**ЛАБОРАТОРНА РОБОТА № 1**

**Початок роботи з ASP.NET Core 5. Middleware**

***Мета***: ознайомитися зі структурою ASP.NET Code додатків, навчитися створювати найпростіші та використовувати наявні компоненти middleware у ASP.NET Core.

**Хід роботи:**

**Завдання 2:** Створити у Visual Studio **ASP.NET Core Empty**.

Назва рішення: **AspDotNetLab1**

Назва проекту: **AspDotNetLab1**

При створенні проекту виберіть версію Framework: .NET 5.0 (Current) Якщо у Вас не доступна версія .NET 5.0, то встановіть відповідний компонент (слайд 2 презентації до лекції №1)

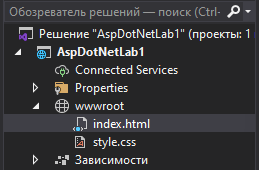


Рис. 2.1. Оглядач рішень з відкритим проектом

**Завдання 3:** Реалізація статичного сервера.

Створюємо папку з назвою “wwwroot”. Розміщуємо у папці wwwroot декілька статичних файлів з довільним контентом, наприклад файл index.html та таблицю стилів, яка у ньому підключена:

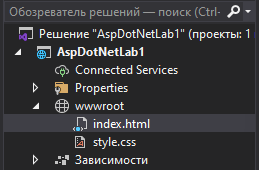


Рис. 3.1. Результат виконання.

Відкриваємо файл Startup.cs, знаходимо метод Configure. Після рядка **app.UseRouting();** додаємо виклик метода: **app.UseFileServer();:**

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseRouting();

app.UseFileServer();

app.UseMiddleware<LoggerMiddleware>();

app.UseMiddleware<SecretMiddleware>();

app.UseEndpoints(endpoints =>

{

endpoints.MapGet("/", async context =>

{

await context.Response.WriteAsync("Hello World!");

});

});

}

Запускаємо проект і дивимось, що виводиться при зверненні до кореня сайту “/” та до сторінки “/index.html”.

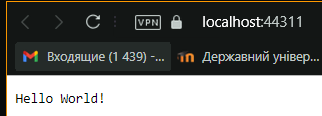


Рис. 3.2. Результат відображення при зверненні до кореня сайту “/”.

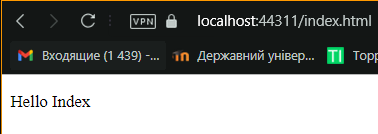


Рис. 3.3. Результат відображення при зверненні до сторінки “/index.html”.

Міняємо рядки місцями:

**app.UseFileServer();**

**app.UseRouting();**

Запускаємо проект і дивимось, що виводиться при зверненні до кореня сайту “/” та до сторінки “/index.html”.

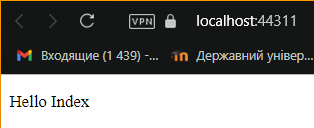


Рис. 3.4. Результат відображення при зверненні до кореня сайту “/”.

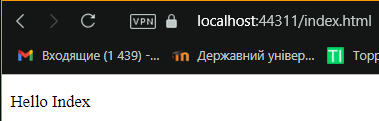


Рис. 3.5. Результат відображення при зверненні до сторінки “/index.html”.

***Пояснення відмінностей***

В конвейєрі обробки запитів дуже важливе значення має те, в якому поряду у програмному коді викликані компоненти middleware. Тому в першому випадку спочатку увімкнеться маршрутизація і ми отримуємо відповідний результат.

У другому випадку, так як компонент useFileServer() викланий раніше і включає в себе декілька інших компонентів, зокрема UseDefaultFiles(), то при зверненні до адреси «/» буде шукатись один із дефолтних файлів на статичному сервері, у нашому випадку буде знайдено файл index.html

Додамо ще один статичний сервер до проекту. Для цього створимо ще одну папку з назвою “static”, у ній - підкаталог “files”, у ньому - файл page.html:

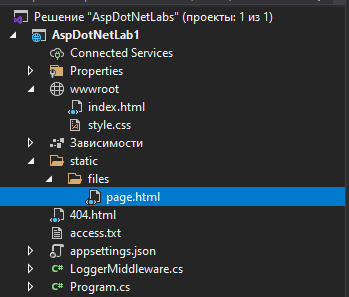


Рис. 3.6. Створена папка папку з назвою “static”, у ній - підкаталог “files”, у ньому - файл page.html.

Лістинг програмного коду методу Configure після додавання статичного серверу:

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseRouting();

app.UseFileServer();

app.UseFileServer(new FileServerOptions

{

EnableDirectoryBrowsing = true,

FileProvider = new PhysicalFileProvider(

Path.Combine(Directory.GetCurrentDirectory(), @"static"))

});

app.UseEndpoints(endpoints =>

{

endpoints.MapGet("/", async context =>

{

await context.Response.WriteAsync("Hello World!");

});

});

}

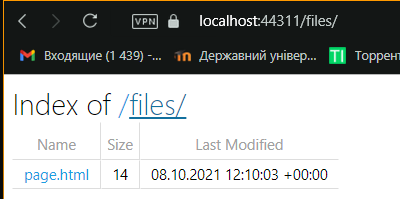


Рис. 3.7. Перевірка роботи статичного серверу

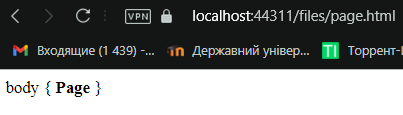


Рис. 3.8. Результат відображення при зверненні до сторінки “/page.html”

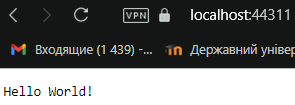


Рис. 3.9. Перевірка відображення дефолтної сторінки

**Завдання 4:** Реалізація обробки помилок.

Зпочатку переглянемо виведення помилок по замовчуванню, для цього зімітуємо виникнення помилки. Лістинг програмного коду, який імітує виникнення помилки:

app.Run(async context =>

{

int a = 0;

await context.Response.WriteAsync($"Result = {5/a}");

});



Рис. 4.1. Відображення помилки на стадії додатку “Development”

***Перший спосіб (у програмному коді).***

Використаємо middlware задля обробки помилок HTTP.

Оновлений лістинг методу Configure:

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseRouting();

app.UseFileServer();

app.UseFileServer(new FileServerOptions

{

EnableDirectoryBrowsing = true,

FileProvider = new PhysicalFileProvider(

Path.Combine(Directory.GetCurrentDirectory(), @"static"))

});

app.UseStatusCodePages();

app.UseEndpoints(endpoints =>

{

endpoints.MapGet("/", async context =>

{

await context.Response.WriteAsync("Hello World!");

});

});

}

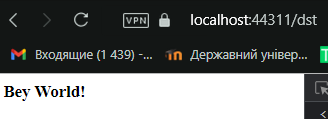
******

Рис. 4.2. – 4.3. Перевірка правильності виведення повідомлення про помилку

Проте і це не є найкращим варіантом відображення помилок, тепер реалізуємо виведення інформації про помилку через html-файли. Створимо файл web.config та додамо до нього налаштування обробки помилок. Створений файл web.config:

Лістинг файлу web.config:

<?xml version="1.0" encoding="utf-8"?>

<configuration>

<system.webServer>

<httpErrors errorMode="Custom" existingResponse="Replace">

<remove statusCode="404" />

<remove statusCode="403" />

<error statusCode="404" path="404.html" responseMode="File" />

<error statusCode="403" path="403.html" responseMode="File" />

</httpErrors>

<handlers>

<remove name="aspNetCore" />

<add name="aspNetCore" path="\*" verb="\*" modules="AspNetCoreModuleV2" resourceType="Unspecified" />

</handlers>

<aspNetCore processPath="%LAUNCHER\_PATH%" arguments="%LAUNCHER\_ARGS%" stdoutLogEnabled="false" stdoutLogFile=".\logs\stdout" hostingModel="InProcess" />

</system.webServer>

</configuration>

У кореневому каталозі файлі розмістимо файл 404.html з відповідним контентом.

Лістинг файлу 404.html:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title></title>

</head>

<body>

<b>Bey World!</b>

</body>

</html>

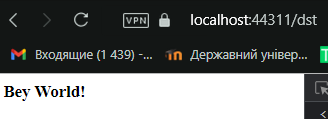
******

Рис. 4.4. Перевірка правильності виведення повідомлення про помилку

**Завдання 5:** Обробка маршрутів.

За допомогою методів Map реалізуйте обробники маршрутів /home/index та /home/about.

Лістинг оновленого методу класу Startup.cs:

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.Http;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.FileProviders;

using Microsoft.Extensions.Hosting;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Threading.Tasks;

namespace AspDotNetLab1

{

public class Startup

{

public void ConfigureServices(IServiceCollection services)

{

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseRouting();

app.UseFileServer();

app.UseFileServer(new FileServerOptions

{

EnableDirectoryBrowsing = true,

FileProvider = new PhysicalFileProvider(

Path.Combine(Directory.GetCurrentDirectory(), @"static"))

});

app.Map("/home", home =>

{

home.Map("/index", Index);

home.Map("/about", About);

});

app.UseEndpoints(endpoints =>

{

endpoints.MapGet("/", async context =>

{

await context.Response.WriteAsync("Hello World!");

});

});

}

private static void Index(IApplicationBuilder app) {

app.Run(async context =>

{

await context.Response.WriteAsync("Index");

});

}

private static void About(IApplicationBuilder app)

{

app.Run(async context =>

{

await context.Response.WriteAsync("About");

});

}

}

}

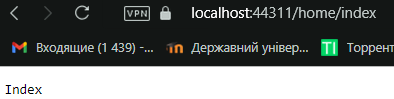


Рис. 5.1. Результат відображення сторінки за маршрутом «/home/index»

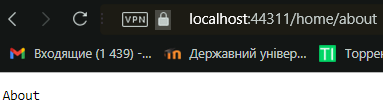


Рис. 5.2.Результат відображення сторінки за маршрутом «/home/about»

**Завдання 6:** Реалізація власних компонентів middleware.

Створіть та підключіть у проекті власний компонент з назвою LoggerMiddleware, який буде записувати інформацію про усі запити до вашого сайту у текстовий файл з назвою access.txt.

Створіть та підключіть у проекті власний компонент з назвою SecretMiddleware, який буде відловлювати декілька маршрутів, наприклад “/secret571743235872348” і виводити певне повідомлення.

Лістинг компоненту middleware LoggerMiddleware:

using Microsoft.AspNetCore.Http;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Threading.Tasks;

namespace AspDotNetLab1

{

public class LoggerMiddleware

{

private readonly RequestDelegate \_next;

public LoggerMiddleware(RequestDelegate next) {

this.\_next = next;

}

public async Task InvokeAsync(HttpContext context) {

File.AppendAllText("access.txt", $"{DateTime.Now.ToString()} {context.Request.Path}\n");

await \_next.Invoke(context);

}

}

}

Також до методу Configure у класі Startup додамо виклик реалізованого компоненту:

app.UseMiddleware<LoggerMiddleware>();

Виконаємо запити до сайту та перевіримо вміст файлу access.txt:

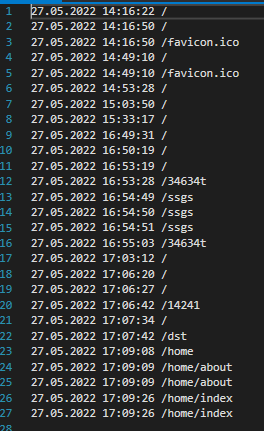


Рис. 6.1. Вміст файлу access.txt

Лістинг компоненту SecretMiddlware

using Microsoft.AspNetCore.Http;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Threading.Tasks;

namespace AspDotNetLab1

{

public class SecretMiddleware

{

private readonly RequestDelegate \_next;

public SecretMiddleware(RequestDelegate next)

{

this.\_next = next;

}

public async Task InvokeAsync(HttpContext context)

{

string path = context.Request.Path.Value.ToLower();

switch (path) {

case "/secret-1":

await context.Response.WriteAsync("secret-1");

break;

case "/secret-2":

await context.Response.WriteAsync("secret-2");

break;

case "/secret-3":

await context.Response.WriteAsync("secret-3");

break;

default:

await \_next.Invoke(context);

break;

}

}

}

}

Лістинг оновленого класу Startup.cs:

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.Http;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.FileProviders;

using Microsoft.Extensions.Hosting;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Threading.Tasks;

namespace AspDotNetLab1

{

public class Startup

{

public void ConfigureServices(IServiceCollection services)

{}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseRouting();

app.UseFileServer();

app.UseFileServer(new FileServerOptions

{

EnableDirectoryBrowsing = true,

FileProvider = new PhysicalFileProvider(

Path.Combine(Directory.GetCurrentDirectory(), @"static"))

});

app.UseMiddleware<LoggerMiddleware>();

app.UseMiddleware<SecretMiddleware>();

app.Map("/home", home =>

{

home.Map("/index", Index);

home.Map("/about", About);

});

app.UseEndpoints(endpoints =>

{

endpoints.MapGet("/", async context =>

{

await context.Response.WriteAsync("Hello World!");

});

});

}

private static void Index(IApplicationBuilder app) {

app.Run(async context =>

{

await context.Response.WriteAsync("Index");

});

}

private static void About(IApplicationBuilder app)

{

app.Run(async context =>

{

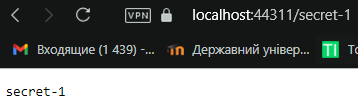
await context.Response.WriteAsync("About");

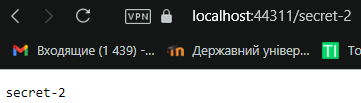
});

}

}

}





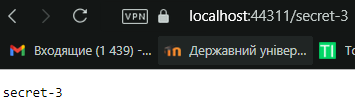


Рис. 6.2 – 6.4. Перевірка виведення компонентом SecretMiddleware повідомлення

**Завдання 7.**

Запушити виконану роботу у репозиторій на GitHub за назвою AspDotNetLab1 і відкрити доступ для викладачів:

● [morozov@ztu.edu.ua](mailto:morozov@ztu.edu.ua)

● [4ov.ztu@gmail.com](mailto:4ov.ztu@gmail.com)

Посилання на репозиторій:

<https://gitlab.com/IvanPaliy/aspdotnetlab1.git>

***Висновки:*** в ході виконання лабораторної роботи я ознайомився зі структурою ASP.NET Code додатків, навчився створювати найпростіші та використовувати наявні компоненти middleware у ASP.NET Core.