**ЛАБОРАТОРНА РОБОТА № 2**

Сервіси та Dependency Injection

**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:** Завдання виконується в новому проекті типу ASP.NET Core Empty.

Назва рішення: **AspDotNetLabs**

Назва проекту: **AspDotNetLab2**

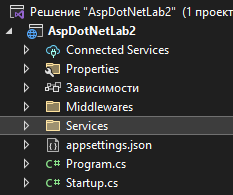


Рис. 1 Результат створення проекту.

1. Зробити обробку звернення до адреси “/services/list”, реалізувавши виведення інформації про всі сервіси, додані у додаток (тип сервісу, тип реалізації сервісу, життєвий цикл сервісу). У звіті до лабораторної роботи навести коротку інформацію про кожний сервіс (достатньо одного речення, що чітко описує сервіс). Лістинг Startup.cs:

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.Http;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using System.Text;

using AspDotNetLab2.Services.Interfaces;

using AspDotNetLab2.Services;

using AspDotNetLab2.Middlewares;

namespace AspDotNetLab2

{

public class Startup

{

private static IServiceCollection \_services;

public void ConfigureServices(IServiceCollection services)

{

\_services = services;

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseRouting();

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

{

services.Map("/list", List);

});

app.UseEndpoints(endpoints =>

{

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

{

var sb = new StringBuilder();

sb.Append("<h1>Лабораторна робота №2</h1>");

sb.Append("<ul>");

sb.Append("<li><ahref=\"/services/list\">AllServices</a></li>");

sb.Append("</ul>");

context.Response.ContentType = "text/html;charset=utf-8";

await context.Response.WriteAsync(sb.ToString());

});

});

}

public static void List(IApplicationBuilder app)

{

app.Run(async context =>

{

var sb = new StringBuilder();

sb.Append("<h1>All services</h1>");

sb.Append("<table border=\"1\">");

sb.Append("<tr><th>#</th><th>Type</th><th>Lifetime</th><th>Realisation</th></tr>");

for (var i = 0; i < \_services.Count; i++)

{

sb.Append("<tr>");

sb.Append($"<td>{i + 1}</td>");

sb.Append($"<td>{\_services[i].ServiceType.FullName}</td>");

sb.Append($"<td>{\_services[i].Lifetime}</td>");

sb.Append($"<td>{\_services[i].ImplementationType?.FullName}</td>");

sb.Append("</tr>");

}

sb.Append("</table>");

context.Response.ContentType = "text/html;charset=utf-8";

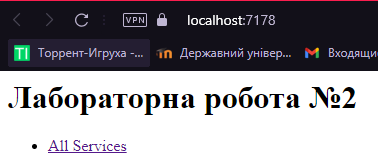
await context.Response.WriteAsync(sb.ToString());

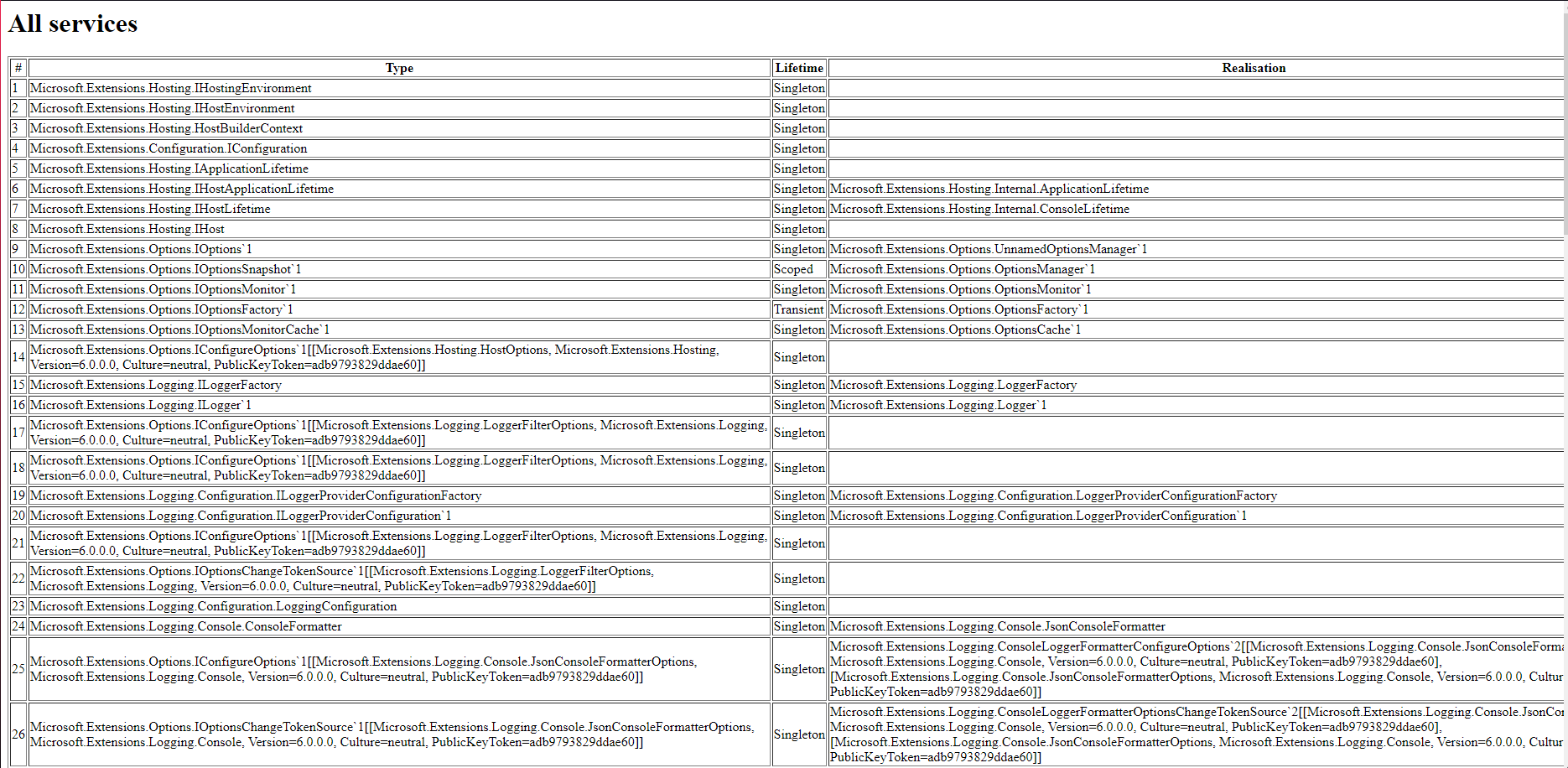
});

}

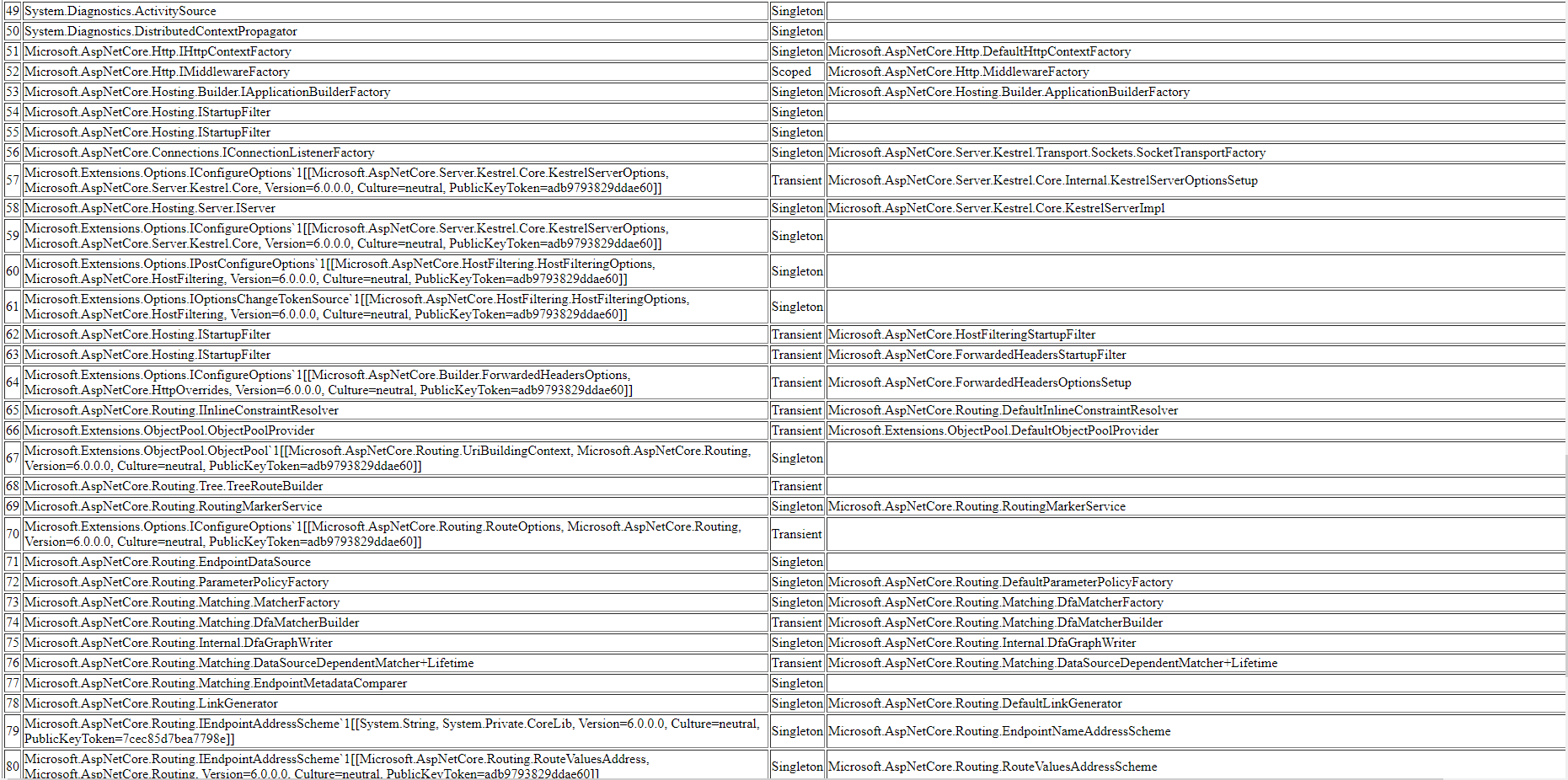
}

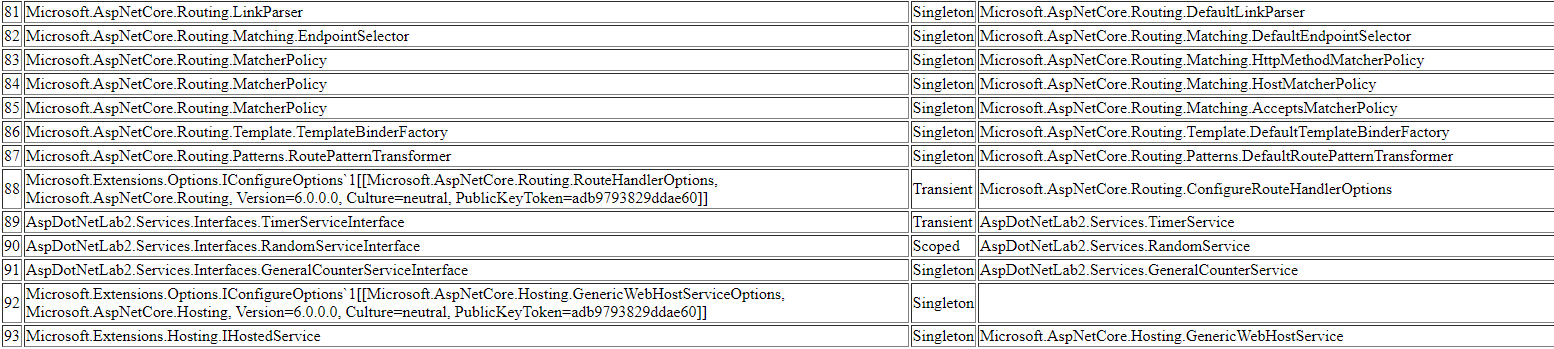
}











Опишемо перші 15 сервіси з 86 зі списку вище:

1) Microsoft.Extensions.Hosting.IHostingEnvironment — oтримує або вказує ім'я програми. Ця властивість автоматично визначається хостом для складання, що містить точку входу програми (застарілий).

2) Microsoft.Extensions.Hosting.IHostEnvironment — oтримує або встановлює назву програми. Ця властивість автоматично встановлюється хостом для збірки, що містить точку входу програми.

3) Microsoft.Extensions.Hosting.HostBuilderContext — центральне місце для обміну станом між компонентами під час процесу побудови хоста.

4) Microsoft.Extensions.Configuration.IConfiguration — представляє набір властивостей конфігурації програми ключ/значення.

5) Microsoft.Extensions.Hosting.IApplicationLifetime — дозволяє споживачам виконувати очищення під час витонченого вимкнення.

6) Microsoft.Extensions.Hosting.IHostApplicationLifetime — дозволяє споживачам отримувати сповіщення про події протягом усього життя програми.

7) Microsoft.Extensions.Hosting.IHostLifetime — контролює, коли хост запускається і коли він зупиняється. Використовується остання зареєстрована реалізація.

8) Microsoft.Extensions.Hosting.IHost — може блокувати потік виклику, доки не буде запущено вимкнення хоста та не буде зупинено всі екземпляри IHostedService.

9) Microsoft.Extensions.Options.IOptions`1 — можна передавати конфігурацію не просто як набір налаштувань у вигляді пар ключ-значення, а як об'єкти певних класів.

10) Microsoft.Extensions.Options.IOptionsSnapshot`1 — використовується для доступу до значення TOptions протягом усього терміну дії запиту.

11) Microsoft.Extensions.Options.IOptionsMonitor`1 — використовується для сповіщень про TOptionsз міну екземплярів.

12) Microsoft.Extensions.Options.IOptionsFactory`1 — використовується для створення TOptionsекземплярів.

13) Microsoft.Extensions.Options.IOptionsMonitorCache`1 — використовується IOptionsMonitor для кешування TOptionsекземплярів

14) Microsoft.Extensions.Options.IConfigureOptions`1 — використовується для налаштування TOptions.

15) Microsoft.Extensions.Logging.ILoggerFactory — представляє тип, який використовується для налаштування системи реєстрації та створення екземплярів ILogger із зареєстрованого ILoggerProvider.

**2)** Створити три сервіси з різними життєвими циклами і продемонструвати їх роботу:

**2.1)** Тип **“Transient”**, назва **“TimerService”** - повертає поточну дату та час. Результати вивести за адресою **“/services/timer”**.

**2.2)** Тип **“Scoped”**, назва **“RandomService”** - в конструкторі генерує випадкове число. За допомогою методу чи властивості повертає його. Результати роботи вивести за адресою **“/services/random”**. Має виводитися два числа, які співпадають (слайди 35-39 та 42-43 до лекції).

**2.3)** Тип **“Singleton”**, назва **“GeneralCounterService”** - веде підрахунок кількості звернень до кожної URL-адреси. Оскільки сервіс матиме тип **“Singleton”**, то підрахунок кількості буде вестися для усіх запитів. Результати вивести за адресою **“/services/general-counter”**

Лістинг інтерфейсу TimerServiceInterface:

namespace AspDotNetLab2.Services.Interfaces

{

public interface TimerServiceInterface

{

string GetCurrentDate();

}

}

Лістинг інтерфейсу TimerService:

using System;

using AspDotNetLab2.Services.Interfaces;

namespace AspDotNetLab2.Services

{

public class TimerService : TimerServiceInterface

{

private GeneralCounterServiceInterface \_generalCounterService;

public TimerService(GeneralCounterServiceInterface generalCounterService)

{

\_generalCounterService = generalCounterService;

}

public string GetCurrentDate()

{

\_generalCounterService.IncrementRequests();

return DateTime.Now.ToString("m/d/yyyy hh:mm:ss");

}

}

}

Лістинг інтерфейсу RandomServiceInterface:

namespace AspDotNetLab2.Services.Interfaces

{

public interface RandomServiceInterface

{

double RandomValue { get; }

}

}

Лістинг інтерфейсу RandomService:

using System;

using AspDotNetLab2.Services.Interfaces;

namespace AspDotNetLab2.Services

{

public class RandomService : RandomServiceInterface

{

private GeneralCounterServiceInterface \_generalCounterService;

private static Random \_random;

private double \_value;

public RandomService(GeneralCounterServiceInterface generalCounterService)

{

\_generalCounterService = generalCounterService;

\_random = new Random();

\_value = Math.Round(\_random.NextDouble(), 3);

\_generalCounterService.IncrementRequests();

}

public double RandomValue => \_value;

}

}

Лістинг класу GeneralCounterServiceInterface:

namespace AspDotNetLab2.Services.Interfaces

{

public interface GeneralCounterServiceInterface

{

int RequestsValue { get; }

void IncrementRequests();

}

}

Лістинг класу GeneralCounterService:

using AspDotNetLab2.Services.Interfaces;

namespace AspDotNetLab2.Services

{

public class GeneralCounterService : GeneralCounterServiceInterface

{

private int \_value;

public GeneralCounterService()

{

\_value = 0;

}

public int RequestsValue => \_value;

public void IncrementRequests()

{

\_value += 1;

}

}

}

Лістинг Startup.cs

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.Http;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using System.Text;

using AspDotNetLab2.Services.Interfaces;

using AspDotNetLab2.Services;

using AspDotNetLab2.Middlewares;

namespace AspDotNetLab2

{

public class Startup

{

private static IServiceCollection \_services;

public void ConfigureServices(IServiceCollection services)

{

\_services = services;

services.AddTransient<TimerServiceInterface, TimerService>();

services.AddScoped<RandomServiceInterface, RandomService>();

services.AddSingleton<GeneralCounterServiceInterface, GeneralCounterService>();

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseRouting();

app.UseGeneralCounter();

app.UseTimer();

app.UseRandom();

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

{

services.Map("/list", List);

});

app.UseEndpoints(endpoints =>

{

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

{

var sb = new StringBuilder();

sb.Append("<h1>Лабораторна робота №2</h1>");

sb.Append("<ul>");

sb.Append("<li><a href=\"/services/list\">All Services</a></li>");

sb.Append("<li><a href=\"/services/timer\">Date and Time</a></li>");

sb.Append("<li><a href=\"/services/random\">Random count</a></li>");

sb.Append("<li><a href=\"/services/general-counter\">Number of requests</a></li>");

sb.Append("</ul>");

context.Response.ContentType = "text/html;charset=utf-8";

await context.Response.WriteAsync(sb.ToString());

});

});

}

public static void List(IApplicationBuilder app)

{

app.Run(async context =>

{

var sb = new StringBuilder();

sb.Append("<h1>All services</h1>");

sb.Append("<table border=\"1\">");

sb.Append("<tr><th>#</th><th>Type</th><th>Lifetime</th><th>Realisation</th></tr>");

for (var i = 0; i < \_services.Count; i++)

{

sb.Append("<tr>");

sb.Append($"<td>{i + 1}</td>");

sb.Append($"<td>{\_services[i].ServiceType.FullName}</td>");

sb.Append($"<td>{\_services[i].Lifetime}</td>");

sb.Append($"<td>{\_services[i].ImplementationType?.FullName}</td>");

sb.Append("</tr>");

}

sb.Append("</table>");

context.Response.ContentType = "text/html;charset=utf-8";

await context.Response.WriteAsync(sb.ToString());

});

}

}

}

Результат виконання програми:

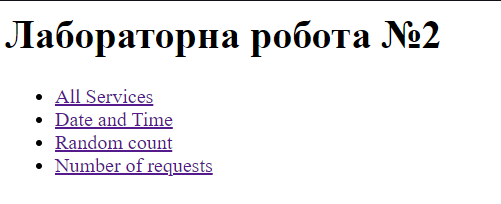


Рис. 2.1. Запуск програми

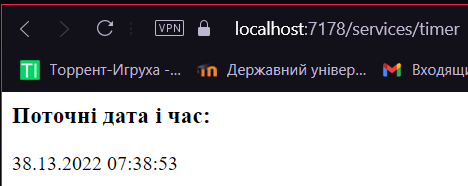


Рис. 2.7. Результат виконання time service

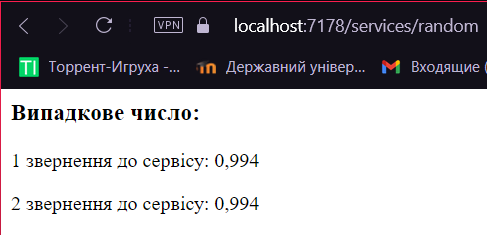
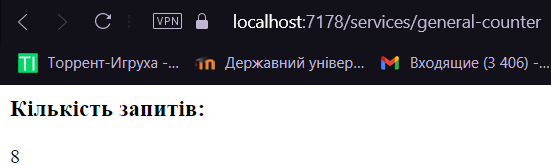


Рис. 2.8. Результат виконання random service

В процесі обробки запиту навіть при двох різних звернень використовується один об’єкт, відповідно однаковий результат. Це зумовлено життєвим циклом сервісу “Scoped”.



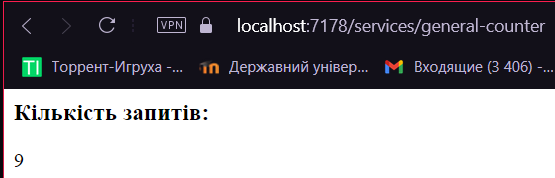


Рис. 2.9 – 2.10. Результат виконання general counter service до та після оновлення сторінки

1. Для кожного сервісу створити відповідний middleware.

Лістинг TimerMiddleware.cs:

using System.Threading.Tasks;

using AspDotNetLab2.Services.Interfaces;

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Http;

namespace AspDotNetLab2.Middlewares

{

public class TimerMiddleware

{

private readonly RequestDelegate \_next;

public TimerMiddleware(RequestDelegate next)

{

\_next = next;

}

public async Task InvokeAsync(HttpContext context, TimerServiceInterface timerService)

{

if (context.Request.Path.Value.ToLower() == "/services/timer")

{

context.Response.ContentType = "text/html; charset=utf-8";

await context.Response.WriteAsync($"<h3>Поточні дата і час:</h3> {timerService?.GetCurrentDate()}");

}

else

{

if (\_next != null)

{

await \_next.Invoke(context);

}

}

}

}

public static class TimerMiddlewareExtension

{

public static IApplicationBuilder UseTimer

(this IApplicationBuilder builder)

{

return builder.UseMiddleware<TimerMiddleware>();

}

}

}

Лістинг RandomMiddleware.cs:

using System.Text;

using System.Threading.Tasks;

using AspDotNetLab2.Services.Interfaces;

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Http;

using Microsoft.Extensions.DependencyInjection;

namespace AspDotNetLab2.Middlewares

{

public class RandomMiddleware

{

private readonly RequestDelegate \_next;

public RandomMiddleware(RequestDelegate next)

{

\_next = next;

}

public async Task InvokeAsync(HttpContext context, RandomServiceInterface firstRandomService)

{

{

if (context.Request.Path.Value.ToLower() == "/services/random")

{

var sb = new StringBuilder();

sb.Append("<h3>Випадкове число:</h3>");

sb.Append("<p> 1 звернення до сервісу:");

sb.Append($" {firstRandomService?.RandomValue} </p>");

sb.Append("<p> 2 звернення до сервісу:");

var secondRandomService =

context.RequestServices.GetService<RandomServiceInterface>();

sb.Append($" {secondRandomService?.RandomValue} </p>");

context.Response.ContentType = "text/html;charset=utf-8";

await context.Response.WriteAsync(sb.ToString());

}

else

{

if (\_next != null)

{

await \_next.Invoke(context);

}

}

}

}

}

public static class RandomMiddlewareExtension

{

public static IApplicationBuilder UseRandom

(this IApplicationBuilder builder)

{

return builder.UseMiddleware<RandomMiddleware>();

}

}

}

Лістинг GeneralCounterMiddleware.cs:

using System.Threading.Tasks;

using AspDotNetLab2.Services.Interfaces;

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Http;

namespace AspDotNetLab2.Middlewares

{

public class GeneralCounterMiddleware

{

private readonly RequestDelegate \_next;

private GeneralCounterServiceInterface \_generalCounterService;

public GeneralCounterMiddleware(RequestDelegate next, GeneralCounterServiceInterface generalCounterService)

{

\_next = next;

\_generalCounterService = generalCounterService;

}

public async Task InvokeAsync(HttpContext context)

{

if (context.Request.Path.Value.ToLower() == "/services/general-counter")

{

\_generalCounterService.IncrementRequests();

context.Response.ContentType = "text/html; charset=utf-8";

await context.Response.WriteAsync($"<h3>Кількість запитів:</h3> {\_generalCounterService?.RequestsValue}");

}

else

{

if (\_next != null)

{

await \_next.Invoke(context);

}

}

}

}

public static class GeneralCounterMiddlewareExtension

{

public static IApplicationBuilder UseGeneralCounter

(this IApplicationBuilder builder)

{

return builder.UseMiddleware<GeneralCounterMiddleware>();

}

}

}

Результат виконання програми:

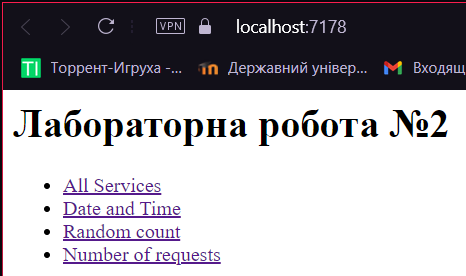


Рис. 3.1. Запуск програми.

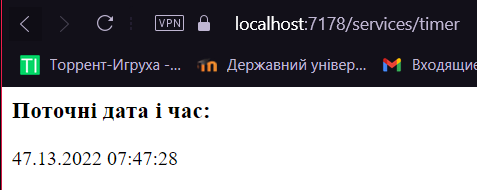


Рис. 3.2. Результат виконання програми TimerService:

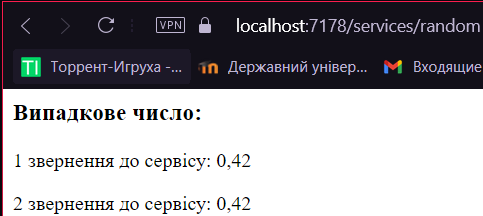


Рис. 3.3. Результат виконання програми RandomService

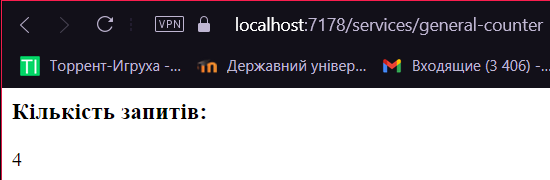


Рис. 3.4 - 3.5. Результат виконання GeneralCounterService до та після оновлення сторінки

**Завдання 3.** 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://github.com/IvanPaliy/AspDotNetLab2.git>

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