ASP.NET CORE MVC MODULE 03

CONFIGURE MIDDLEWARE AND SERVICES IN ASP.NET CORE

Summer 2021 – Web Development using ASP .Net Core MVC

MAIN SOURCES FOR THESE SLIDED

- Unless otherwise specified, the main sources for these slides are:
  - https://github.com/MicrosoftLearning/20486D-DevelopingASPNETMVCWebApplications ←for homework
  - https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-5.0 ←for "textbook"

THE STARTUP CLASS

- Also see pages 962 and 976 …

- The **Startup** class is **created by default** with <u>every web application</u> and is used to:
  - Configure services
  - Configure middleware
  - The application starts with **Main**, which is found in **program.cs**. This calls and runs the **Startup** class.

- **ConfigureServices method** ← used to configure **services**
  - Is optional
  - Used to register **services** (**IMPORTANT:** order does not matter)
    (**services** = reusable components available to the entire app via **dependency injection**)
  - Called before **Configure**

- **Configure method** ← used to configure middleware (the **request handling pipeline**)
  - specifies how the app should respond to HTTP requests
  - The request pipeline is configured by adding **middleware components** to an **IApplicationBuilder** instance.
  - Each **Use…** extension method adds middleware component(s) to the request pipeline (**IMPORTANT:** order matters)
    - As an example, **UseStaticFiles** configures the middleware to serve static files (we'll see details below).

EMPTY VS MVC

Empty

An empty project template for creating an ASP.NET Core application. This template does not have any content in it.

Web Application (Model-View-Controller)

A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.

Solution Explorer

Search Solution Explorer (Ctrl+;)

- Solution 'WebApplication2' (1 of 1 project)
  - WebApplication2
    - Connected Services
    - Dependencies
    - Properties
    - appsettings.json
    - Program.cs
    - Startup.cs

```
public class Startup
{
    // This method gets called by the runtime. Use this method to add services to the container.
    // For more information on how to configure your application, visit https://go.microsoft.com/fwlink/?LinkID=398940
    public void ConfigureServices(IServiceCollection services)
    {
    }

    // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }

        app.UseRouting();

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapGet("/", async context =>
            {
                await context.Response.WriteAsync("Hello World!");
            });
        });
    }
}
```

```
public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    // This method gets called by the runtime. Use this method to add services to th
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddControllersWithViews();
    }

    // This method gets called by the runtime. Use this method to configure the HTTP
    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }
        else
        {
            app.UseExceptionHandler("/Home/Error");
            // The default HSTS value is 30 days. You may want to change this for pr
            app.UseHsts();
        }
        app.UseHttpsRedirection();
        app.UseStaticFiles();

        app.UseRouting();

        app.UseAuthorization();

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

Solution Explorer

Search Solution Explorer (Ctrl+;)

- Solution 'WebApplication1' (1 of 1 project)
  - WebApplication1
    - Connected Services
    - Dependencies
    - Properties
    - wwwroot
      - css
      - js
      - lib
      - favicon.ico
    - Controllers
      - HomeController.cs
    - Models
      - ErrorViewModel.cs
    - Views
      - Home
        - Index.cshtml
        - Privacy.cshtml
      - Shared
        - _Layout.cshtml
        - _ValidationScriptsPartial.cshtml
        - Error.cshtml
      - _ViewImports.cshtml
      - _ViewStart.cshtml
    - appsettings.json
    - Program.cs
    - Startup.cs

ASP.NET CORE MIDDLEWARE

- Also see pages 1018 …

- **Middleware** is an app pipeline used to handle **requests** and **responses**
  - Each middleware component has access to an **HTTPContext** parameter (it has **request** & **response** props).

```
(parameter) HttpContext context
```

- Each middleware component:
  - Can choose whether to **pass** the request to the next component in the pipeline.
  - Can choose to perform work **before** and **after** the next component in the pipeline.

- Let's start with the simplest middleware example (that only includes a single anonymous function):
  - This will always respond with `"Hello from the Evergreen State!"`, regardless of the request URL

```
// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
public void Configure(IApplicationBuilder app)
{

    app.Run(async (context) =>
    {
        await context.Response.WriteAsync("Hello from the Evergreen State!");
    });

}
```

  - Test this code with multiple paths
  - Change the output to
    `"Hello from the Evergreen State!"+ context.Request.Path`

ASP.NET CORE MIDDLEWARE (2)

- To add middleware to the pipeline, use the **IApplicationBuilder** parameter.

```
// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
public void Configure(IApplicationBuilder app)
{

    app.Run(async (context) =>
    {
        await context.Response.WriteAsync("Hello from the Evergreen State!");
    });

}
```

- Request delegates are configured using **Run**, **Map**, and **Use** extension methods.
  - Each delegate can include operations **before** and **after** the next delegate is called.
  - **Use middleware**: allows a parameter, **next**, which is a reference to the next middleware in the pipeline.
    - Can chain multiple request delegates: use **next.Invoke()** to proceed to the next middleware in the pipeline.
    - If a **Use** middleware does not call **next.Invoke()** then it is said to **short-circuit** the pipeline (for example: access denied)
  - **Run middleware**: will always be the final middleware in the pipeline (it does not support **next**).
    - Run middleware should always be placed at the very bottom of the middleware pipeline.
  - **Map middleware**: branches the request pipeline based on matches of the given request **path**.
    - This isn't frequently used. Nested Maps are allowed.

See examples below…

EXAMPLE (1):

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
        if (env.IsDevelopment())      //use this if you get errors and want to debug them … details later
        {
            app.UseDeveloperExceptionPage();
        }

        app.Use(async (context, next) =>
        {
            await context.Response.WriteAsync("Hello from Middleware component 1 start\n");
            await next.Invoke();
            await context.Response.WriteAsync(" Hello from Middleware component 1 end\n");
        });

        app.Run(async (context) =>
        {
            await context.Response.WriteAsync(" Hello from Middleware component 2\n");
        });
    }
```

Source: https://www.lynda.com/ASP-NET-tutorials/ASP-NET-Core-Middleware/647683-2.html

EXAMPLE (2):

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }

        app.Use(async (context, next) =>
        {
            await context.Response.WriteAsync("Hello from Middleware component 1 start\n");
            ~~await next.Invoke(); //what happens if we leave this out?~~
            await context.Response.WriteAsync(" Hello from Middleware component 1 end\n");
        });

        app.Run(async (context) =>
        {
            await context.Response.WriteAsync(" Hello from Middleware component 2\n");
        });
    }
```

Source: https://www.lynda.com/ASP-NET-tutorials/ASP-NET-Core-Middleware/647683-2.html

EXAMPLE (3):

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }

        app.Run(async (context) =>
        {
            await context.Response.WriteAsync(" Hello from Middleware component 2\n");
        });

        app.Use(async (context, next) =>
        {
            await context.Response.WriteAsync("Hello from Middleware component 1 start\n");
            await next.Invoke();
            await context.Response.WriteAsync(" Hello from Middleware component 1 end\n");
        });
}
```

Source: https://www.lynda.com/ASP-NET-tutorials/ASP-NET-Core-Middleware/647683-2.html

EXAMPLE (4):

```
public void Configure(IApplicationBuilder app)
{
        private static void HandleMapTest1(IApplicationBuilder app)
        {
                app.Run(async context =>
                {
                        await context.Response.WriteAsync("Map Test 1");
                });
        }

        private static void HandleMapTest2(IApplicationBuilder app)
        {
                app.Run(async context =>
                {
                        await context.Response.WriteAsync("Map Test 2");
                });
        }

        public void Configure(IApplicationBuilder app)
        {
                app.Map("/map1", HandleMapTest1);
                app.Map("/map2", HandleMapTest2);
                app.Run(async context =>
                {
                        await context.Response.WriteAsync("Hello from non-Map delegate. <p>");
                 });
        }
}
```

REQUEST	RESPONSE
localhost:1234	Hello from non-Map delegate.
localhost:1234/map1	Map Test 1
localhost:1234/map2	Map Test 2
localhost:1234/map3	Hello from non-Map delegate.

Source: page 1024

BUILT-IN MIDDLEWARE

- Check out pages 1022 and 1026.
- We'll see some of them in this course.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
        app.UseDatabaseErrorPage();
    }
    else
    {
        app.UseExceptionHandler("/Error");
        app.UseHsts();
    }

    app.UseHttpsRedirection();
    app.UseStaticFiles();
    app.UseCookiePolicy();
    app.UseRouting();
    app.UseAuthentication();
    app.UseAuthorization();
    app.UseSession();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapRazorPages();
    });
}
```

MIDDLEWARE	DESCRIPTION	ORDER
Authentication	Provides authentication support.	Before `HttpContext.User` is needed. Terminal for OAuth callbacks.
Authorization	Provides authorization support.	Immediately after the Authentication Middleware.
Cookie Policy	Tracks consent from users for storing personal information and enforces minimum standards for cookie fields, such as `secure` and `SameSite`.	Before middleware that issues cookies. Examples: Authentication, Session, MVC (TempData).
CORS	Configures Cross-Origin Resource Sharing.	Before components that use CORS. `UseCors` currently must go before `UseResponseCaching` due to this bug.
Diagnostics	Several separate middlewares that provide a developer exception page, exception handling, status code pages, and the default web page for new apps.	Before components that generate errors. Terminal for exceptions or serving the default web page for new apps.
Forwarded Headers	Forwards proxied headers onto the current request.	Before components that consume the updated fields. Examples: scheme, host, client IP, method.
Health Check	Checks the health of an ASP.NET Core app and its dependencies, such as checking database availability.	Terminal if a request matches a health check endpoint.
Header Propagation	Propagates HTTP headers from the incoming request to the outgoing HTTP Client requests.	
HTTP Method Override	Allows an incoming POST request to override the method.	Before components that consume the updated method.
HTTPS Redirection	Redirect all HTTP requests to HTTPS.	Before components that consume the URL.
HTTP Strict Transport Security (HSTS)	Security enhancement middleware that adds a special response header.	Before responses are sent and after components that modify requests. Examples: Forwarded Headers, URL Rewriting.
MVC	Processes requests with MVC/Razor Pages.	Terminal if a request matches a route.
...		
Static Files	Provides support for serving static files and directory browsing.	Terminal if a request matches a file.
URL Rewrite	Provides support for rewriting URLs and redirecting requests.	Before components that consume the URL.
WebSockets	Enables the WebSockets protocol.	Before components that are required to accept WebSocket requests.

IN-CLASS DEMO: HOW TO CREATE CUSTOM MIDDLEWARE

Demonstration: How to Create Custom Middleware

- Source/Steps
- https://github.com/MicrosoftLearning/20486D-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486D_MOD03_DEMO.md#demonstration-how-to-create-custom-middleware

DEMO - STEPS

Configure your new project

ASP.NET Core Web Application C# Linux macOS Windows Cloud Service Web

Project name

ConfigureMiddlewareExample

Location

L:\ASP_TESC\Allfiles\Mod03\Democode\01_ConfigureMiddlewareExample_begin ...

Solution name

ConfigureMiddlewareExample

☐ Place solution and project in the same directory

Create a new ASP.NET Core web application

.NET Core | ASP.NET Core 3.1

Empty
An empty project template for creating an ASP.NET Core application. This template does not have any content in it.

API
A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.

Web Application
A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content.

Web Application (Model-View-Controller)
A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.

Angular
A project template for creating an ASP.NET Core application with Angular

React.js

Get additional project templates

Authentication
No Authentication
Change

Advanced
☑ Configure for HTTPS
☐ Enable Docker Support
(Requires Docker Desktop)
Linux

Author: Microsoft
Source: Templates 3.1.13

Back | Create

DEMO - STEPS

Solution Explorer

Search Solution Explorer (Ctrl+;)

- Solution 'ConfigureMiddlewareExample' (1 of 1 project)
  - **ConfigureMiddlewareExample**
    - Connected Services
    - Dependencies
    - Properties
    - appsettings.json
    - Program.cs
    - Startup.cs

Startup.cs

ConfigureMiddlewareExample | ConfigureMiddlewareExample.Startup

```
7   using System.Collections.Generic;
8   using System.Linq;
9   using System.Threading.Tasks;
10
11  namespace ConfigureMiddlewareExample
12  {
13      public class Startup
14      {
15          // This method gets called by the runtime. Use this method to add services to the container.
16          // For more information on how to configure your application, visit https://go.microsoft.com/fwlink/
17          public void ConfigureServices(IServiceCollection services)
18          {
19          }
20
21          // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
22          public void Configure(IApplicationBuilder app)
23          {
24              app.Run(async (context) =>
25              {
26                  await context.Response.WriteAsync("This text was generated by the app.Run middleware.");
27              });
28          }
29      }
30  }
```

https://localhost:44324

localhost:44324

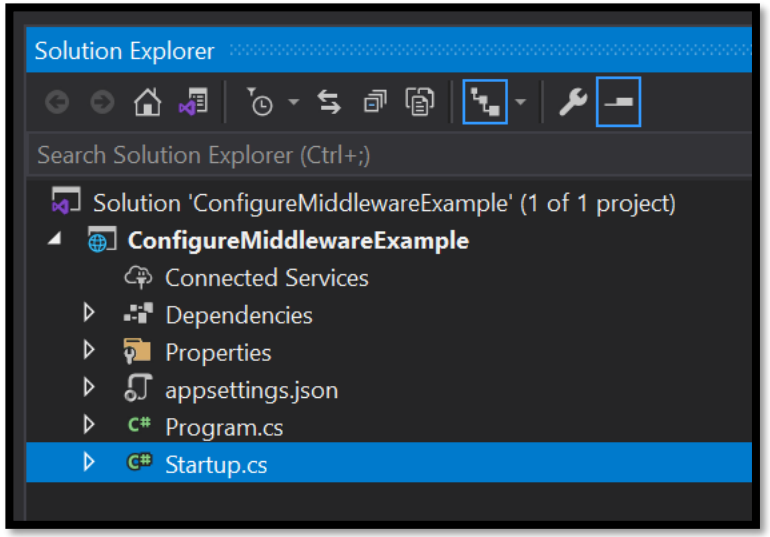
This text was generated by the app.Run middleware.

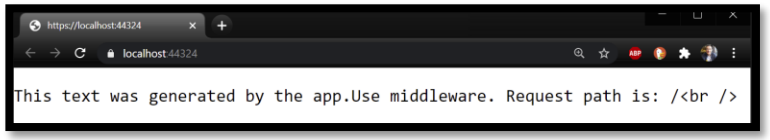
https://localhost:44324/UrlTest1

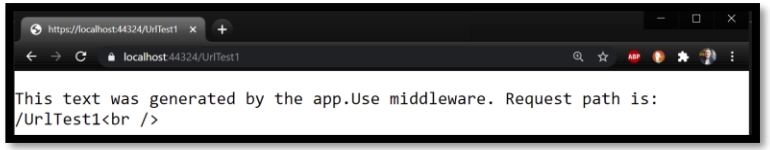
localhost:44324/UrlTest1

This text was generated by the app.Run middleware.

DEMO - STEPS





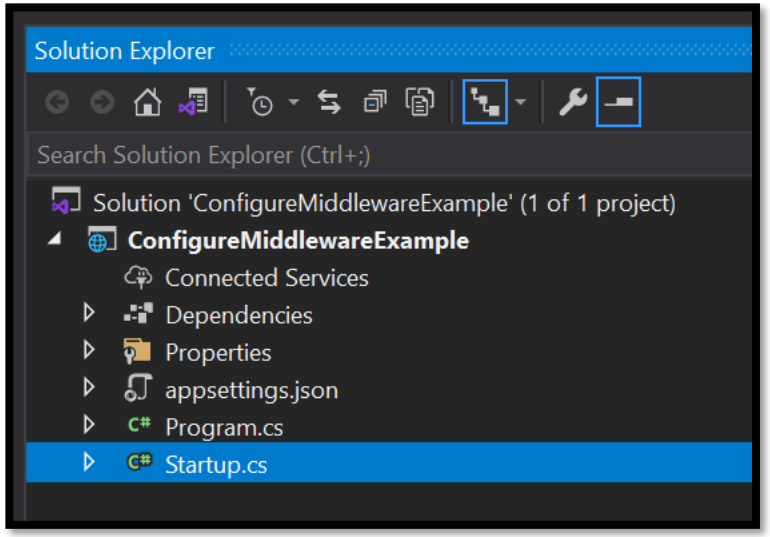


Startup.cs

ConfigureMiddlewareExample | ConfigureMiddlewareExample.Startup

```
7   using System.Collections.Generic;
8   using System.Linq;
9   using System.Threading.Tasks;
10
11  namespace ConfigureMiddlewareExample
12  {
13      public class Startup
14      {
15          // This method gets called by the runtime. Use this method to add services to the container.
16          // For more information on how to configure your application, visit https://go.microsoft.com/fwlink/?LinkID=398940
17          public void ConfigureServices(IServiceCollection services)
18          {
19          }
20
21          // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
22          public void Configure(IApplicationBuilder app)
23          {
24              app.Use(async (context, next) =>
25              {
26                  await context.Response.WriteAsync("This text was generated by the app.Use middleware. Request path is: "
27                                                    + context.Request.Path.Value + "<br />");
28              });
29
30
31              app.Run(async (context) =>
32              {
33                  await context.Response.WriteAsync("This text was generated by the app.Run middleware.");
34
35              });
36          }
37      }
38  }
```

DEMO - STEPS

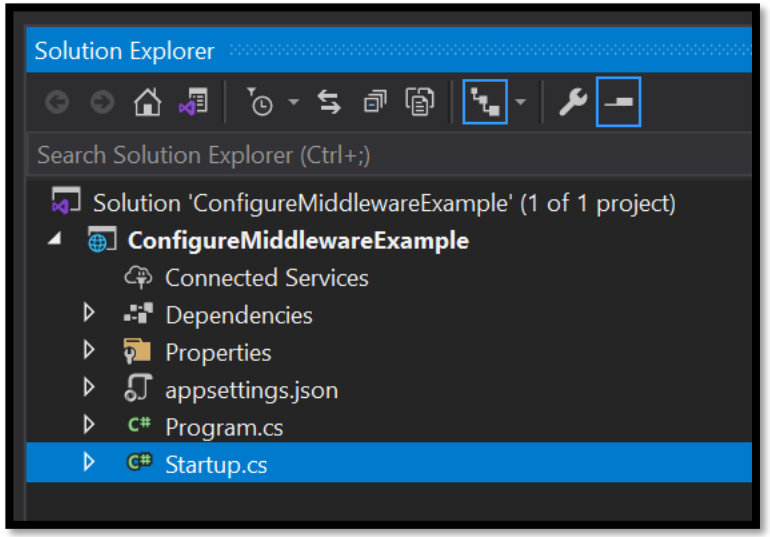


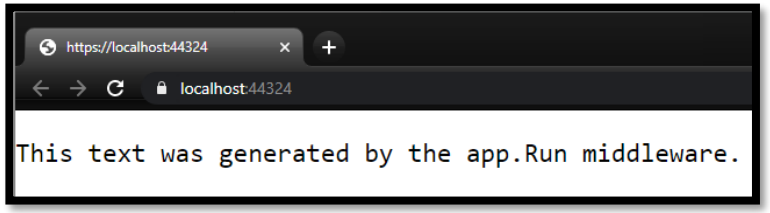
Startup.cs

ConfigureMiddlewareExample | ConfigureMiddlewareExample.Startup

```
7   using System.Collections.Generic;
8   using System.Linq;
9   using System.Threading.Tasks;
10
11  namespace ConfigureMiddlewareExample
12  {
13      public class Startup
14      {
15          // This method gets called by the runtime. Use this method to add services to the container.
16          // For more information on how to configure your application, visit https://go.microsoft.com/fwlink/?LinkID=39894
17          public void ConfigureServices(IServiceCollection services)
18          {
19          }
20
21          // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
22          public void Configure(IApplicationBuilder app)
23          {
24              app.Use(async (context, next) =>
25              {
26                  await context.Response.WriteAsync("This text was generated by the app.Use middleware. Request path is: "
27                                                    + context.Request.Path.Value + "<br />");
28                  await next.Invoke();
29              });
30
31
32              app.Run(async (context) =>
33              {
34                  await context.Response.WriteAsync("This text was generated by the app.Run middleware.");
35
36              });
37          }
38      }
39  }
```

DEMO - STEPS



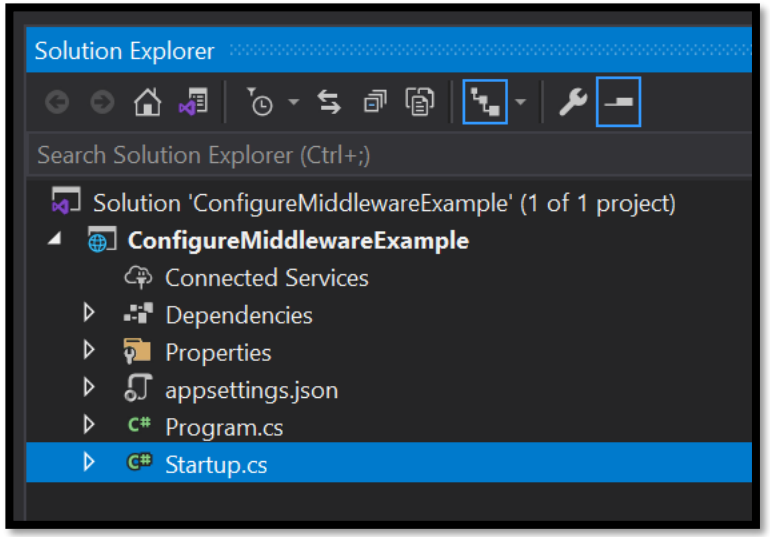


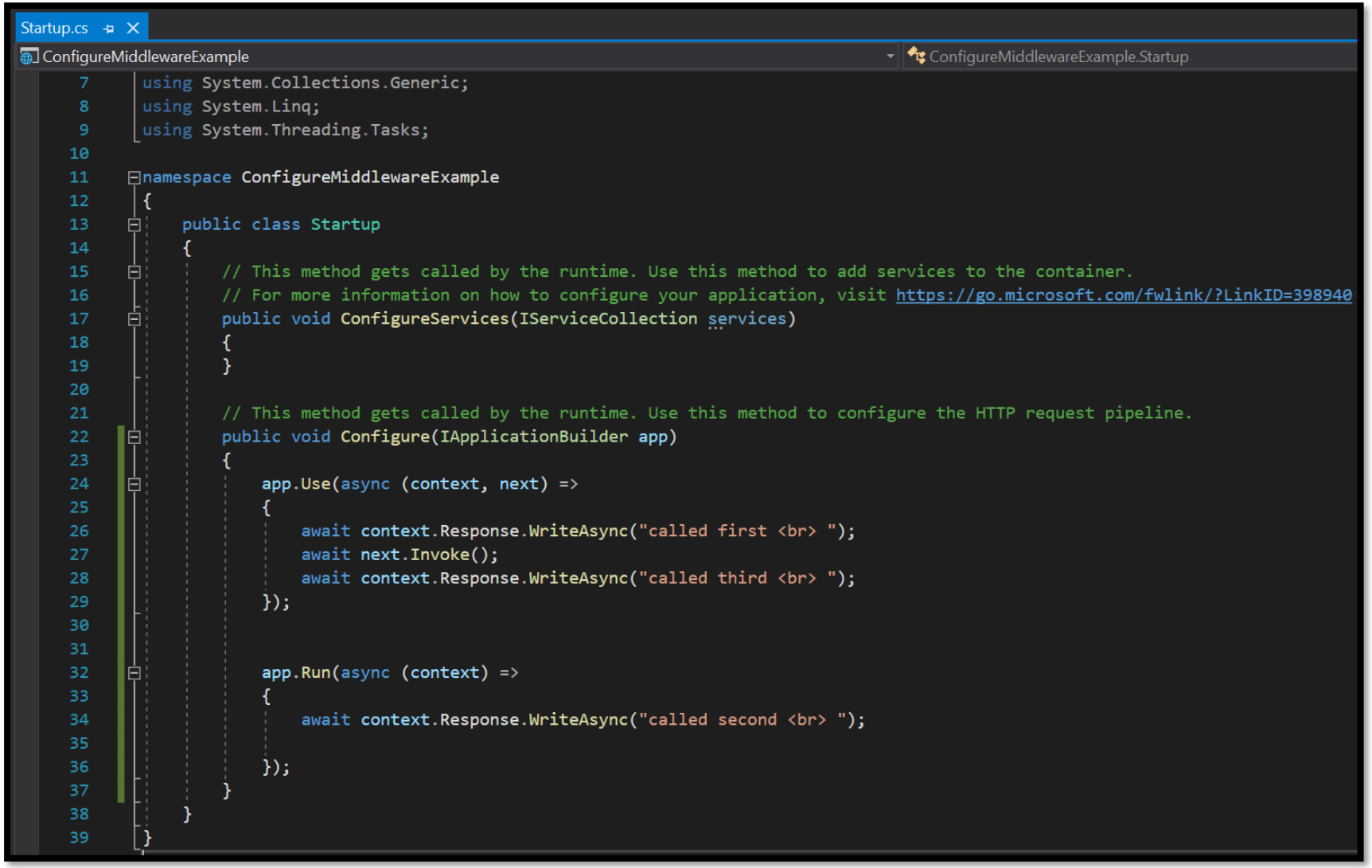
Startup.cs

ConfigureMiddlewareExample | ConfigureMiddlewareExample.Startup

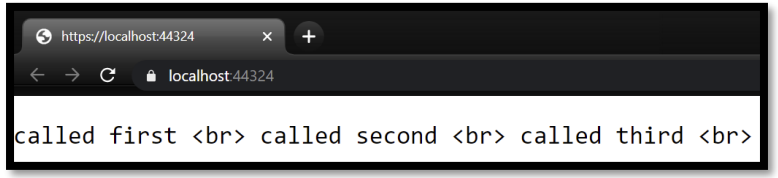
```
7   using System.Collections.Generic;
8   using System.Linq;
9   using System.Threading.Tasks;
10
11  namespace ConfigureMiddlewareExample
12  {
13      public class Startup
14      {
15          // This method gets called by the runtime. Use this method to add services to the container.
16          // For more information on how to configure your application, visit https://go.microsoft.com/fwlink/?LinkID=398940
17          public void ConfigureServices(IServiceCollection services)
18          {
19          }
20
21          // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
22          public void Configure(IApplicationBuilder app)
23          {
24              app.Run(async (context) =>
25              {
26                  await context.Response.WriteAsync("This text was generated by the app.Run middleware.");
27
28              });
29
30              app.Use(async (context, next) =>
31              {
32                  await context.Response.WriteAsync("This text was generated by the app.Use middleware. Request path is: "
33                                                    + context.Request.Path.Value + "<br />");
34                  await next.Invoke();
35              });
36
37          }
38      }
39  }
```

DEMO – STEPS - EXTRA



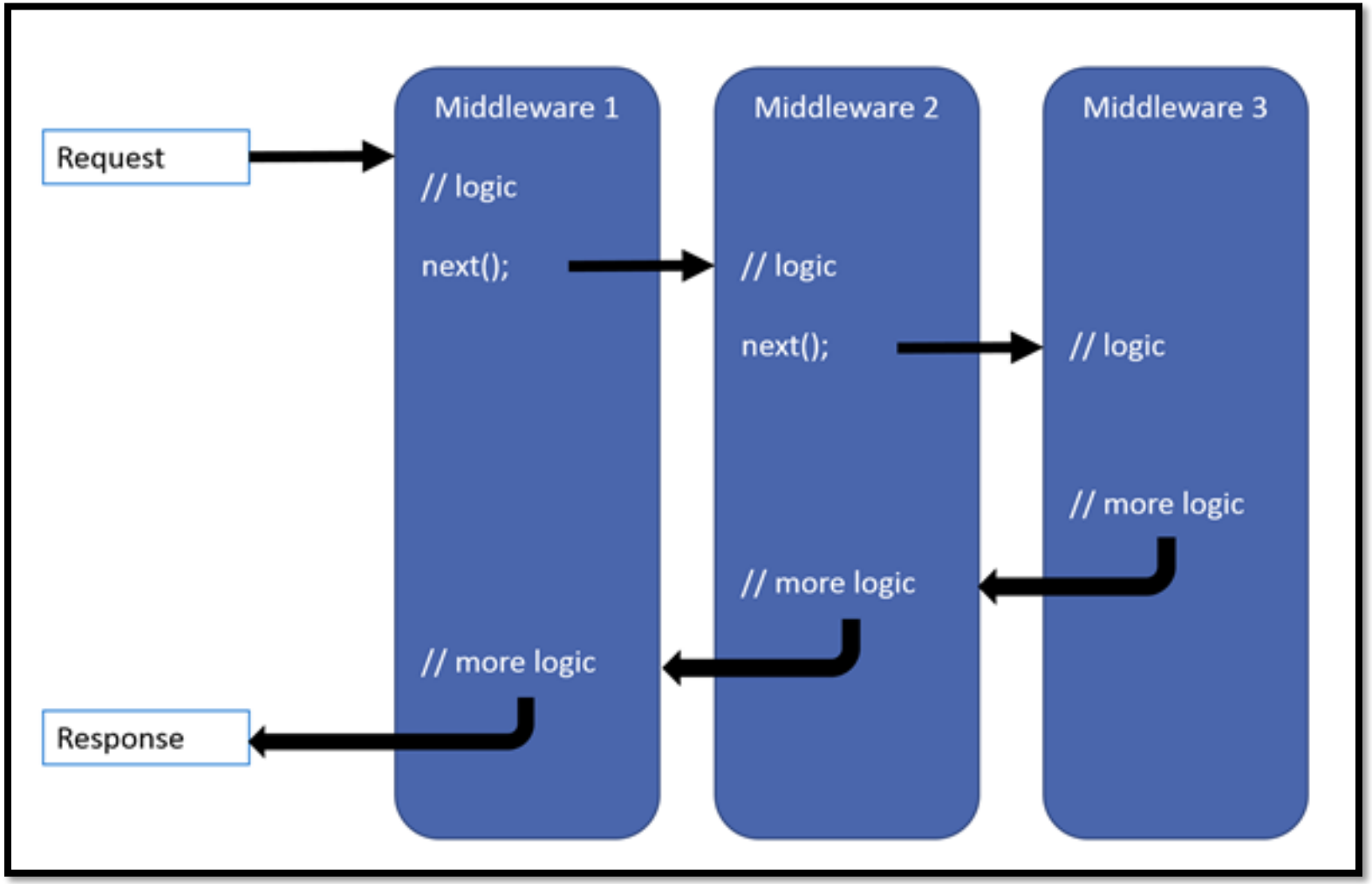



```
7   using System.Collections.Generic;
8   using System.Linq;
9   using System.Threading.Tasks;
10
11  namespace ConfigureMiddlewareExample
12  {
13      public class Startup
14      {
15          // This method gets called by the runtime. Use this method to add services to the container.
16          // For more information on how to configure your application, visit https://go.microsoft.com/fwlink/?LinkID=398940
17          public void ConfigureServices(IServiceCollection services)
18          {
19          }
20
21          // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
22          public void Configure(IApplicationBuilder app)
23          {
24              app.Use(async (context, next) =>
25              {
26                  await context.Response.WriteAsync("called first <br> ");
27                  await next.Invoke();
28                  await context.Response.WriteAsync("called third <br> ");
29              });
30
31
32              app.Run(async (context) =>
33              {
34                  await context.Response.WriteAsync("called second <br> ");
35
36              });
37          }
38      }
39  }
```



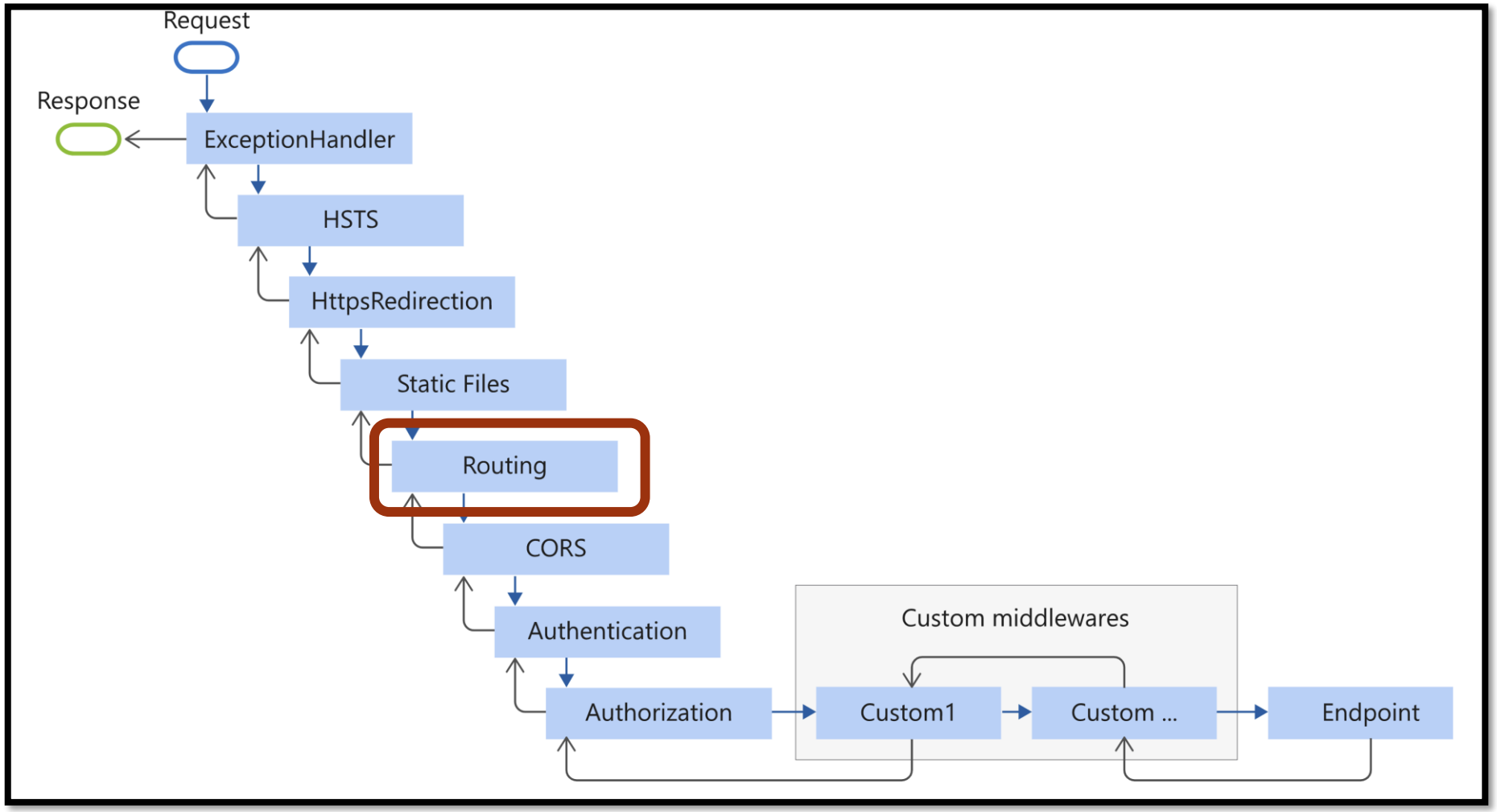
SEE ALSO

Source: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-5.0



LATER WE'LL SEE ...

Source: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-5.0



STATIC FILES IN ASP.NET CORE

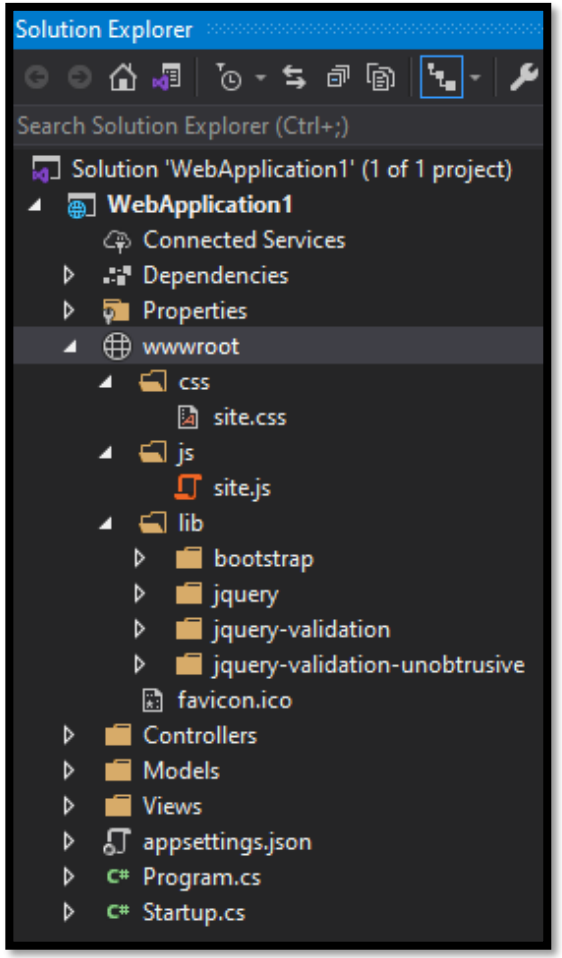
- See page 1384
- **Static files** are files that do not change at run time, hence they are served directly to clients
  - Examples: some HTML files, CSS, images, and JavaScript files. Another example: the company's logo.

- **Static files** are stored inside a project's **web root directory**.
  - The default directory for the web root directory is {content root}/wwwroot
  - Using **UseWebRoot** method, one can change the location of the web root directory

- **Static files** are accessible via a path relative to the web root.
  - If you store an image stored at: ***wwwroot****/images/MyFancyImage.jpg*
  - Then you can access it as: **https://<hostname>**/images/ *MyFancyImage.jpg*

- To enable the use of **static files** use the **UseStaticFiles** middleware

```
public void Configure(IApplicationBuilder app)
{
        app.UseStaticFiles();
        app.Run(async (context) =>   {await context.Response.WriteAsync(" Your static file was not found"); });
}
```



SERVE FILES OUTSIDE OF WEB ROOT

- Please read on your own how to allow an application to serve static files from outside of the **webroot**
  - See page 1385 for an example.

- Important:
  - For static files, no **authorization checks** are performed. They are **publicly** accessible.
  - If you need them behind **authorization**, you'll need to store them outside **wwwroot**.

- On wwwroot, **directory browsing** is disabled by default (for security reasons)
  - See pages 1389-1390 for how to enable **directory browsing**.
  - **AddDirectoryBrowser** in **ConfigureServices**.
  - **UseDirectoryBrowser** in **Configure**.

- If interested, check also: "Serve files from multiple locations" (page 1397)

SETTING A DEFAULT PAGE

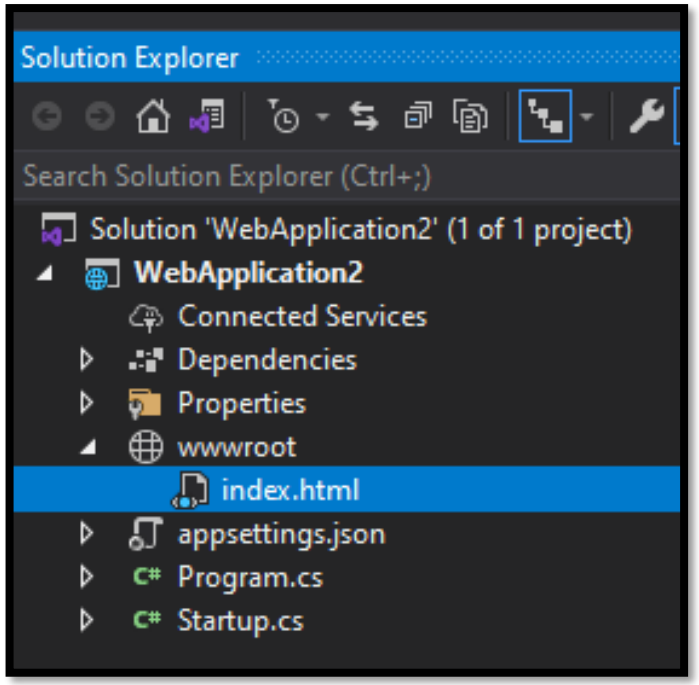
- See page 1391
- To serve a **default page** from **wwwroot** without a fully qualified URI:
  - Inside **Configure** method, call **UseDefaultFiles** before **UseStaticFiles**
  - Inside wwwroot place a file with either of the names: default.htm, default.html, index.htm, index.html

```
// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.UseDefaultFiles();
    app.UseStaticFiles();

    app.UseRouting();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapGet("/", async context =>
        {
            await context.Response.WriteAsync("Hello World!");
        });
    });
}
}
```

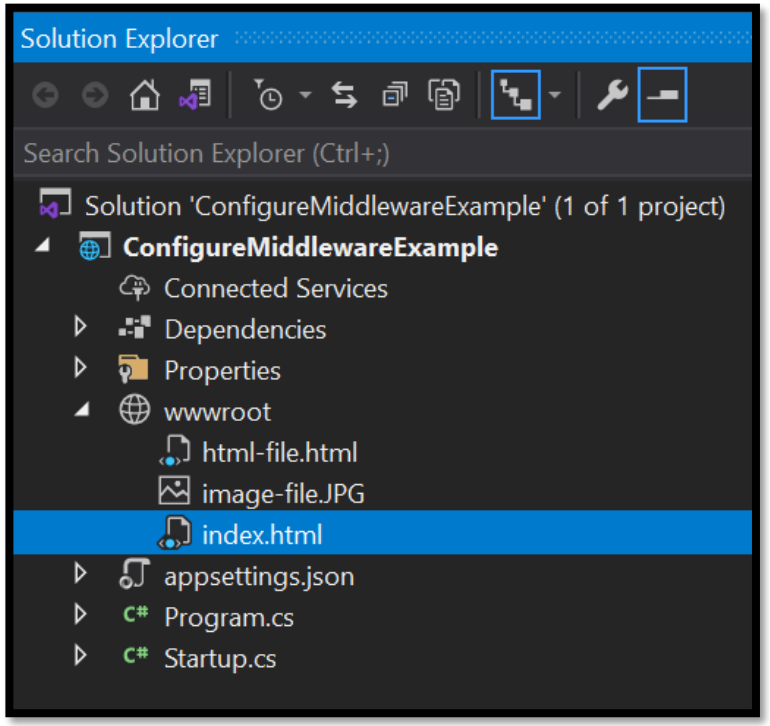


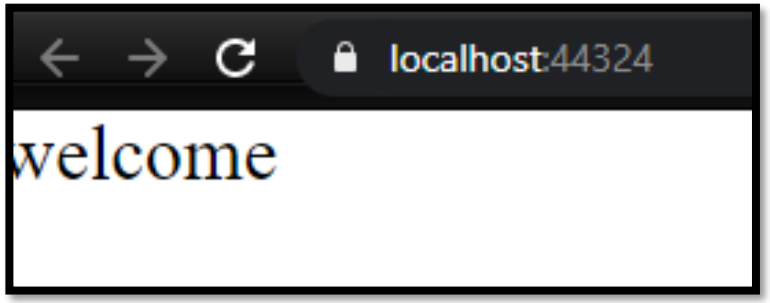
IN-CLASS DEMO: HOW TO WORK WITH STATIC FILES

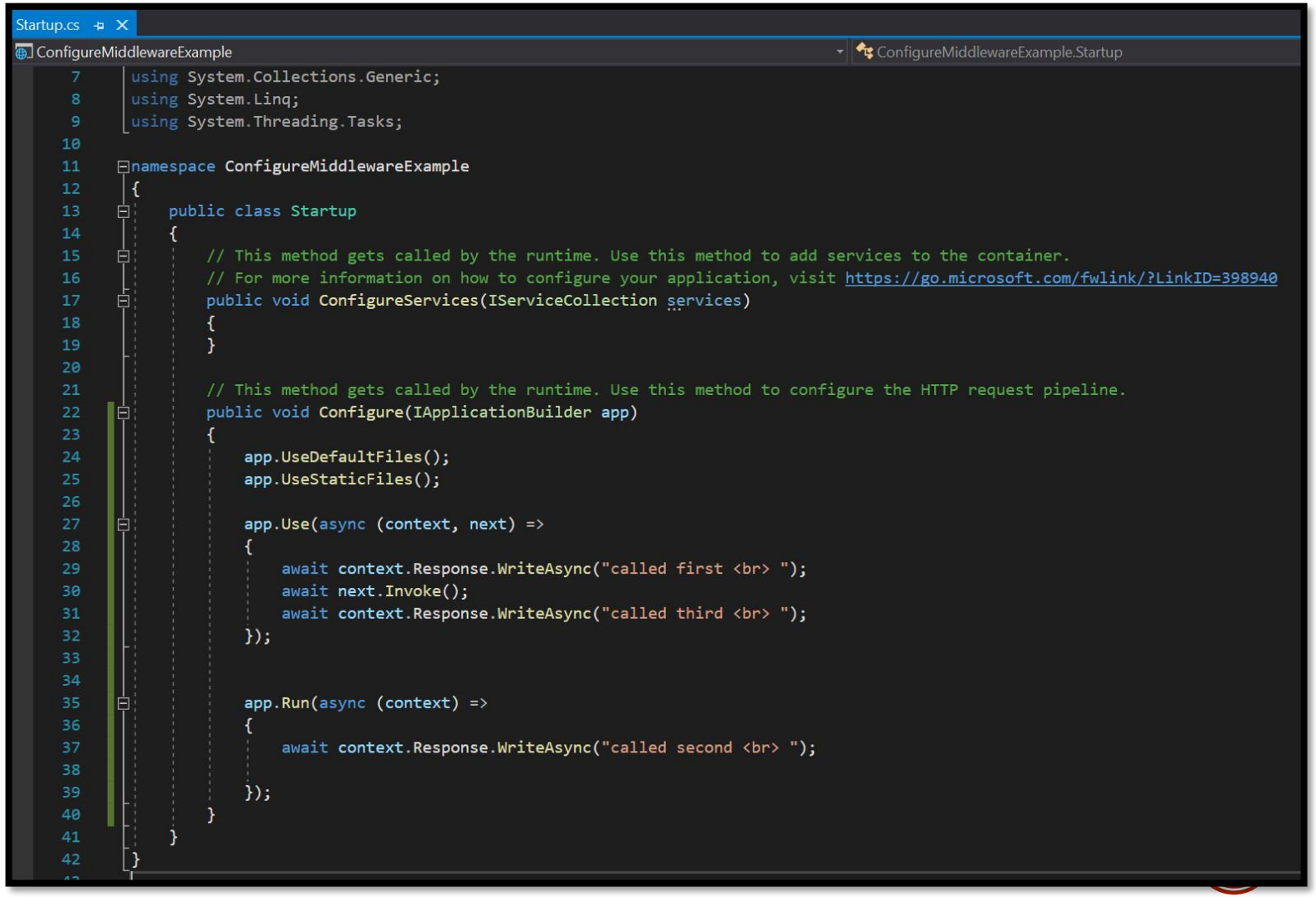
Demonstration: How to Work with Static Files

- Source/Steps
- https://github.com/MicrosoftLearning/20486D-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486D_MOD03_DEMO.md#demonstration-how-to-work-with-static-files

DEMO – CONTINUE EXISTING …








```
7   using System.Collections.Generic;
8   using System.Linq;
9   using System.Threading.Tasks;
10
11  namespace ConfigureMiddlewareExample
12  {
13      public class Startup
14      {
15          // This method gets called by the runtime. Use this method to add services to the container.
16          // For more information on how to configure your application, visit https://go.microsoft.com/fwlink/?LinkID=398940
17          public void ConfigureServices(IServiceCollection services)
18          {
19          }
20
21          // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
22          public void Configure(IApplicationBuilder app)
23          {
24              app.UseDefaultFiles();
25              app.UseStaticFiles();
26
27              app.Use(async (context, next) =>
28              {
29                  await context.Response.WriteAsync("called first <br> ");
30                  await next.Invoke();
31                  await context.Response.WriteAsync("called third <br> ");
32              });
33
34
35              app.Run(async (context) =>
36              {
37                  await context.Response.WriteAsync("called second <br> ");
38
39              });
40          }
41      }
42  }
```

SERVICES … AND DEPENDENCY INJECTION

- The **Startup** class is created by default with every web application and is used to:
  - Configure middleware (**Configure** method) ← seen in the previous slides
  - Configure services (**ConfigureServices** method) ← seen next

- **Services** are **classes** that can be **reused easily in multiple locations** without having to worry about instantiating them and their various dependencies and sub-dependencies.
  - This is facilitated via a technique known as **Dependency Injection (DI)**.
  - **DI** is a factory responsible for **creating an instance** of the dependency and **disposing** it when no longer needed.
  - As a result, large segments of code can easily be reused (in different **controllers**, **views**, the **Configure** method, etc.).
  - **Use** the **ConfigureServices** method to **register** services with the Dependency Injection.

SERVICES AND INJECTION

- In order to be able to inject services, you'll first need to:

- Create a service:
  - Any **class** that implements any **interface** can act as a **service**.

```
public interface IMyService
{
    string DoSomething();
}
public class MyService : IMyService
{
    public string DoSomething()
    {
        return "I am doing something";
    }
}
```

- Register a Service:
  - We use the **ConfigureServices** method
  - This adds the service to the **DI container**.
    - You will provide:
      - the **interface** that you want to declare and
      - the **type** of the class that you want to instantiate.

```
// This method gets called by the runtime. Use this method to add services to the container.
// For more information on how to configure your application, visit https://go.microsoft.com
public void ConfigureServices(IServiceCollection services)
{
    services.AddSingleton<IMyService, MyService>();
}
```

- Then you can inject the service where needed.
  - Notice how we did not need to create an instance of MyService!

```
// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
public void Configure(IApplicationBuilder app, IWebHostEnvironment env, IMyService myFirstService)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.UseDefaultFiles();
    app.UseStaticFiles();

    app.UseRouting();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapGet("/", async context =>
        {
            await context.Response.WriteAsync(myFirstService.DoSomething());
        });
    });
}
```

SERVICES AND INJECTION (2)

- This will make more sense later (and we'll revisit this).
- To **inject a service** into a **controller**, you need to follow the steps:

```
public interface IMyService
{
    string DoSomething();
}
public class MyService : IMyService
{
    public string DoSomething()
    {
        return "I am doing something";
    }
}
```

- Create a service …

- Register the service …
  - I.e. add the service to the **DI service container**

```
// This method gets called by the runtime. Use this method to add services to the container.
// For more information on how to configure your application, visit https://go.microsoft.com
public void ConfigureServices(IServiceCollection services)
{
    services.AddSingleton<IMyService, MyService>();
}
```

- Constructor Injection …
  - **Services** are added as a **constructor parameter**.
  - **Services** are typically defined using **interfaces**.
  - Then **services** can be used throughout the class.

```
public class HomeController : Controller
{
    private readonly ILogger<HomeController> _logger;
    private IMyService _myFirstService;

    public HomeController(ILogger<HomeController> logger, IMyService myFirstService)
    {
        _logger = logger;
        _myFirstService = myFirstService;
    }

    public IActionResult Index()
    {
        _myFirstService.DoSomething();
        return View();
    }
```

- See page 1817

SERVICE LIFETIME

- See also
  - https://www.c-sharpcorner.com/article/understanding-addtransient-vs-addscoped-vs-addsingleton-in-asp-net-core/
  - https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-5.0
  - https://stackoverflow.com/questions/38138100/addtransient-addscoped-and-addsingleton-services-differences

- AddSingleton – Instantiates once in the application's lifetime
  - Singleton objects are the same for every object and every request.

- AddScoped – Instantiates once per request made to the server
  - Scoped objects are the same within a request, but different across different requests.

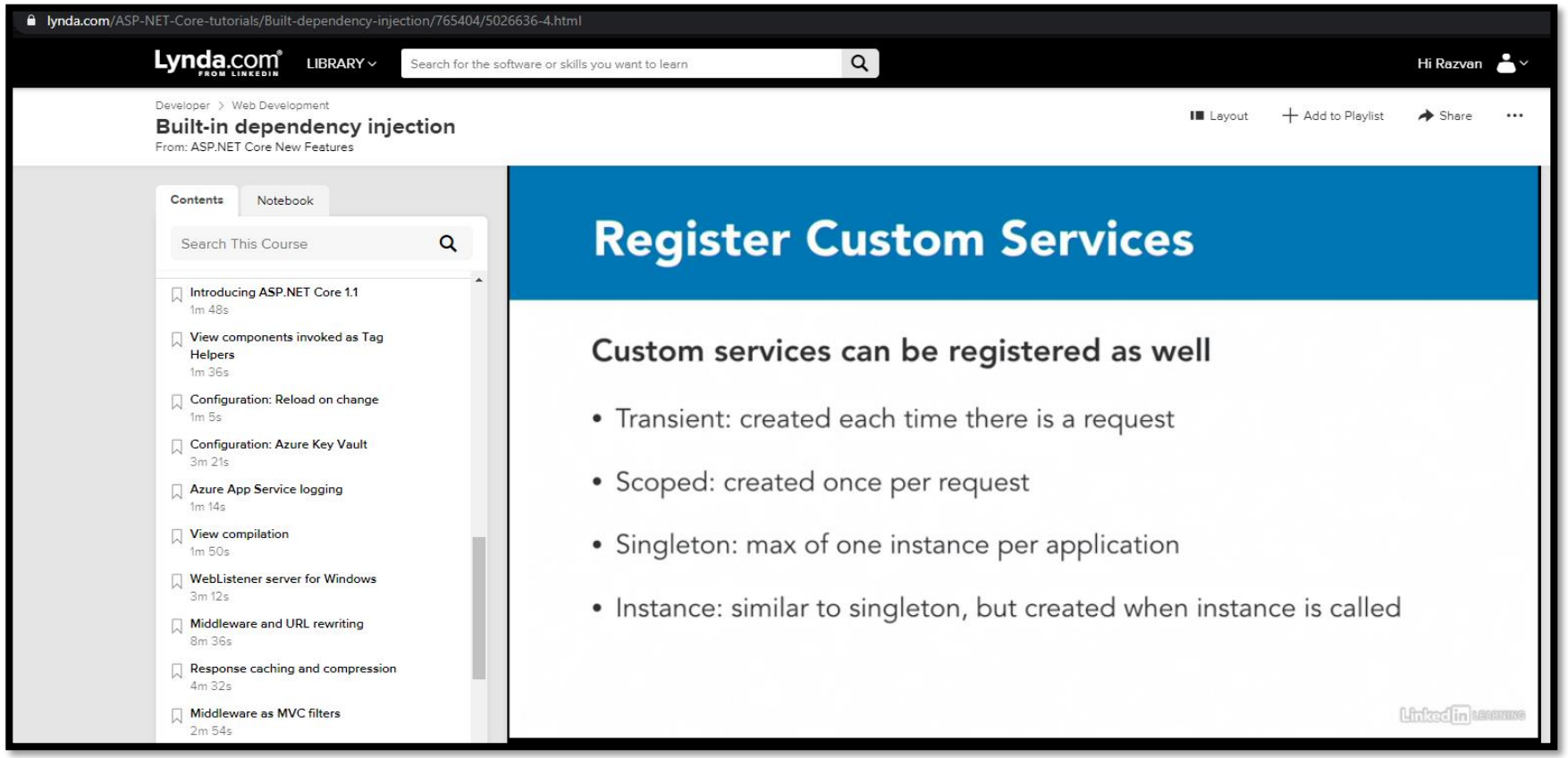
- AddTransient – Instantiates every single time the service is injected
  - Transient objects are always different; a new instance is provided to every controller and every service.

```
// This method gets called by the runtime. Use this method to add services to the container.
public void ConfigureServices(IServiceCollection services)
{
    services.AddSingleton<IMyService1, MyService1>();
    services.AddScoped<IMyService2, MyService2>();
    services.AddTransient<IMyService3, MyService3>();
}
```

SERVICE LIFETIME - EXTRA

- You may want to spend some time with Lynda/LinkedIn Learning courses.
  - City library gives you free access to Lynda: https://www.trl.org/reference-database-results?topics=120

- One such example:
  - https://www.lynda.com/ASP-NET-Core-tutorials/Built-dependency-injection/765404/5026636-4.html



IN-CLASS DEMO: HOW TO USE DEPENDENCY INJECTION

Demonstration: How to Use Dependency Injection

- Source/Steps
- https://github.com/MicrosoftLearning/20486D-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486D_MOD03_DEMO.md#demonstration-how-to-use-dependency-injection

LAB/HOMEWORK: EXPLORING ASP.NET CORE MVC

- **Module 03**
- Exercise 1: Working with Static Files
- Exercise 2: Creating Custom Middleware
- Exercise 3: Using Dependency Injection
- Exercise 4: Injecting a Service to a Controller

For the lab, when you are asked to use	**services.AddMvc();**
please use the following:	**services.AddMvc(x => x.EnableEndpointRouting = false);**

- You will find the **high-level** steps on the following page:
https://github.com/MicrosoftLearning/20486D-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486D_MOD03_LAB_MANUAL.md

- You will find the **detailed** steps on the following page:
https://github.com/MicrosoftLearning/20486D-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486D_MOD03_LAK.md

- For your homework submit one zipped folder with your complete solution.