

МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМ. ТАРАСА ШЕВЧЕНКА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра мережевих та інтернет технологій

СУЧАСНІ ІНТЕРНЕТ ТЕХНОЛОГІЇ

КОНФІГУРАЦІЯ ПРОЄКТУ ЗАСТОСУНКУ ASP.NET CORE

Лабораторне заняття № 3

Заяць Діани

Хід виконання роботи:

1. Забезпечення проєкту файлами `sharedsettings.json`, `appsettings.Development.json` та `appsettings.Production.json`.

У проєкті створено та підключено конфігураційні файли `sharedsettings.json`, `appsettings.Development.json` і `appsettings.Production.json`.

```
1
2  {
3    "ApplicationName": "WebAppCore (Shared)",
4    "Logging": {
5      "LogLevel": {
6        "Default": "Information"
7      }
8    }
9  }
10
11
```

Рис 3.1 - Конфігураційний файл `shared.settings` в форматі json

```
11 "ProjectSettings": {
12   "Theme": "Light",
13   "ShowDebugInfo": true
14 },
15 "ConnectionStrings": {
16   "DefaultConnection": "Server=(localdb)\\MSSQLLocalDB;Database=WebAppCore;Trusted_Connection=True;MultipleActiveResultSets=true;TrustServerCertificate=True",
17   "ApplicationDbContextConnection": "Server=(localdb)\\mssqllocaldb;Database=WebAppCore;Trusted_Connection=True;MultipleActiveResultSets=true"
18 },
19 "Logging": {
20   "LogLevel": {
21     "Default": "Information",
22     "Microsoft.AspNetCore": "Warning"
23   }
24 }
```

Рис 3.2 - Конфігураційний файл `appsettings.Development` в форматі json

```

1 {
2   "ApplicationName": "WebAppCore (Production)",
3   "ProjectSettings": {
4     "Theme": "Dark",
5     "ShowDebugInfo": false
6   },
7
8   "ConnectionStrings": {
9     "DefaultConnection": "Server=prod-server;Database=WebAppCoreProd;User Id=admin;Password=StrongPassword123;MultipleActiveResultSets=true",
10    "ApplicationDbContextConnection": "Server=prod-server;Database=WebAppCoreIdentity;User Id=admin;Password=StrongPassword123;MultipleActiveResultSets=true"
11  },
12
13  "Logging": {
14    "LogLevel": {
15      "Default": "Error",
16      "Microsoft.AspNetCore": "Warning"
17    }
18  }
19 }
20

```

Рис 3.3 - Конфігураційний файл `appsettings.Production` в форматі `json`

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

У застосунку налаштовано коректне розташування параметра `ConnectionString` у конфігурації та вибір його значення залежно від середовища `Development` або `Production`. Для цього використовується перевірка поточного оточення й отримання рядка підключення через `builder.Configuration.GetConnectionString(...)`

Також додано підтримку секретів і різних оточень: рядок підключення та інші чутливі параметри зчитуються з конфігурації та секретного сховища окремо

```

var builder = WebApplication.CreateBuilder(args);
{
    if (builder.Environment.IsDevelopment())
    {
        builder.Configuration.AddUserSecrets<Program>();
    }

    bool isLinux = RuntimeInformation.IsOSPlatform(OSPlatform.Linux);
    var mitConfig = builder.Configuration.GetMitConfigurations();
    if (mitConfig == null)
    {
        throw new InvalidOperationException("MitConfiguration is not set in configuration.");
    }
    builder.Services.AddSingleton(mitConfig);
    var mapSettings = mitConfig?.MapSettings;

    // Add services to the container
    var connectionString = builder.Configuration.GetConnectionString("DefaultConnection");
    ?? throw new InvalidOperationException("Connection string 'DefaultConnection' not found.");

    if (RuntimeInformation.IsOSPlatform(OSPlatform.Linux))
    {
        connectionString = builder.Configuration.GetConnectionString("LinuxDockerConnection");
    }

    builder.Services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(connectionString));

    builder.Services.AddScoped<IMitRepository, SqlServerRepository>();
    builder.Services.AddDatabaseDeveloperPageExceptionFilter();

    builder.Services.AddIdentity<ApplicationUser, IdentityRole>(options =>
    {
        options.SignIn.RequireConfirmedAccount = false;
    })
    .AddEntityFrameworkStores<ApplicationDbContext>()
    .AddDefaultTokenProviders();

    builder.Services.AddControllersWithViews();
    builder.Services.AddRazorPages();
}

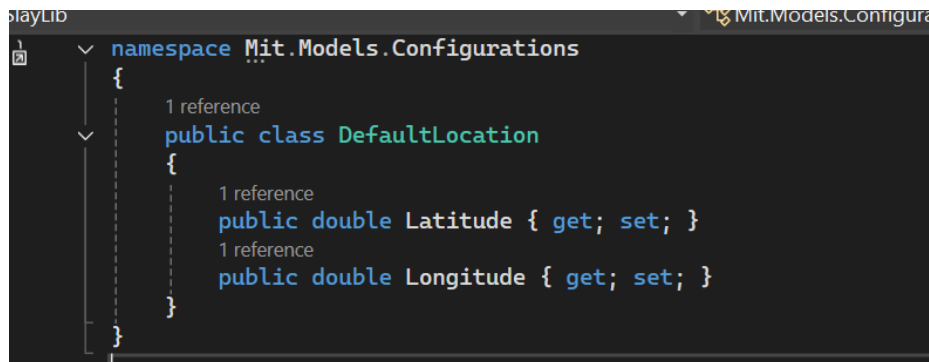
```

Рисунок 3.2 – Включення обробки секретів для різних оточень

3. Створіть строго типізоване налаштування всієї ієрархії

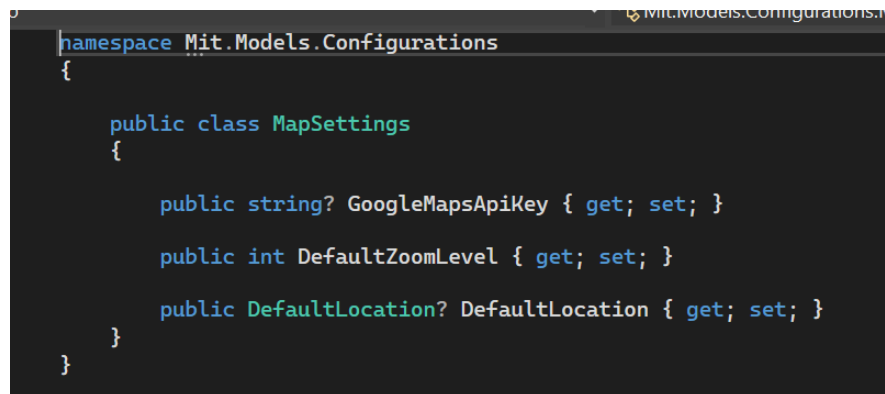
параметрів конфігурації. Додайте у контейнер DI застосунку сервіс конфігурації з життєвим циклом Singleton. Інжектуйте сервіс конфігурації через конструктор у контролері та використайте його для виведення у Footer інтерфейсу параметрів із завдання 1

Було сформовано структуру строго типізованої конфігурації застосунку. Для цього у проєкті було створено окремі класи, що відповідають структурі конфігураційного файлу. Класи MitConfiguration, MapSettings, DefaultLocation та ProjectSettings описують відповідні розділи конфігурації, що дозволяє застосунку працювати з параметрами у типізованому вигляді та уникати помилок, пов'язаних із використанням рядкових ключів



```
namespace Mit.Models.Configurations
{
    1 reference
    public class DefaultLocation
    {
        1 reference
        public double Latitude { get; set; }
        1 reference
        public double Longitude { get; set; }
    }
}
```

Рис. 3.3 – Клас DefaultLocation



```
namespace Mit.Models.Configurations
{
    public class MapSettings
    {
        public string? GoogleMapsApiKey { get; set; }
        public int DefaultZoomLevel { get; set; }
        public DefaultLocation? DefaultLocation { get; set; }
    }
}
```

Рис. 3.4 – Клас MapSettings

```

namespace Mit.Models.Configurations
{
    public class MitConfiguration
    {
        public string? ApplicationName { get; set; }
        public string? MyAppConnString { get; set; }
        public string? DefaultConnection { get; set; }
        public MapSettings? MapSettings { get; set; }
        public string? ApiKey { get; set; }
        public ProjectSettings? ProjectSettings { get; set; }
    }
}

```

Рис. 3.5 – Клас MitConfiguration

```

namespace Mit.Models.Configurations
{
    public class ProjectSettings
    {
        public string? Theme { get; set; }
        public bool ShowDebugInfo { get; set; }
    }
}

```

Рис. 3.6 – Клас ProjectSettings

Після визначення моделей конфігурації виконано інтеграцію цих параметрів у процес ініціалізації застосунку. У файлі Program.cs конфігурацію було зчитано за допомогою механізму `builder.Configuration.Get<MitConfiguration>()`, а отриманий об'єкт зареєстровано у контейнері залежностей як `Singleton`. Це забезпечує доступ до конфігураційних даних у будь-якій частині застосунку без повторного зчитування файлів.

```

var mitConfig = builder.Configuration.Get<MitConfiguration>();

if (mitConfig == null)
    throw new InvalidOperationException("MitConfiguration is not set in configuration.");

builder.Services.AddSingleton(mitConfig);

```

Рис. 3.7 – Зчитування і реєстрація конфігурації

Реалізована конфігурація була використана у візуальній частині застосунку. Через механізм Razor-інжекції (`@inject MitConfiguration Cfg`) об'єкт конфігурації було інтегровано у файл `_Layout.cshtml`. Це дозволило вивести у футері сторінки значення параметрів, зокрема назву застосунку, координати за замовчуванням, тему та інші допоміжні дані,

продемонструвавши практичне застосування конфігурації у UI.

```
@using Mit.Models.Configurations
@Inject MitConfiguration Cfg
```

Рис. 3.8 – Інжекція конфігурації

```
@if (Cfg.MapSettings?.DefaultLocation is not null)
{
    <span> · @{Cfg.MapSettings.DefaultLocation.Latitude:0.####}, {Cfg.MapSettings.DefaultLocation.Longitude:0.####}</span>
}

@if (!string.IsNullOrWhiteSpace(Cfg.ProjectSettings?.Theme))
{
    <span> · Theme: @Cfg.ProjectSettings.Theme</span>
}

@if (Cfg.ProjectSettings?.ShowDebugInfo == true)
{
    <span> · Debug: on</span>
}

@if (!string.IsNullOrWhiteSpace(Cfg.ApiKey))
{
    <span> · ApiKey: @Mask(Cfg.ApiKey)</span>
}
```

Рис. 3.8 – Виведення конфігурації у футері

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

Було реалізовано механізм зберігання параметра `ApiKey`, значення якого має відрізнятися залежно від середовища виконання. У файлі `appsettings.Development.json` було визначено тестове значення ключа, що застосовується лише в режимі розробки. Це демонструє можливість гнучкого налаштування параметрів для різних оточень без зміни коду застосунку.

```
"ApplicationName": "WebAppCore (Development)",
"ApiKey": "DEV-123456-PLACEHOLDER",
"MapSettings": {
    "DefaultZoomLevel": 9,
    "DefaultLocation": {
        "Latitude": 50.45,
        "Longitude": 30.52
    }
},
```

Рис. 3.9 – Вміст `appsettings.Development.json`

З міркувань безпеки було підключено механізм *User Secrets*, що дозволяє зберігати справжні значення ключів поза межами проєкту. Такий підхід виключає можливість випадкового потрапляння конфіденційних даних у публічний доступ та відповідає сучасним вимогам щодо безпечної розробки. У файлі

Program.cs передбачено автоматичне підключення User Secrets у середовищі Development.

```
> dotnet user-secrets set "ApiKey" "DEV-REAL-SECRET"
```

Рис. 3.10 - Консольна команда User Secrets

5. Ознайомтеся з теоретичними основами middleware: що таке конвеєр обробки запитів, як працює делегування next, які є типи middleware.

Наведіть приклади системного та користувацького middleware.

Middleware — це проміжний компонент веб-застосунку, який обробляє HTTP-запит перед тим, як він потрапляє до кінцевого обробника або контролера.

Конвеєр обробки запитів (request pipeline) — це послідовність middleware, через які проходить кожен запит у тому порядку, в якому вони

zareestrovani v zastosunku. Kozhne middleware mozhe vykonuvati vlasnu logiku, zminiovati zapyt, pereviritи umovi abo formuvati vidpovid bez pereдачи keruvannya dali.

Механізм делегування next використовується для передачі керування наступному middleware у конвеєрі, що дозволяє забезпечити послідовну обробку запита. Якщо middleware не викликає функцію next, виконання конвеєра зупиняється, а відповідь формується у цьому ж middleware. У системах існують два основні типи middleware: системні (вбудовані) та користувацькі (створені розробником):

- Системне middleware входить до складу фреймворку та забезпечує стандартні функції, такі як аутентифікація, роутинг, обробка помилок та обслуговування статичних файлів.
- Користувацьке middleware створюється розробником для реалізації специфічних потреб застосунку, наприклад логування, перевірки токенів або обмеження частоти запитів.

Middleware zabezpeчує gnučku i rozshiryuvanu arkhitekturu veb-zastosunku, dozvolayuchi legko dodavati novu funktsionalnist bez

зміни основного коду.

- 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. (*max - 20 балів*)
- Зафіксувати зміни у проєкті на GitHub. (*max - 10 балів*)

Завдання 6. 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.

```
builder.Services.AddRateLimiter(options =>
{
    options.GlobalLimiter = PartitionedRateLimiter.Create<HttpContext, string>(context =>
    {
        var key = context.User.Identity?.IsAuthenticated == true
            ? context.User.Identity.Name ?? "authenticated"
            : "anonymous";

        if (context.User.Identity?.IsAuthenticated == true)
        {
            return RateLimiterPartition.GetFixedWindowLimiter(key, _ => new FixedWindowRateLimiterOptions
            {
                PermitLimit = 120,
                Window = TimeSpan.FromMinutes(1),
                QueueProcessingOrder = QueueProcessingOrder.OldestFirst,
                QueueLimit = 0
            });
        }
        else
        {
            return RateLimiterPartition.GetFixedWindowLimiter(key, _ => new FixedWindowRateLimiterOptions
            {
                PermitLimit = 60,
                Window = TimeSpan.FromMinutes(1),
                QueueProcessingOrder = QueueProcessingOrder.OldestFirst,
                QueueLimit = 0
            });
        }
    });
});

options.OnRejected = async (context, cancellationToken) =>
{
    context.HttpContext.Response.StatusCode = 429;
    context.HttpContext.Response.ContentType = "text/html; charset=utf-8";

    var executor = context.HttpContext.RequestServices.GetRequiredService<Microsoft.AspNetCore.Mvc.Infrastructure.IActionResultExecutor>();
    var viewResult = new Microsoft.AspNetCore.Mvc.ViewResult
    {
        ViewName = "~/Views/Shared/TooManyRequests.cshtml"
    };

    await executor.ExecuteAsync(
        new Microsoft.AspNetCore.Mvc.ActionContext(
            context.HttpContext,
            context.HttpContext.GetRouteData() ?? new Microsoft.AspNetCore.Routing.RouteData(),
            new Microsoft.AspNetCore.Mvc.Abstractions.ActionDescriptor()
        ),
        viewResult
    );
};
});
```

Рисунок 3.5 – Зареєстрований сервіс RateLimiter

```

@{
    Layout = null;
}

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>429 Too Many Requests</title>
    <style>
        body {
            background-color: #fff;
            color: #000;
            font-family: Arial, sans-serif;
            text-align: center;
            padding-top: 10%;
        }
        h1 { font-size: 50px; margin-bottom: 0; }
        p { font-size: 20px; }
        .box {
            display: inline-block;
            padding: 40px;
            border: 1px solid #ccc;
            box-shadow: 2px 2px 12px rgba(0,0,0,0.1);
        }
    </style>
</head>
<body>
    <div class="box">
        <h1>429</h1>
        <p>Too Many Requests</p>
        <p>Please try again later.</p>
        <a href="/">Go Home</a>
    </div>
</body>
</html>

```

Рисунок 3.6 – TooManyRequests Razor View

```
// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseMigrationsEndPoint();
}
else
{
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRateLimiter();

app.UseRouting();

app.UseAuthentication();
app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");
app.MapRazorPages();

app.Run();
```

Рисунок 3.7 – Включення RateLimiting Middleware до конвеєра для відхилення запитів

В разі перевищення вказаної частоти запитів на хвилину, відповідно до того, чи авторизований користувач, додаток повертатиме помилку 429 Too many requests

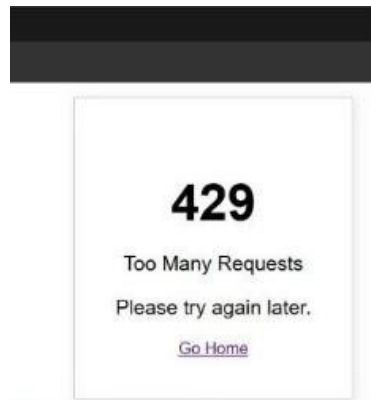


Рисунок 3.8 – Повернення помилки 429 Too Many Requests

Висновок:

У рамках лабораторної роботи були налаштовані конфігураційні файли для середовищ **Development** і **Production**, які централізовано містять параметри застосунку (ApplicationName, ConnectionString, ApiKey) і дають змогу змінювати їх без втручання в програмний код. Застосування строго типізованих налаштувань

та доступу до конфігурації через **Dependency Injection** підвищує безпечність і спрощує роботу з параметрами в різних модулях системи.

Крім того, впроваджено **Partitioned Rate Limiting middleware**, яке регулює інтенсивність запитів і розділяє ліміти для різних груп користувачів. Це демонструє ефективність використання конфігураційних файлів і middleware для покращення масштабованості та надійності веб-застосунку.