Module 3

Configure Middleware and Services in ASP.NET Core

Module Overview

- Configuring Middleware
- Configuring Services

Lesson 1: Configuring Middleware

- Application Startup
- Middleware Fundamentals
- Demonstration: How to Create Custom Middleware
- Working with Static Files
- Demonstration: How to Work with Static Files

Application Startup

ConfigureServices

Used to set up all custom service in our application

Configure

Used to set up all middleware used in our application

ConfigureServices Method

ConfigureServices is used to set up services for our application

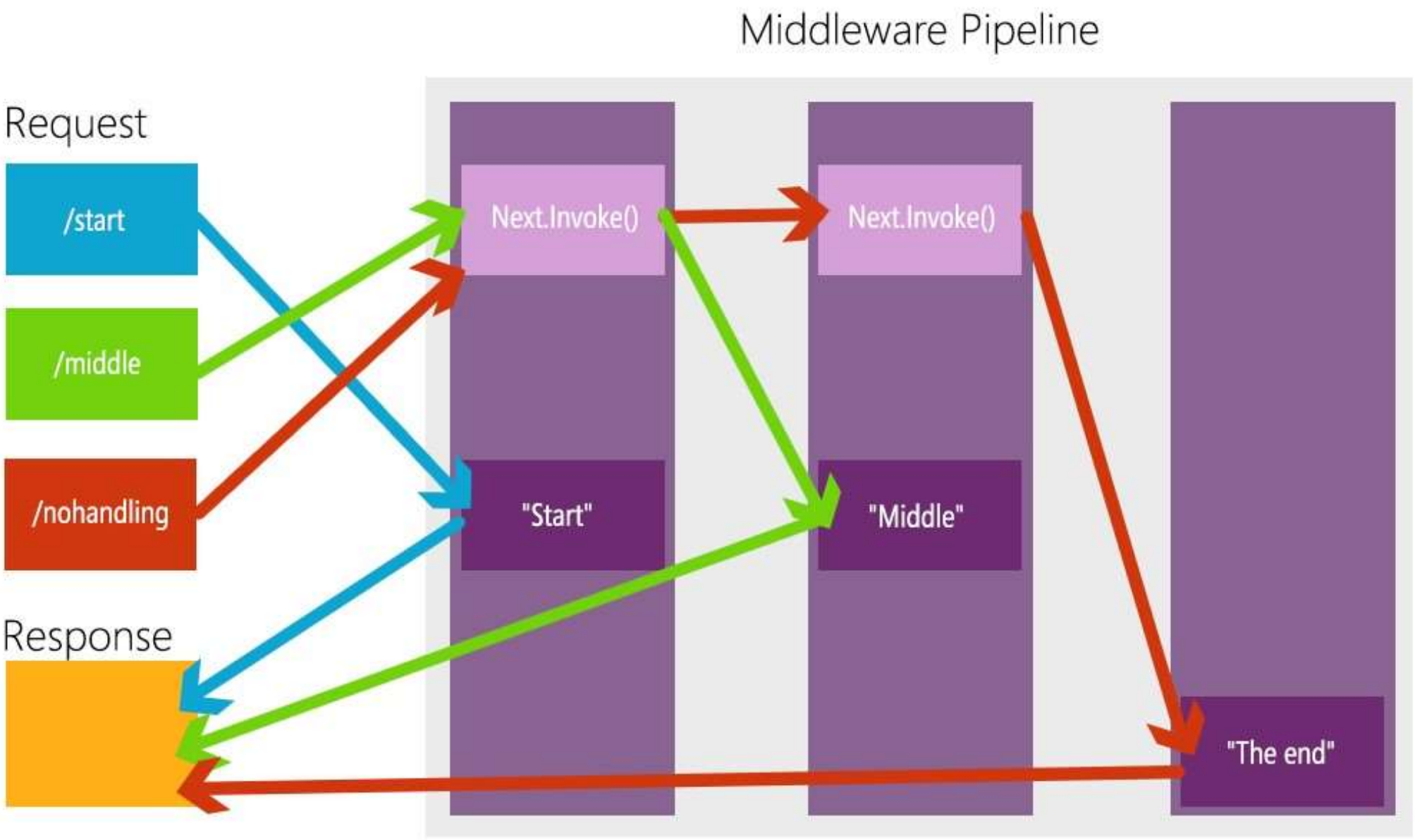
```
public void ConfigureServices(IServiceCollection services)
{
    services.AddSingleton<IService, Service>();
}
```

Configure Method

Configure is used to set up middleware for our application

```
public void Configure(IApplicationBuilder app)
{
    app.Run(async (context) =>
    {
        await context.Response.WriteAsync("Hello World!");
    });
}
```

Middleware Fundamentals



Run Middleware

- Always terminates the middleware pipeline
  - Middleware after it will never be run
- Should always provide a response for handling the final case

```
app.Run(async (context) =>
{
  await context.Response.WriteAsync("Inside run middleware");
});
```

Use Middleware

- Can call the next middleware in the chain by using the next parameter
- Can short circuit the pipeline chain by not calling for the next middleware
- The middleware most frequently used in the pipeline

```
app.Use(async (context, next) =>
{
    await context.Response.WriteAsync("Inside use middleware");
    await next.Invoke();
});
```

MapUsing the Startup Class to Configure Services Middleware

- Allows us to create alternative behavior for specific paths
- Does not continue the main path
- Can be nested
- Does not do anything when used on its own
- Not frequently used

```
app.Map("/Map", (map) =>
{
    map.Run(async (context) =>
    {
        await context.Response.WriteAsync("Run middleware inside of
                map middleware");
    });
});
```

Demonstration: How to Create Custom Middleware

In this demonstration, you will learn:

- How to add custom middleware to the application
- How to use the middleware's context parameter
- How to use the Invoke method on the "next" parameter
- How the order of middleware affects their execution

Working with Static Files

- Static files are files that do not change at run time
- In ASP.NET Core applications static files can be served to clients by using the **UseStaticFiles** middleware

```
public void Configure(IApplicationBuilder app)
{
    app.UseStaticFiles();
}
```

Common types of static files

Common types of static files:

- HTML files
- CSS stylesheets
- JavaScript files
- Images and other assets
- JSON or XML files

Demonstration: How to Work with Static Files

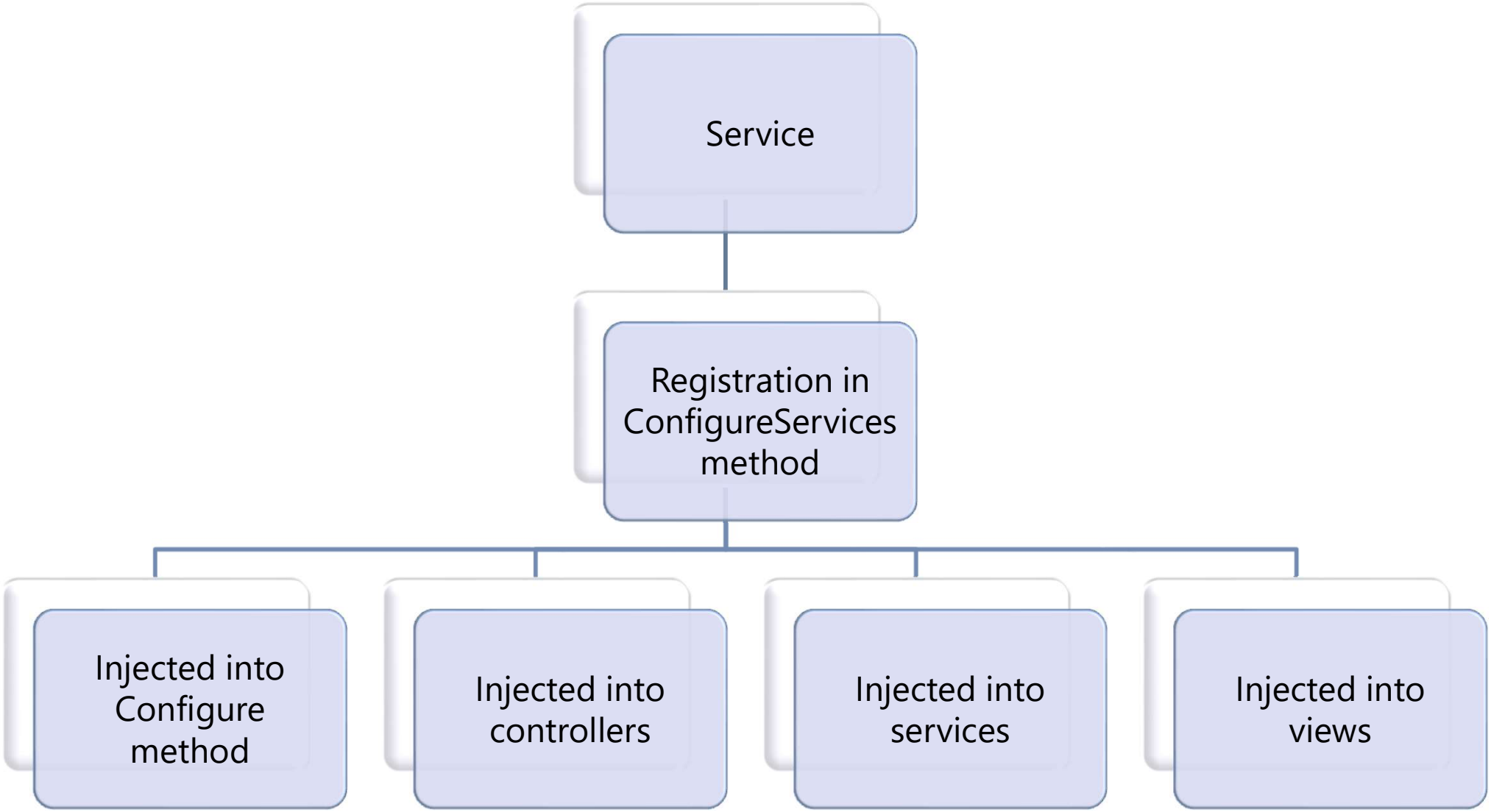
In this demonstration, you will learn:

- How to add HTML and image files as static files in the **wwwroot** folder
- How to serve static files by using the **UseStaticFiles** middleware
- What is the result of attempting to access a file that does not exist

Lesson 2: Configuring Services

- Introduction to Dependency Injection
- Using the Startup Class to Configure Services
- Demonstration: How to Use Dependency Injection
- Inject Services to Controllers
- Service Lifetime

Introduction to Dependency Injection



Using the Startup Class to Configure Services

- Any class can act as a service
- By using the **ConfigureServices** method, you can register any service you would like to use in the application
- By using Dependency Injection in the **Configure** method you can utilize the services inside the middleware

Service Configuration and Injection

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddSingleton<IService, Service>();
}

public void Configure(IApplicationBuilder app, IService service)
{
    app.Run(async (context) =>
    {
        await service.DoSomething();
    });
}
```

Demonstration: How to Use Dependency Injection

In this demonstration, you will learn how to:

- Create a service in ASP.NET Core
- Register a service in the **ConfigureServices** method
- Inject a service into the **Configure** method by using Dependency Injection

Inject Services to Controllers

- A controller is used to handle requests from the client
- Controllers support constructor dependency injection
- If the internal behavior of a service or its dependencies change, you will not need to update the controller
- You cannot have more then one constructor in the controller, as the default dependency injection container cannot handle it

Injecting a Service to a Controller

```
public class HomeController : Controller
{
    private IMyService _myService;

    public HomeController(IMyService myService)
    {
        _myService = myService;
    }

    public IActionResult Index()
    {
        return Content(_myService.DoSomething());
    }
}
```

Service Lifetime

- AddSingleton – Instantiates once in the application's lifetime
- AddScoped – Instantiates once per request made to the server
- AddTransient – Instantiates every single time the service is injected

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddSingleton<IFirstService, FirstService>();
    services.AddScoped<ISecondService, SecondService>();
    services.AddTransient<IThirdService, ThirdService>();
}
```

Lab: Configuring Middleware and Services in ASP.NET Core

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

Estimated Time: 75 Minutes

Lab Scenario

The Adventure Works company wants to develop a website about ball games. For this, the company needs to perform a survey to determine the popularity of different ball games. As their employee, you are required to create a survey site.

Lab Review

- What is the difference between app.Use and app.Run in the Configure method in the Startup class?
- What will change when you update the service configuration in the ConfigureServices method from AddSingleton to AddScoped, or to AddTransient?
- What happens to the UseStaticFiles middleware when the browser is directed to a path where no static file is found?

Module Review and Takeaways

- Review Question
- Best Practice
- Common Issues and Troubleshooting Tips