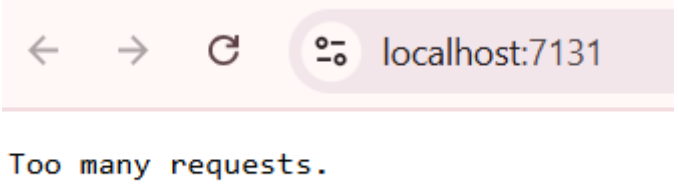


рис. 3.12 - перевірка роботи

Висновок: Під час виконання роботи я навчилась налаштовувати конфігураційні файли ASP.NET Core для різних середовищ (Development і Production), створювати строго типізовані класи для зчитування параметрів, використовувати User Secrets для безпечного зберігання даних, а також реалізувала перевірку ApiKey через власний API-контролер.