Setting up an ASP.NET Core MVC Application

Gill Cleeren

CTO Xebia Microsoft Services Belgium

@gillcleeren

Overview

Creating a new project

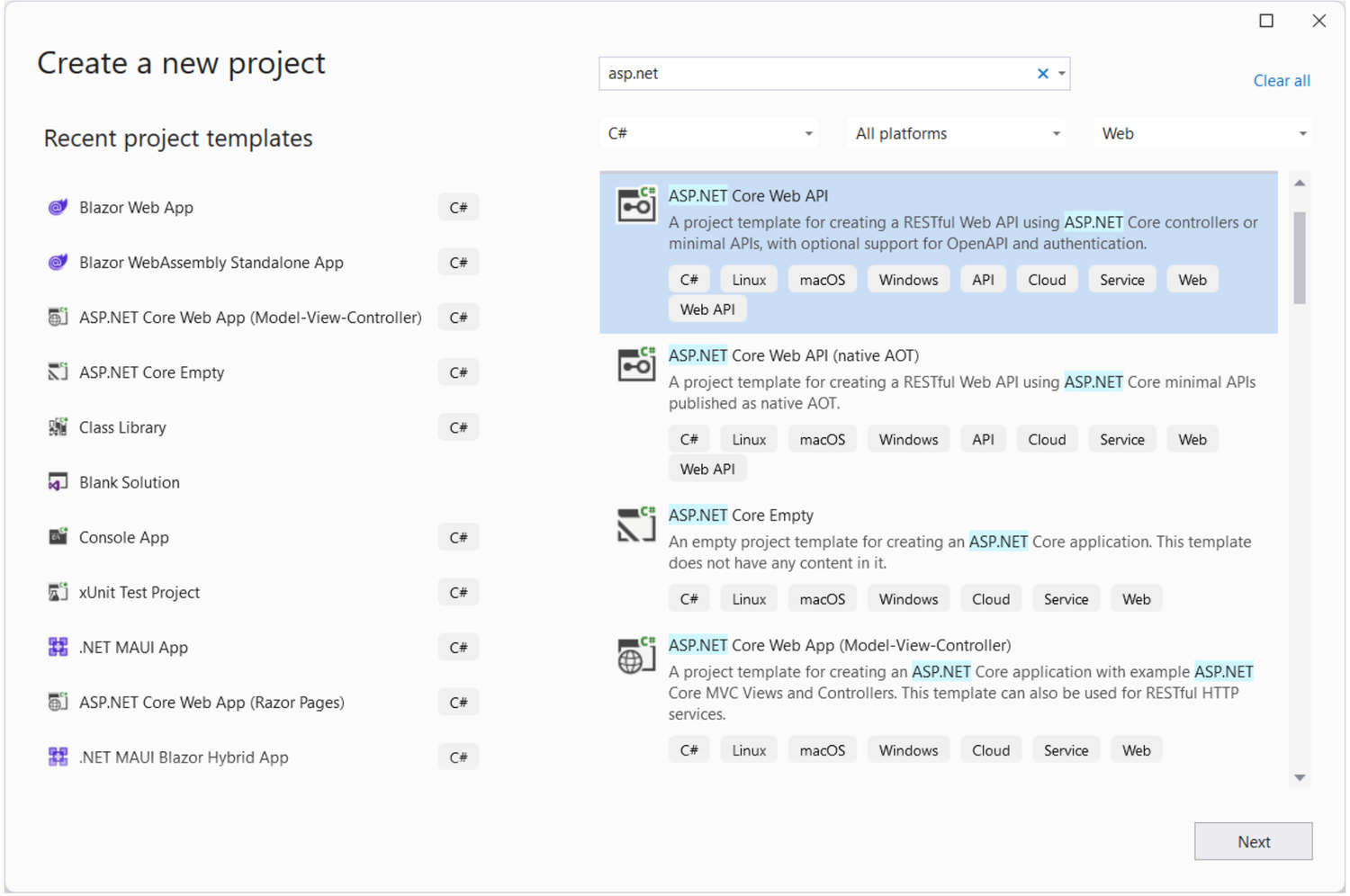
Exploring the generated files

Configuring the site

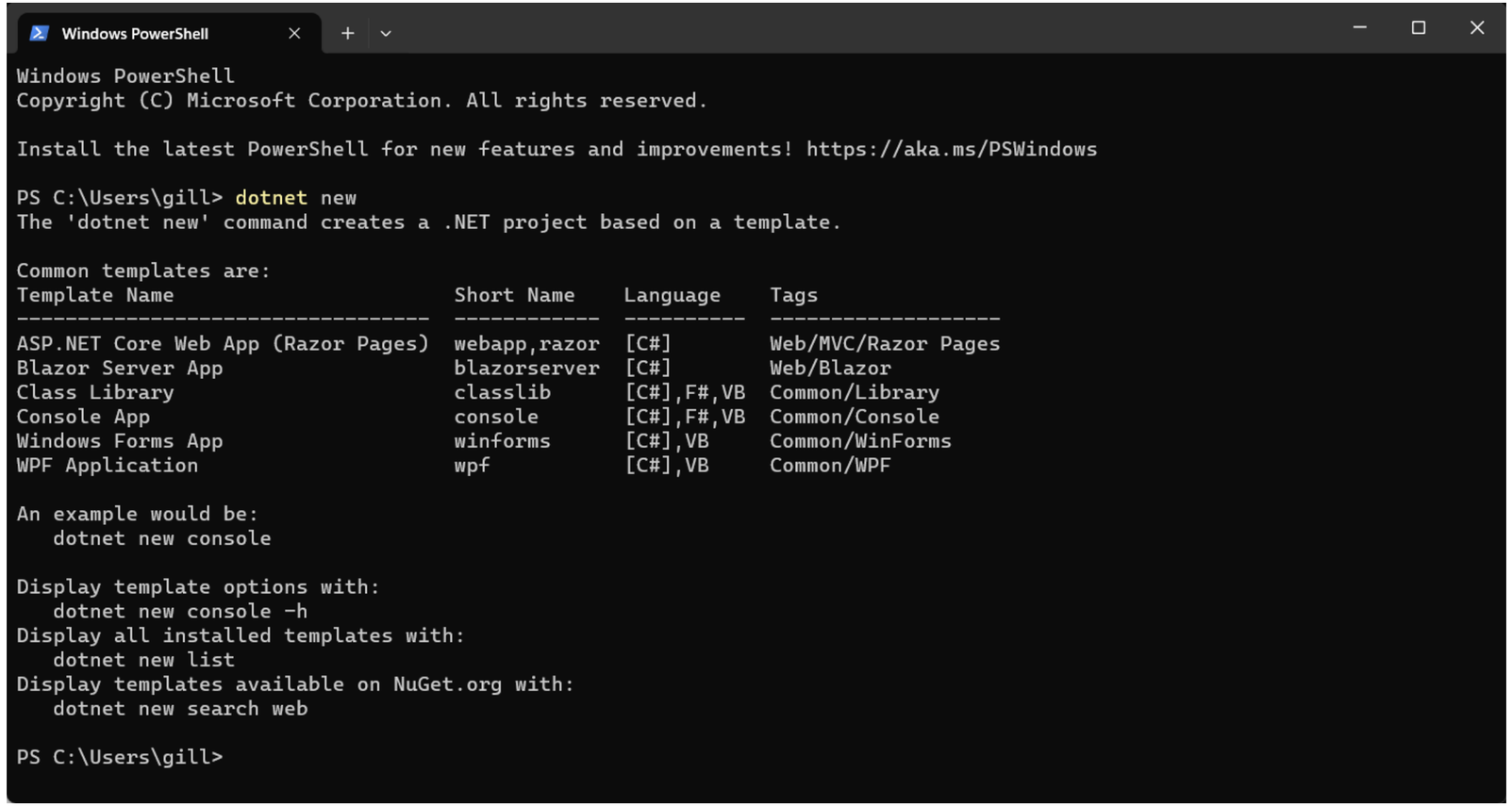
How ASP.NET Core handles a request

Creating a New Project

Templates


Create a new project
asp.net
Clear all
Recent project templates
C#
All platforms
Web
Blazor Web App
C#
Blazor WebAssembly Standalone App
C#
ASP.NET Core Web App (Model-View-Controller)
C#
ASP.NET Core Empty
C#
Class Library
C#
Blank Solution
Console App
C#
xUnit Test Project
C#
.NET MAUI App
C#
ASP.NET Core Web App (Razor Pages)
C#
.NET MAUI Blazor Hybrid App
C#
ASP.NET Core Web API
A project template for creating a RESTful Web API using ASP.NET Core controllers or minimal APIs, with optional support for OpenAPI and authentication.
C# Linux macOS Windows API Cloud Service Web Web API
ASP.NET Core Web API (native AOT)
A project template for creating a RESTful Web API using ASP.NET Core minimal APIs published as native AOT.
C# Linux macOS Windows API Cloud Service Web Web API
ASP.NET Core Empty
An empty project template for creating an ASP.NET Core application. This template does not have any content in it.
C# Linux macOS Windows Cloud Service Web
ASP.NET Core Web App (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.
C# Linux macOS Windows Cloud Service Web
Next

Templates in the .NET CLI



Demo

Creating a new project using a template

Building and running the application

Demo

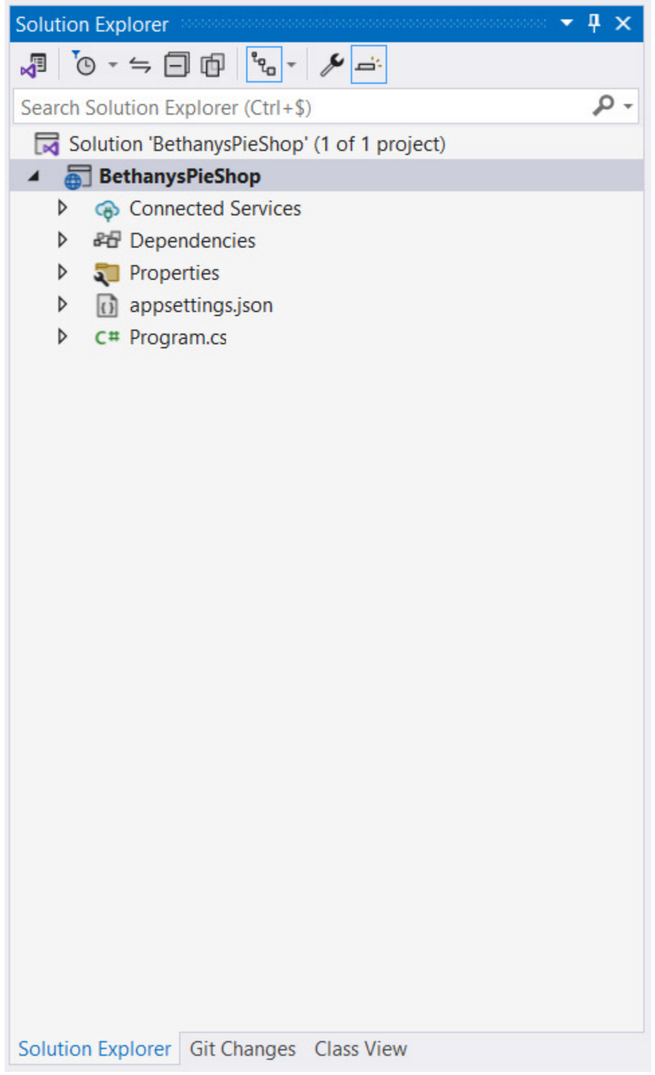
Using the CLI to create a new web application

Demo

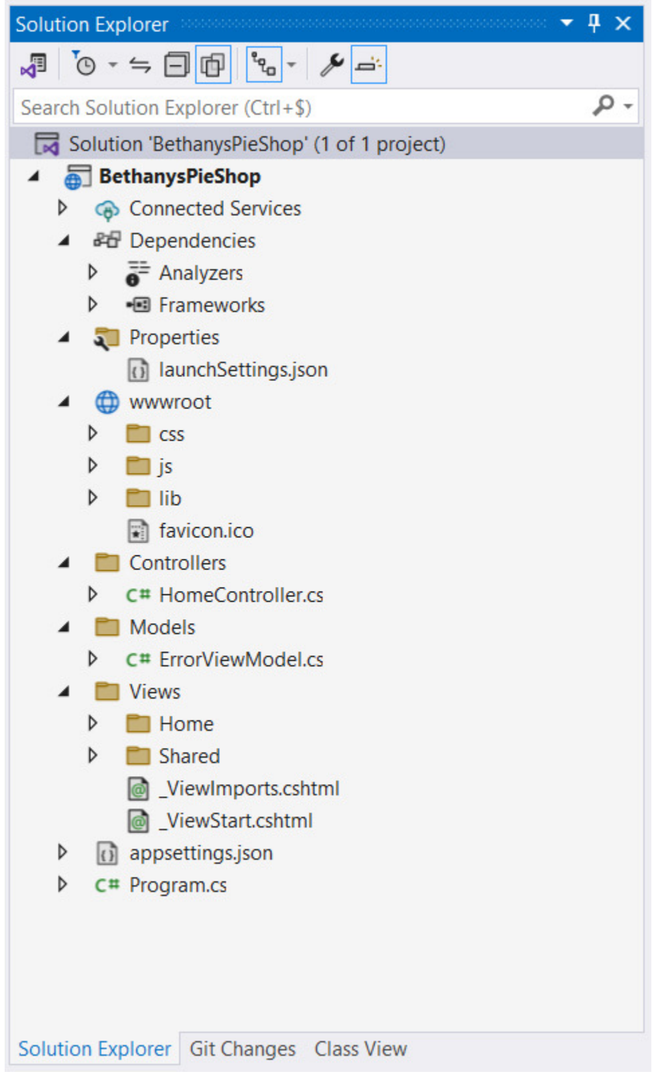
Using VS Code and the C# Dev Kit to create a new ASP.NET Core application

Exploring a New Project

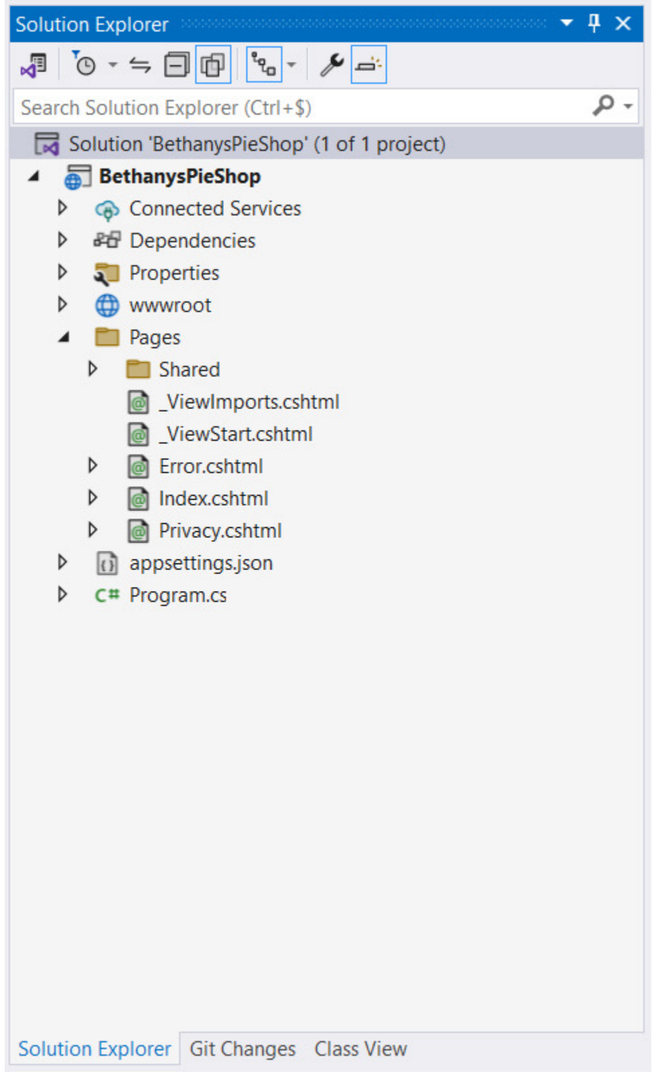
Project Structure (Empty Web Application)



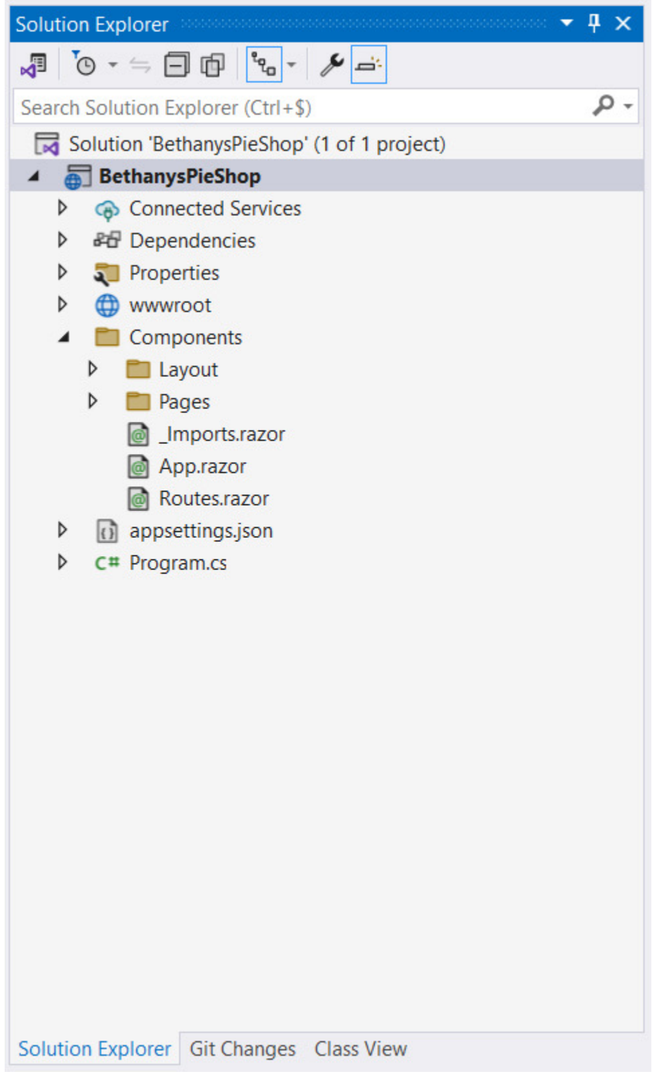
Project Structure (Web Application MVC)



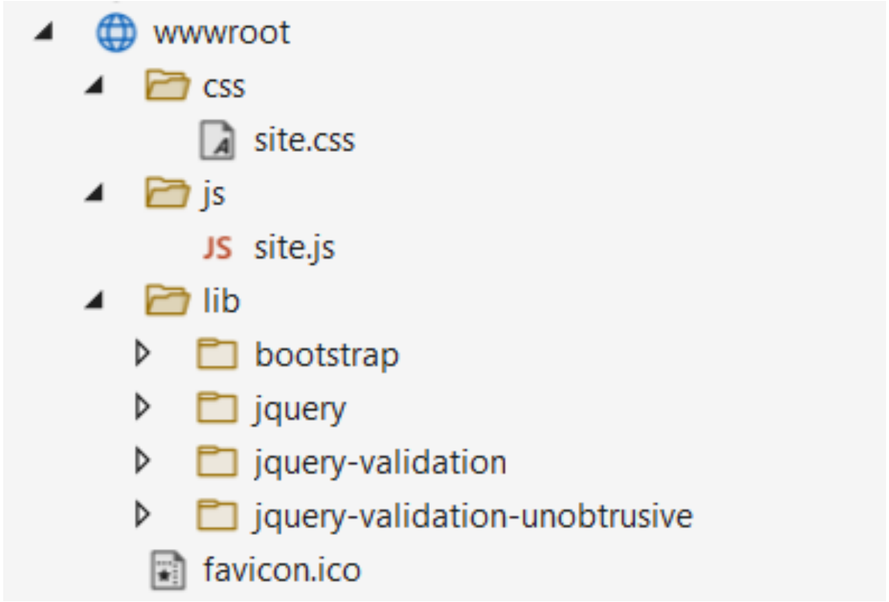
Project Structure (Razor Pages)



Project Structure (Blazor Server-side Rendering)



The wwwroot Folder



wwwroot/image1.jpg

http://bethanyspieshop.com/image1.jpg

The csproj File

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>net8.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
  </PropertyGroup>

</Project>
```

Adding Dependencies

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>net8.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Newtonsoft.Json" Version="13.0.3" />
  </ItemGroup>

</Project>
```

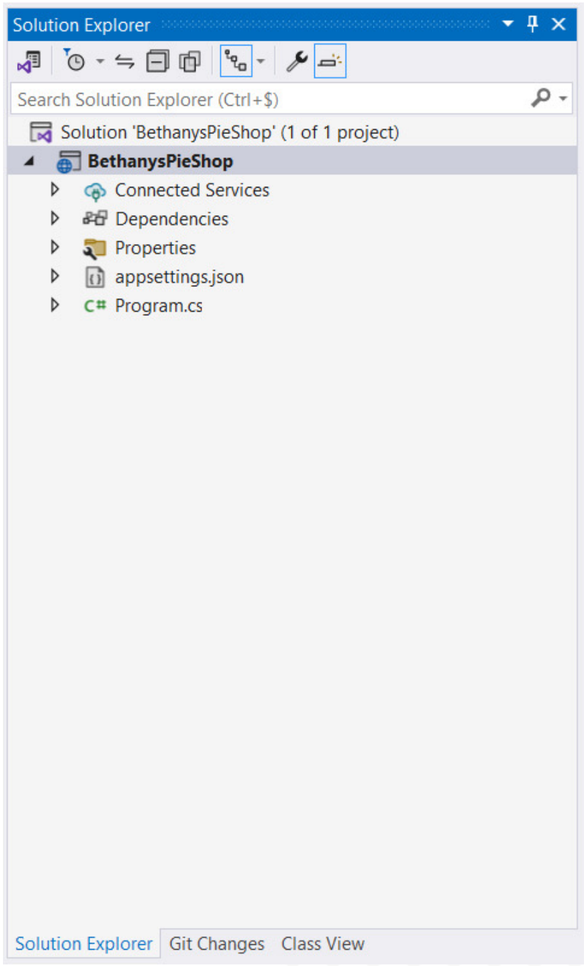
Demo

Exploring the generated files

Looking at launchSettings.json

Configuring the Site

The Program Class



ASP.NET Core applications start like console applications

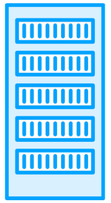
- static void main
- “Replaced” since .NET 6 & C# 10 with top-level statements

Contains logic to start the server and listen for requests as well as configuration of the application

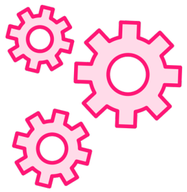
```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.MapGet("/", () => "Hello World!");

app.Run();
```

The Default Program.cs

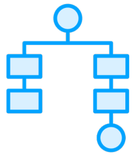
The CreateBuilder Method



Set up Kestrel server



Configure IIS integration



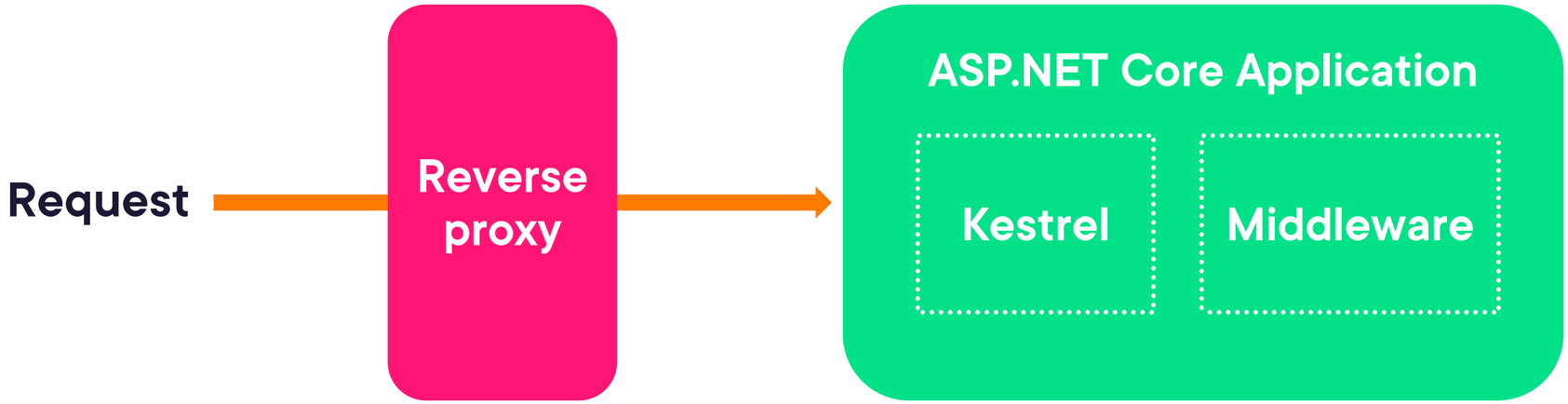
Specify content root

Read application settings

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.MapGet("/", () => "Hello World!");

app.Run();
```

The Default Program.cs

Sidestep: Running a Server with Kestrel



Configuration of the Application

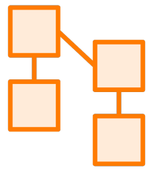
Service registration

Middleware

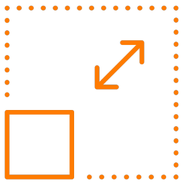
A More Extended Program.cs

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    // The default HSTS value is 30 days. You may want to change this for production scenarios, see
https://aka.ms/aspnetcore-hsts.
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

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

app.Run();
```

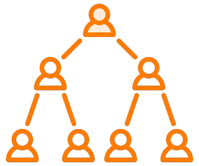
Service Registration



All classes used by the application



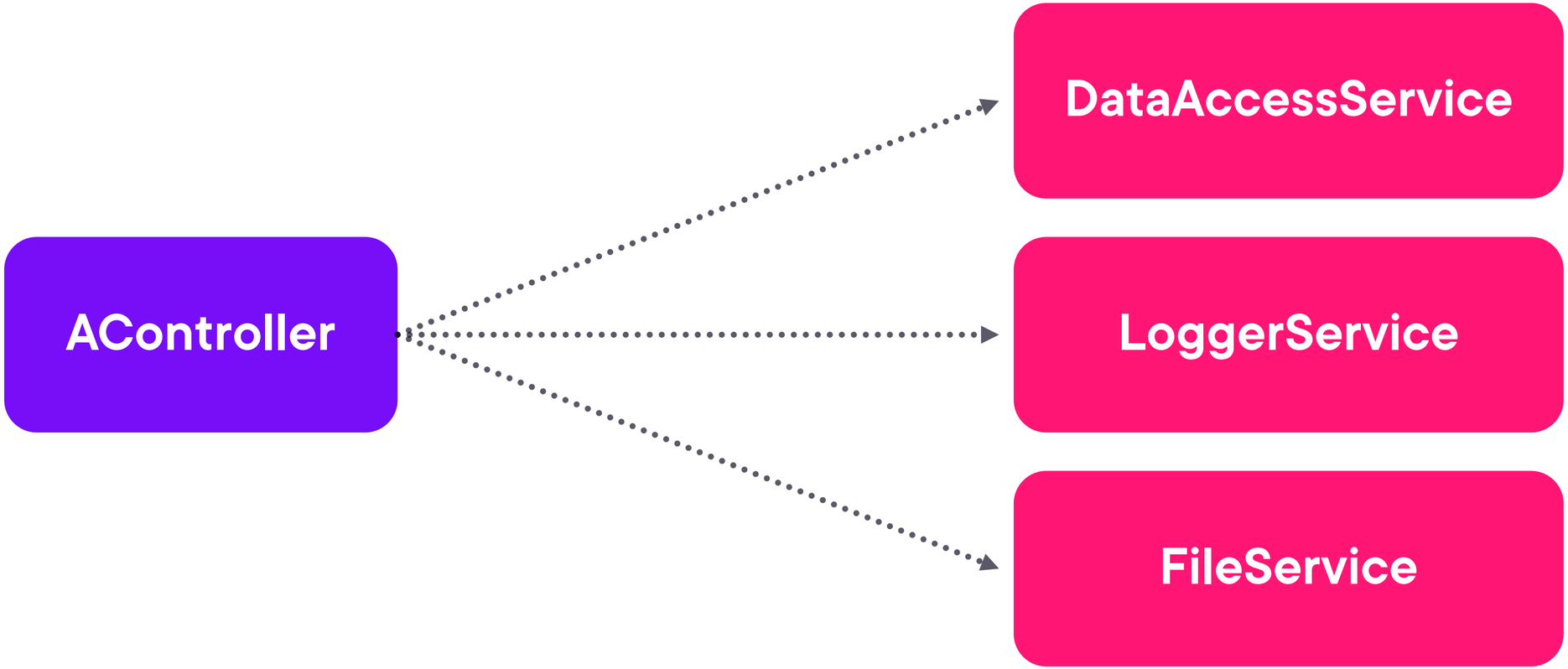
More modular approach



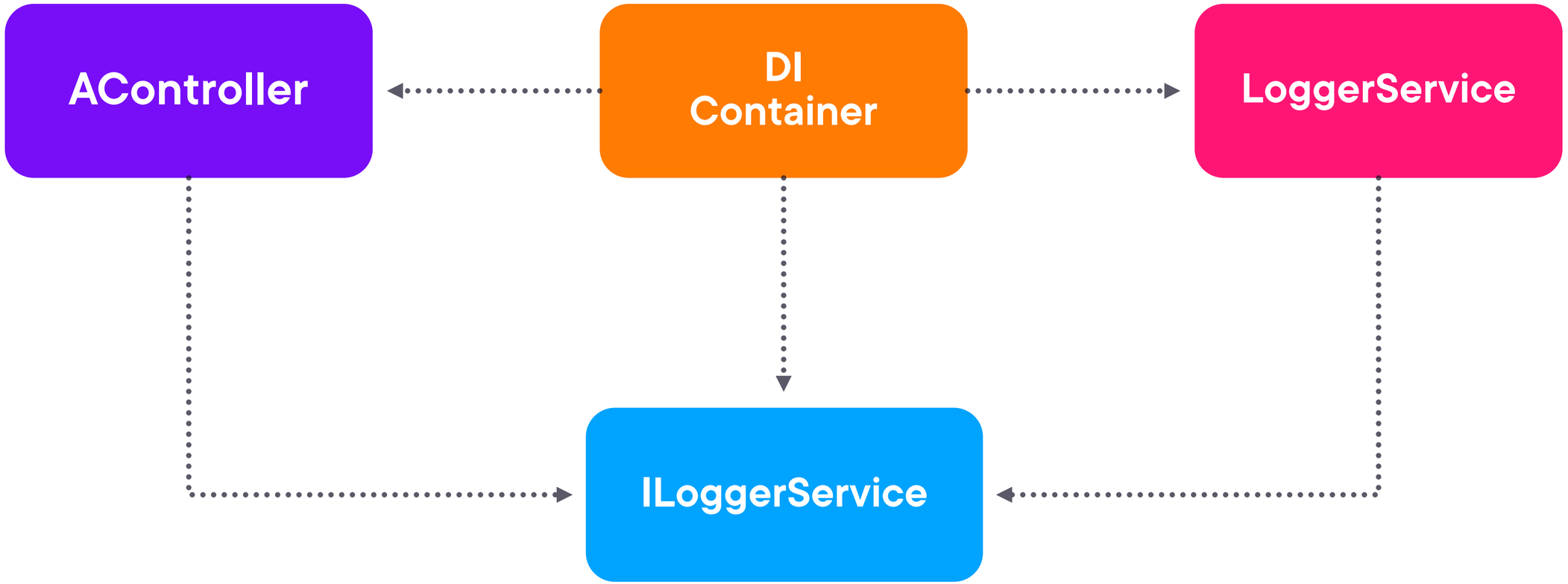
More dependencies will need to be injected

Using Services

Initializing Dependencies

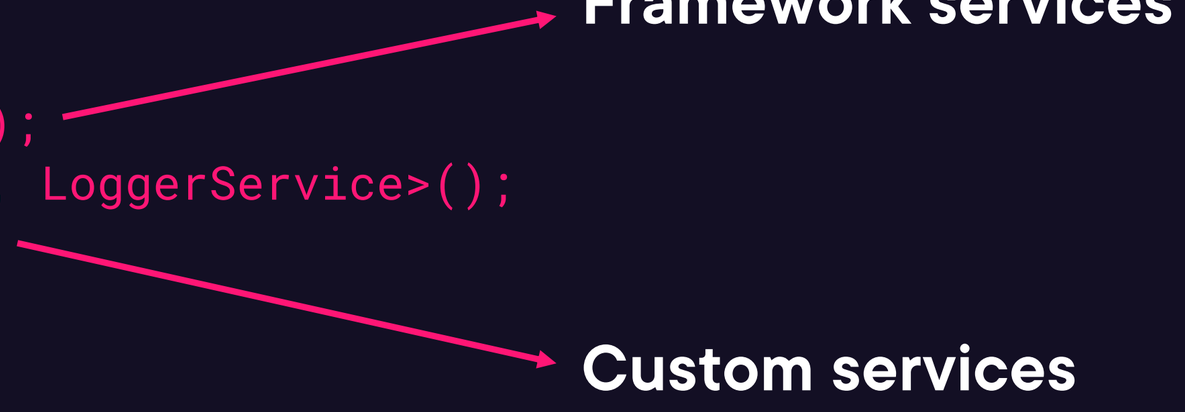


Introducing Dependency Injection (DI)



Registering Services

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();
builder.Services.AddScoped<ILoggerService, LoggerService>();

var app = builder.Build();

...

app.Run();
```

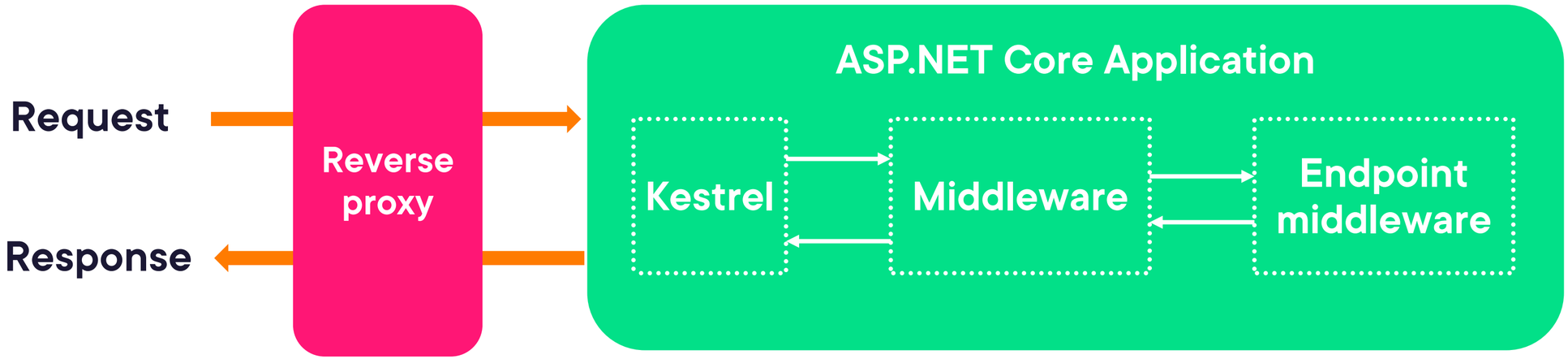


```
public class OrderController : Controller
{
    private readonly ILoggerService _loggerService;

    public OrderController(ILoggerService loggerService)
    {
        _loggerService = loggerService;
    }
}
```

Using Services

Injected via constructor

Handling Requests with Middleware

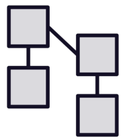


Middleware will create response based on incoming request

The Middleware Request Pipeline

Pipeline consists out of set of components

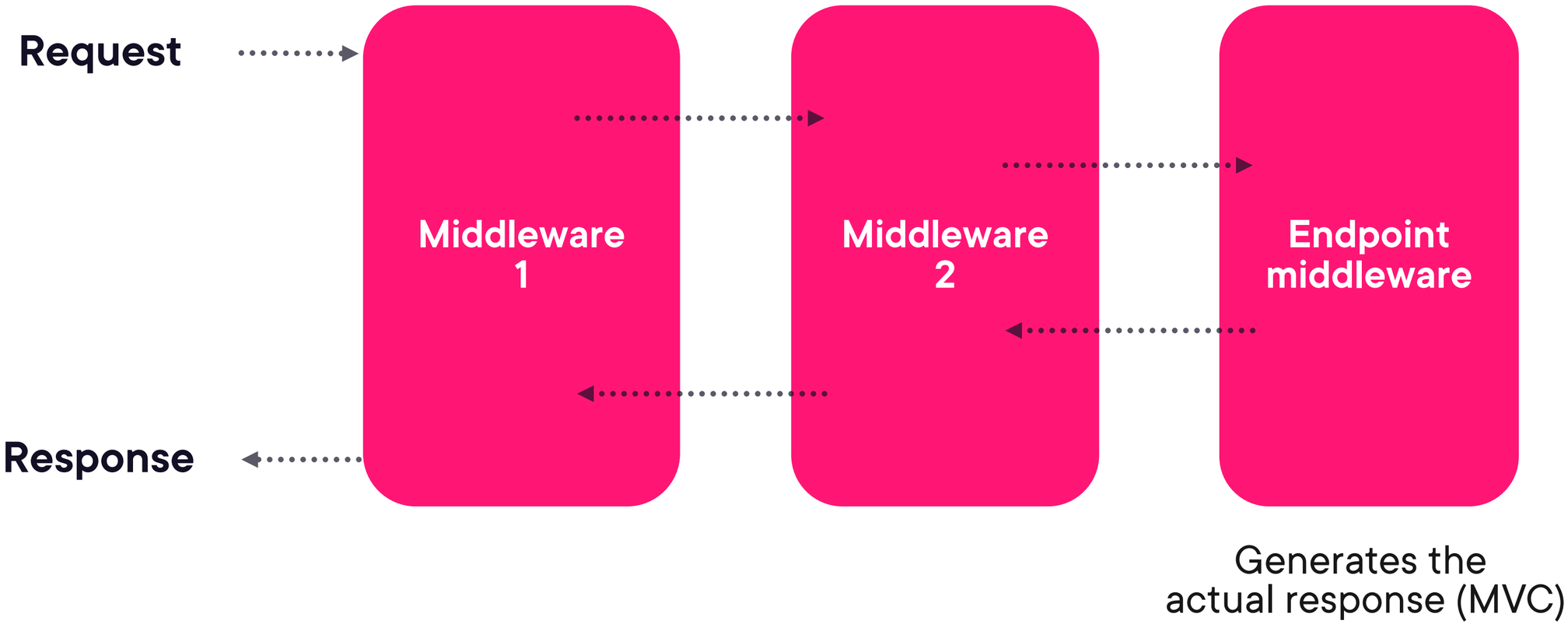
Components work on request or response



Many built-in components

Endpoint middleware sits at the end

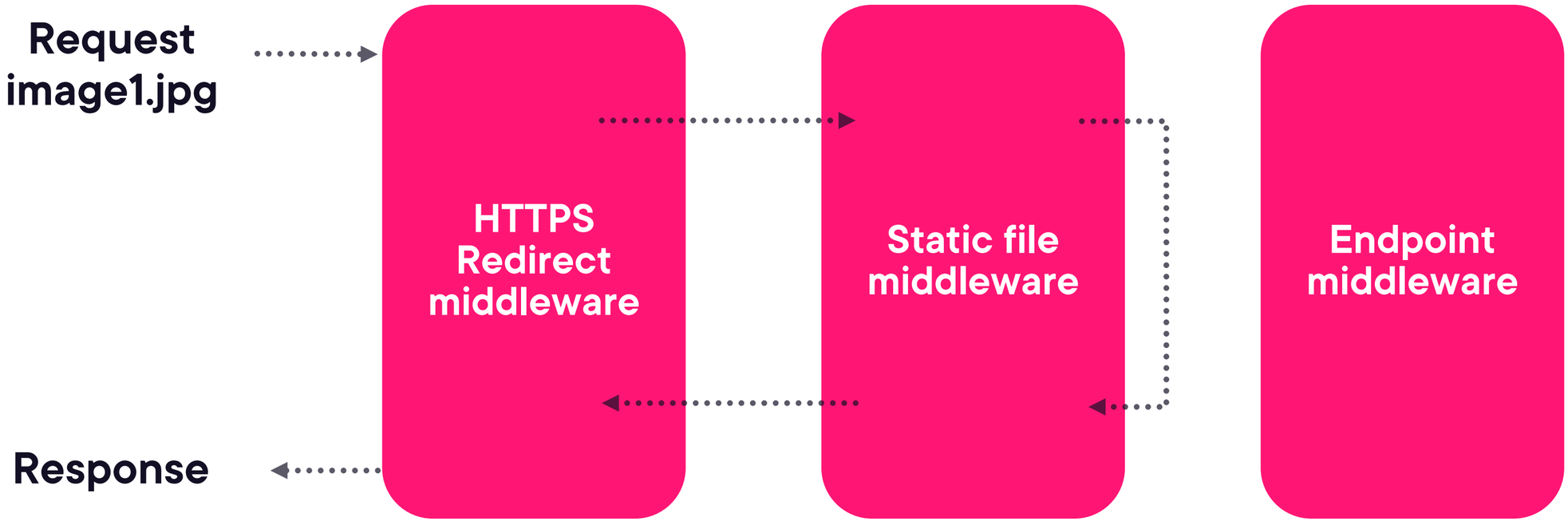
The Middleware Request Pipeline



Middleware Request Pipeline

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    // The default HSTS value is 30 days. You may want to change this for production scenarios, see
https://aka.ms/aspnetcore-hsts.
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

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

app.Run();
```

Exploring Static File Middleware



The order we add the components in, will be the order of the components in the pipeline!

Program.cs

Service registration

Middleware

What About the Old Model?

Program.cs

```
public class Program
{
  public static void Main(string[] args)
  {
    CreateHostBuilder(args).Build().Run();
  }

  public static IHostBuilder
    CreateHostBuilder(string[] args) =>
      Host.CreateDefaultBuilder(args)
      .ConfigureWebHostDefaults(webBuilder =>
      {
        webBuilder.UseStartup<Startup>();
      });
}
```

Startup.cs

```
public class Startup
{
  public void
ConfigureServices(IServiceCollection
    services)
  {
    services.AddControllersWithViews();
  }

  public void Configure(IApplicationBuilder app,
    IWebHostEnvironment env)
  {
    ...
  }
}
```

Demo

Configuring the application

Summary

- Program class starts up application
- ASP.NET Core comes with built-in web server (Kestrel)
- Dependency injection is used by default
- Handling requests is done through middleware

Up Next:

Creating your first page