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

Черкун Єви Сергіївни

Хід виконання роботи:

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

sharedsettings.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

appsettings.Development.json

```
    "ProjectSettings": {
        "Theme": "Light",
        "ShowDebugInfo": true
    },

    "ConnectionStrings": {
        "DefaultConnection": "Server=(localdb)\\MSSQLLocalDB;Database=WebAppCore;Trusted_Connection=True;MultipleActiveResultSets=true;TrustServerCertificate=True",
        "ApplicationDbContextConnection": "Server=(localdb)\\mssqllocaldb;Database=WebAppCore;Trusted_Connection=True;MultipleActiveResultSets=true"
    },

    "Logging": {
        "LogLevel": {
            "Default": "Information",
            "Microsoft.AspNetCore": "Warning"
        }
    }
}
```

Рис 3.2 - Конфігураційний файл appsettings.Development в форматі json

appsettings.Production.json.

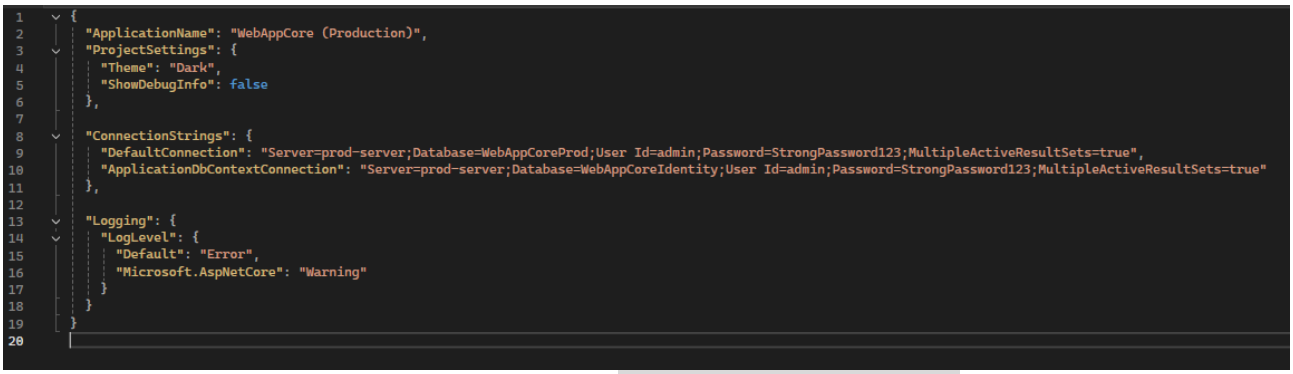
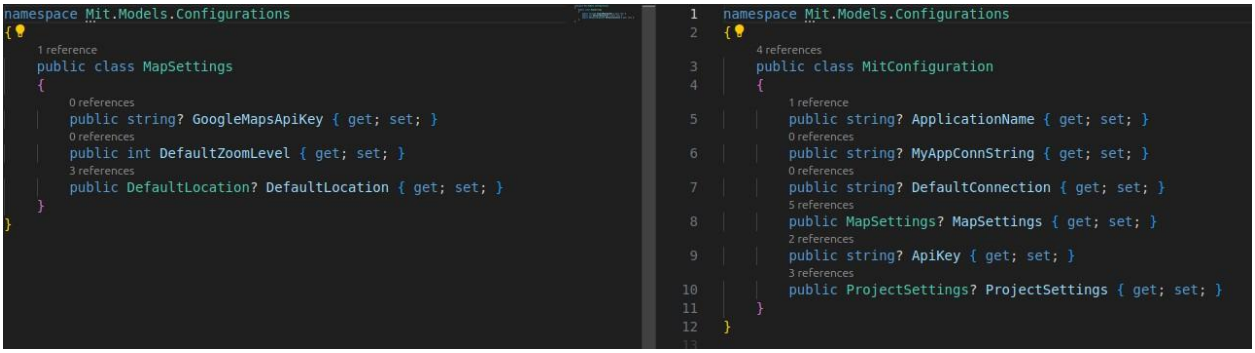


Рис 3.3 - Конфігураційний файл appsettings.Production в форматі json

2. Забезпечте належне розташування параметра ConnectionString та коректну обробку різних значень для середовищ Development та Production. (*max - 10 балів*)

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. (*max - 25 балів*)
4. Забезпечте конфігурацію параметром ApiKey. Забезпечте використання різних значень для середовища розробки та промислового середовища. (*max - 5 балів*)



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

```
namespace Mit.Models.Configurations
{
    4 references
    public class MitConfiguration
    {
        1 reference
        public string? ApplicationName { get; set; }
        0 references
        public string? MyAppConnString { get; set; }
        0 references
        public string? DefaultConnection { get; set; }
        5 references
        public MapSettings? MapSettings { get; set; }
        2 references
        public string? ApiKey { get; set; }
        3 references
        public ProjectSettings? ProjectSettings { get; set; }
    }
}
```

Рисунок 3.3 – Створення конфігурацій

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. (*max - 10 балів*)

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 зареєстровані в застосунку. Кожне middleware може виконувати власну логіку, змінювати запит, перевіряти умови або формувати відповідь без передачі керування далі.

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 забезпечує гнучку і розширювану архітектуру веб-застосунку, дозволяючи легко додавати нову функціональність без зміни основного коду.

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. (*max - 20 балів*) 7. 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 RateLimitPartition.GetFixedWindowLimiter(key, _ => new FixedWindowRateLimiterOptions
            {
                PermitLimit = 120,
                Window = TimeSpan.FromMinutes(1),
                QueueProcessingOrder = QueueProcessingOrder.OldestFirst,
                QueueLimit = 0
            });
        }
        else
        {
            return RateLimitPartition.GetFixedWindowLimiter(key, _ => new FixedWindowRateLimiterOptions
            {
                PermitLimit = 40,
                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<Microsoft.AspNetCore.Mvc.ViewResult>>();
        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

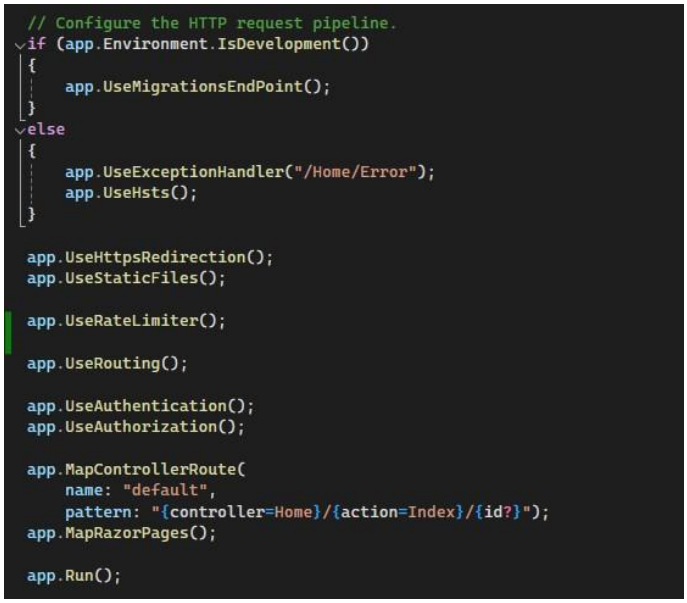


Рисунок 3.7 – Включення RateLimiting Middleware до конвеєра для відхилених запитів

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

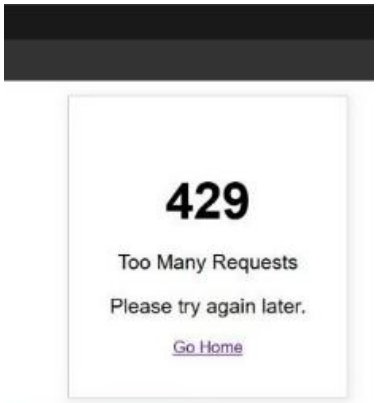


Рисунок 3.8 – Повернення помилки 429 Too Many Requests

Висновок:

У ході лабораторної роботи налаштовано конфігураційні файли для середовищ Development та Production, які централізовано зберігають параметри застосунку (ApplicationName, ConnectionString, ApiKey) і дозволяють змінювати їх без редагування коду. Використання строго типізованих налаштувань і сервісу конфігурації через Dependency Injection підвищує безпеку та зручність доступу до параметрів у різних частинах застосунку.

Також додано Partitioned Rate Limiting middleware, що ефективно контролює кількість запитів і розмежовує доступ для різних категорій користувачів. Це підтверджує доцільність використання конфігураційних файлів і middleware для підвищення масштабованості та надійності веб-застосунку.