

- Rain makes the grass and leaves look **greener!**
 - Why?



DotNetCore Pre MVC

Prepared for Vth semester DDU-CE students
2022-23 WAD

Apurva A Mehta

asp net core mvc - Google Search

google.com/search?q=asp+net+core+mvc&oq=asp+net+core+mvc&aqs=chrome..69i57j0l4j69i60l2j69i61.6823j0j7&source=

Apps IP WDDN Introduction - YouT...

Google

asp net core mvc

All

Videos

Images

News

Shopping

More

Settings

Tools

About 3,09,00,000 results (0.52 seconds)

docs.microsoft.com > en-us > core > mvc > overview

Overview of ASP.NET Core MVC | Microsoft Docs

Feb 12, 2020 - The **ASP.NET Core MVC** framework is a lightweight, open source, highly testable presentation framework optimized for use with **ASP.NET Core**. **ASP.NET Core MVC** provides a patterns-based way to build dynamic websites that enables a clean separation of concerns.

Videos

ASP.NET CORE TUTORIAL

FACEBOOK.COM/PRAGMITECH | TWITTER.COM/KUDVENKAT

7:50

7:50

PART 15 - ASP.NET CORE MVC TUTORIAL

ASP NET Core MVC tutorial

kudvenkat

YouTube - Feb 9, 2019

INTRO TO ASP.NET CORE MVC

1:03:56

1:03:56

INTRO TO ASP.NET CORE MVC

TIM COREY

Introduction to ASP.NET Core MVC in C# plus LOTS of Tips

IAmTimCorey

YouTube - Jun 8, 2020

.NET Core

3:13:18

3:13:18

ASP.NET Core in 3 hours

NO ADS

Learn ASP.NET Core 3.1 - Full Course for Beginners [Tutorial]

freeCodeCamp.org

YouTube - Feb 5, 2020

asp net core mvc - YouTube

x

+

←

→

↺

youtube.com/results?search_query=asp+net+core+mvc

Apps

IP

WDDN

Introduction - YouT...

☰

YouTube^{IN}

asp net core mvc

🔍

🔖

🏠

Home

🔥

Trending

📺

Subscriptions

📖

Library

ASP.NET CORE TUTORIAL

FACEBOOK.COM/PRAGIMTECH | TWITTER/PRAGIMTECH

124

📺

PART 1 - ASP.NET CORE TUTORIAL INTRODUCTION

ASP.NET core tutorial for beginners

kudvenkat ✓

ASP NET Core Tutorial • 5:35

Setting up machine for asp net core development • 5:28

VIEW FULL PLAYLIST

Version

ASP.NET Core 3.1 ▾

Filter by title

ASP.NET Core documentation

What's new in ASP.NET Core docs

> Overview

Get started

> Release notes

> Tutorials

> Fundamentals

> Web apps

> Razor Pages

> MVC

Overview

> Tutorial

Views

Partial views

Controllers

Routing

Dependency injection - controllers

Dependency injection - views

Unit test

Overview of ASP.NET Core MVC

02/12/2020 • 9 minutes to read • 10

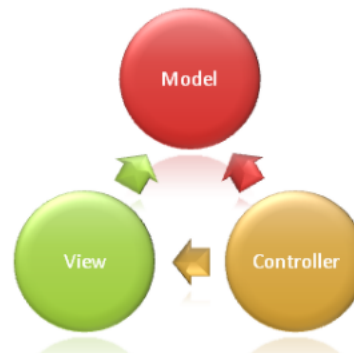
By [Steve Smith](#)

ASP.NET Core MVC is a rich framework for building web apps and APIs using the Model-View-Controller design pattern.

What is the MVC pattern?

The Model-View-Controller (MVC) architectural pattern separates an application into three main groups of components: Models, Views, and Controllers. This pattern helps to achieve [separation of concerns](#). Using this pattern, user requests are routed to the Controller which is responsible for working with the Model to perform user actions and/or retrieve results of queries. The Controller chooses the View to display to the user, and provides it with any Model data it requires.

The following diagram shows the three main components and which ones reference the others:



ASP.NET Core Web Application

Create a new project

Recent project templates

 Console App (.NET Framework)

C#

Web



Clear all

All languages

All platforms

All project types



ASP.NET Core Web Application

Project templates for creating ASP.NET Core web apps and web APIs for Windows, Linux and macOS using .NET Core or .NET Framework. Create web apps with Razor Pages, MVC, or Single Page Apps (SPA) using Angular, React, or React + Redux.

Cloud

C#

Linux

macOS

Service

Web

Windows



ASP.NET Web Application (.NET Framework)

Project templates for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Cloud

Visual Basic

Web

Windows



ASP.NET Web Application (.NET Framework)

Project templates for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Cloud

C#

Web

Windows



ASP.NET Core Web Application

Project templates for creating ASP.NET Core web apps and web APIs for Windows, Linux and macOS using .NET Core or .NET Framework. Create web apps with Razor Pages, MVC, or Single Page Apps (SPA) using Angular, React, or React + Redux.

F#

Linux

macOS

Web

Windows

Back

Next

Configure your new project

ASP.NET Core Web Application

Cloud

C#

Linux

macOS

Service

Web

Windows


Project name

StaffManagement

Location

C:\Users\AAM\source\repos

...

Solution name 

StaffManagement

☐

Place solution and project in the same directory

Back

Create

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](#))

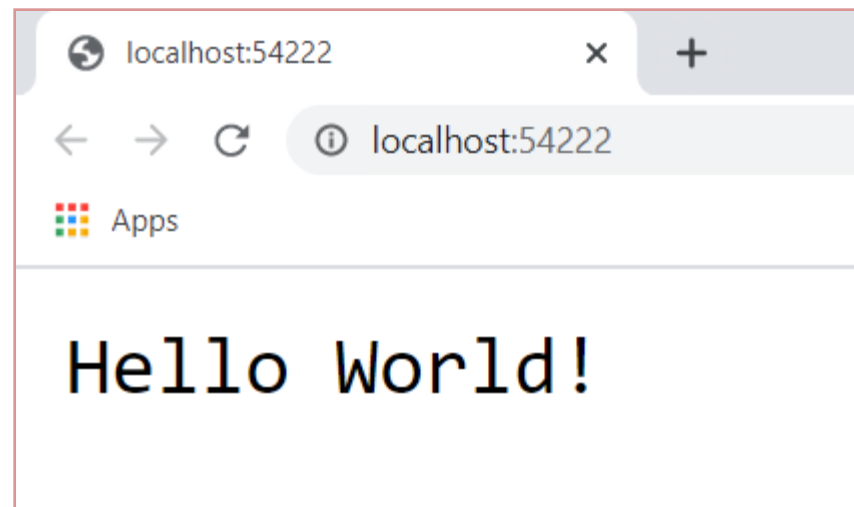
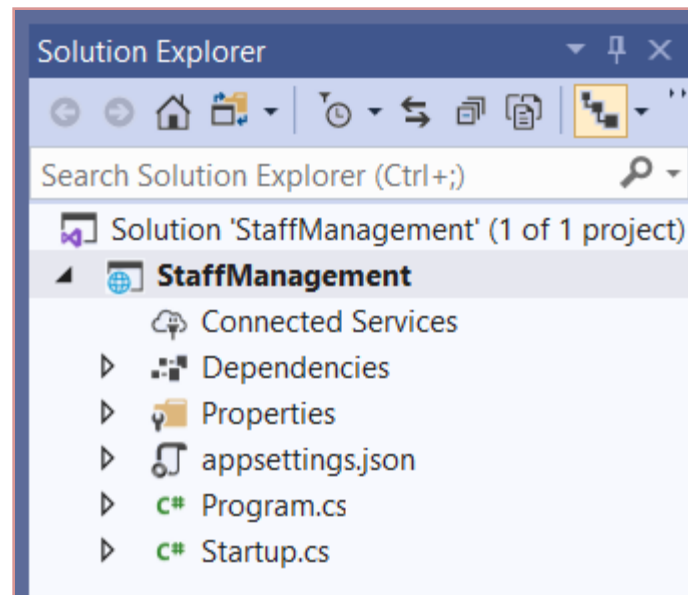
Linux

Author: Microsoft

Source: Templates 3.1.6

[Back](#)

Create





ASP.NET Core project file

- .csproj or .vbproj depending on the programming language used.
- No need to unload the project to edit the project file.
- File/ folder references are no longer included in the project file.
- The file system determines what files/ folders part of the project.



Your IDE

productivity
guide

code faster

Solution Explorer

Search Solution

Solution

Staff

Co

De

Pr

ap

Pr

St

St

St

St

St

St

St

St

St

St

St

St

St

St

St

St

St

St

St

St

St

St

St

St

St

St

St

St

St

St

St

St



Build



Rebuild



Clean



View



Analyze and Code Cleanup



Pack



Publish...



Configure Application Insights...



Overview



Scope to This



New Solution Explorer View



File Nesting



Edit Project File



Add



Manage NuGet Packages...



Manage Client-Side Libraries...



Manage User Secrets



Set as Startup Project



Debug



Source Control



Cut

Ctrl+X



Remove

Del



Rename

F2



Unload Project



Load Direct Dependencies of Project



Load Entire Dependency Tree of Project



Copy Full Path



Open Folder in File Explorer



Properties

Alt+Enter

Solution Explorer

Properties

StaffManage

File Name

Full Path

Project Folder

UserSecrets

UserSecrets

UserSecrets

UserSecrets

UserSecrets

UserSecrets

UserSecrets

UserSecrets

UserSecrets

UserSecrets

UserSecrets

UserSecrets



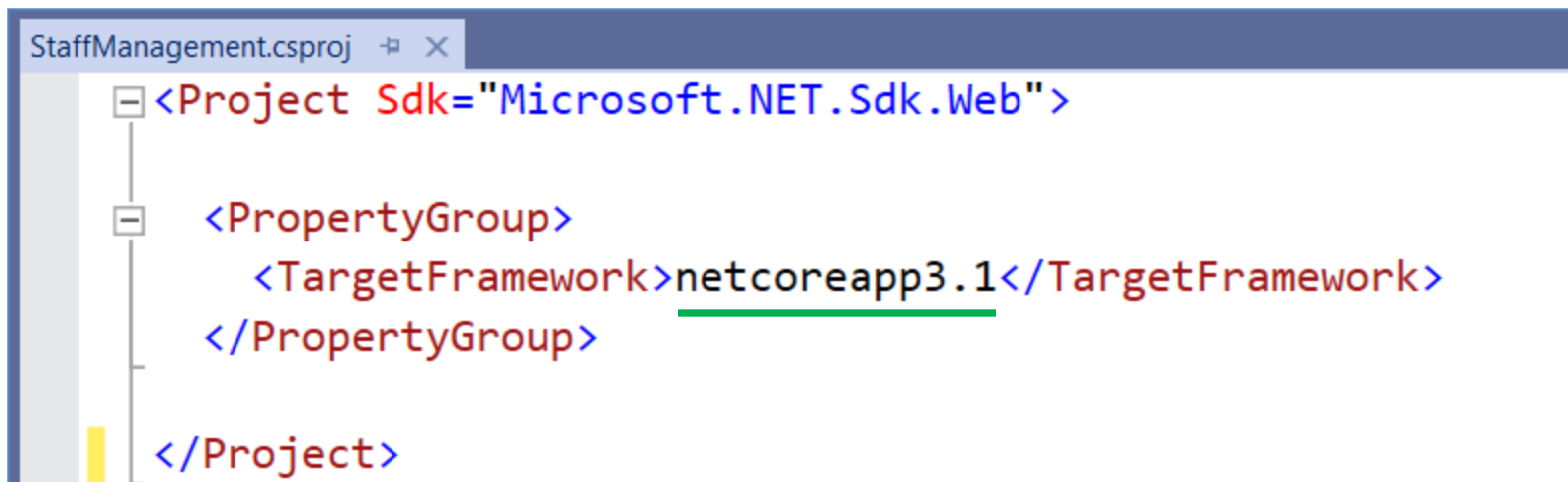
12:03 PM

13-07-2020



TargetFramework

- This element is used to specify the target framework for your application.
- A Target Framework Moniker (TFM) is a standardized token format for specifying the target framework of a .NET app or library.
- As you can see in the below example, our application targets one framework **netcoreapp3.1**. netcoreapp3.1 is the Moniker for .NET Core 3.1.



```
StaffManagement.csproj
<Project Sdk="Microsoft.NET.Sdk.Web">
  <PropertyGroup>
    <TargetFramework>netcoreapp3.1</TargetFramework>
  </PropertyGroup>
</Project>
```

Target Framework	Latest Stable Version	Target Framework Moniker (TFM)
.NET Standard	2.1	netstandard2.1
.NET Core	3.1	netcoreapp3.1
.NET Framework	4.8	net48

Main method in asp net core

- A Console application usually has a main method.
- Why do we have a Main() in ASP.NET Core web application?
- ASP.NET Core application initially starts as a Console application.
- The Main() configures ASP.NET Core and starts it and at that point it becomes an ASP.NET Core web application.

0 references

```
public class Program
```

```
{
```

0 references

```
public static void Main(string[] args)
```

```
{
```

```
    CreateHostBuilder(args).Build().Run();
```

```
}
```

1 reference

```
public static IHostBuilder CreateHostBuilder(string[] args) =>
```

```
    Host.CreateDefaultBuilder(args)
```

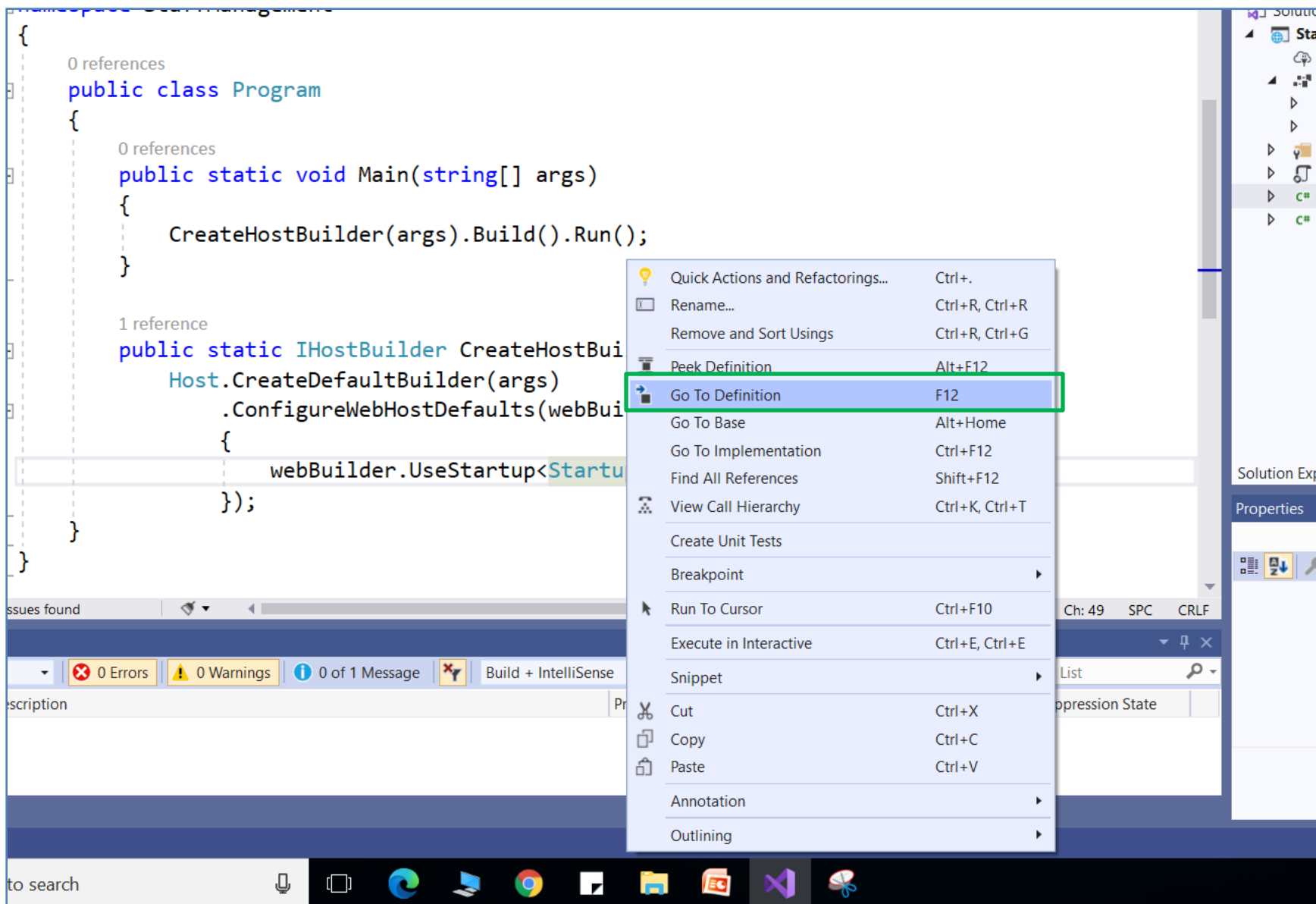
```
        .ConfigureWebHostDefaults(webBuilder =>
```

```
        {
```

```
            webBuilder.UseStartup<Startup>();
```

```
        });
```

```
}
```

```

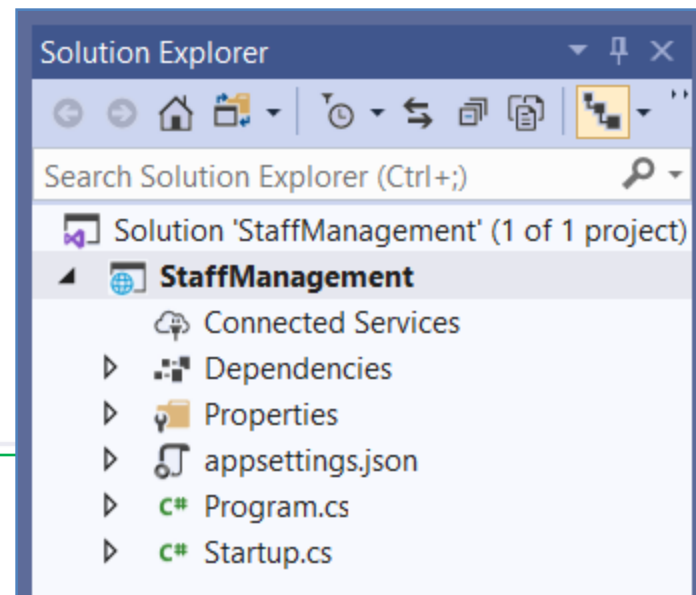
public class Startup
{
    // This method gets called by the runtime. Use this method to add services to the container.
    // For more information on how to configure your application, visit https://go.microsoft.com/fwlink/?LinkID=398940
    0 references
    public void ConfigureServices(IServiceCollection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

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

```
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
    0 references
    public void ConfigureServices(IServiceCollection services)
    {
    }

    // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
    0 references
    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!");
            });
        });
    }
}
```

What are the differences between
`app.UseRouting()` and
`app.UseEndpoints()`?

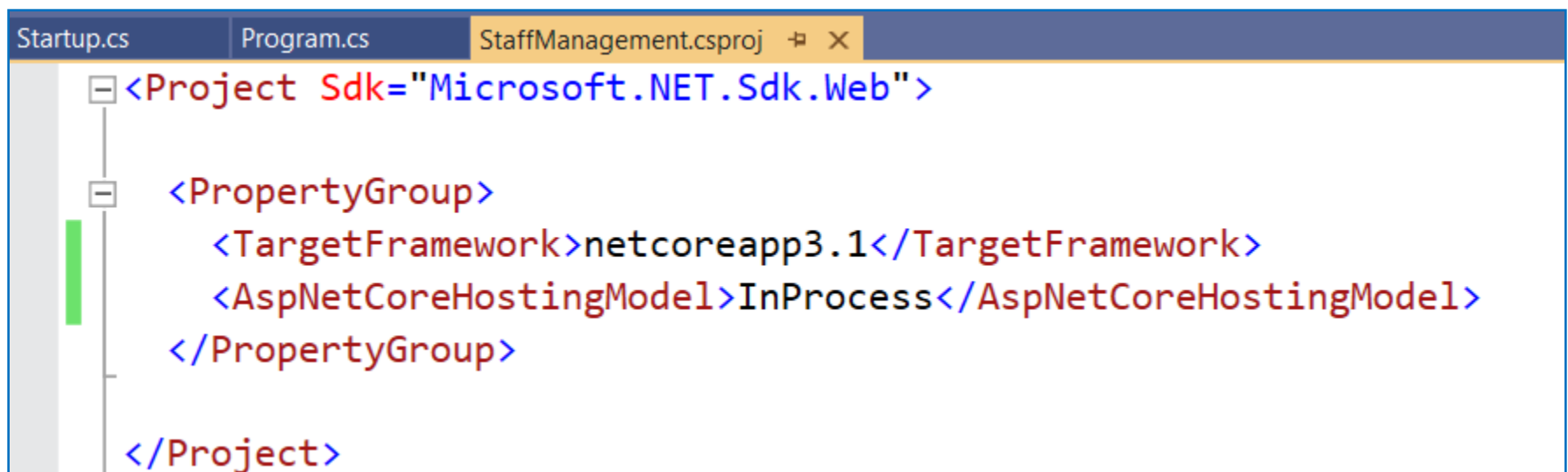
app.UseRouting()	app.UseEndpoints()
Find Endpoint	Execute Endpoint
Adds route matching to the middleware pipeline. This middleware looks at the set of endpoints defined in the app, and selects the best match based on the request.	Adds endpoint execution to the middleware pipeline. It runs the delegate associated with the selected endpoint.
UseRouting calculates what route should be used for a request URL path, but doesn't route at this point in the pipeline. UseRouting adds metadata that can be used by subsequent middleware.	UseEndpoints executes the Controller and corresponding handler.

ASP NET Core in process hosting

- When an ASP.NET core application is executed, the .NET runtime looks for Main() method which is the entry point for the application.
- The Main() method then calls CreateDefaultBuilder() static method of the Host class.
- This CreateDefaultBuilder() method performs several tasks like
 - **Setting up the web server**
 - Loading the host and application configuration from various configuration sources and
 - Configuring logging

What the CreateDefaultBuilder() method does to configure and set up the web server?

- An ASP.NET core application can be hosted InProcess or OutOfProcess.
- To configure InProcess hosting, add `<AspNetCoreHostingModel>` element to the app's project file with a value of InProcess.



```
Startup.cs  Program.cs  StaffManagement.csproj  X
<Project Sdk="Microsoft.NET.Sdk.Web">
  <PropertyGroup>
    <TargetFramework>netcoreapp3.1</TargetFramework>
    <AspNetCoreHostingModel>InProcess</AspNetCoreHostingModel>
  </PropertyGroup>
</Project>
```

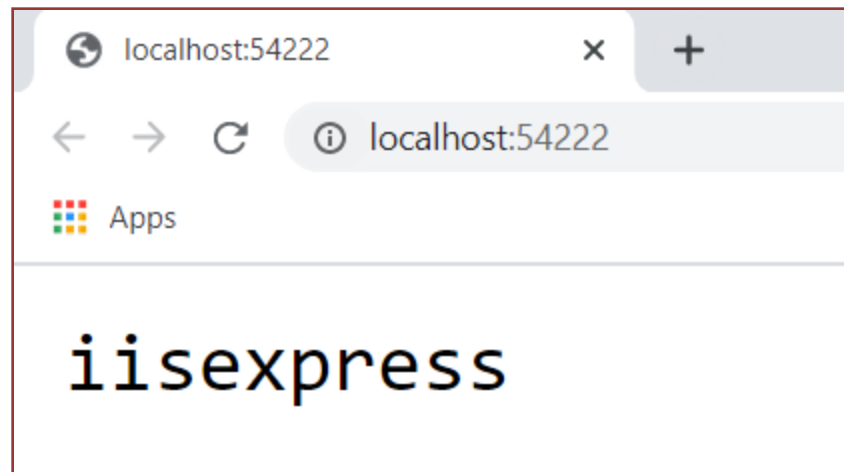

Cont.

- When we create a new ASP.NET Core project using one of the available project templates, the project defaults to the in-process hosting model for all IIS and IIS Express scenarios.
- In case of InProcess hosting, `CreateDefaultBuilder()` method calls `UseIIS()` method and host the app inside of the IIS worker process (`w3wp.exe` or `iisexpress.exe`).

```
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(System.Diagnostics.Process.GetCurrentProcess().ProcessName);
        });
    });
}
```



Cont.

- out of process hosting
 - There are 2 web servers - An internal web server and an external web server.
 - The internal web server is Kestrel and the external web server can be IIS, Nginx or Apache.
 - With InProcess hosting, there is only one web server i.e the IIS that hosts the asp.net core application.
 - So, we do not have the performance penalty of proxying requests between internal and external web server.

Cont.

- Kestrel
 - Kestrel is a cross-platform web server for ASP.NET Core.
 - It is supported on all platforms and versions that .NET Core supports.
 - It is included by default as internal server in ASP.NET Core.
 - Kestrel can be used, by itself as an edge server
 - i.e Internet-facing web server that can directly process the incoming HTTP requests from the client.
 - In Kestrel, the process used to host the app is **dotnet.exe**.
 - When we run a .NET Core application using the .NET Core CLI (Command-Line Interface), the application uses **Kestrel as the web server**.

Out of Process Hosting in ASP.NET Core

```
<PropertyGroup>  
  <TargetFramework>netcoreapp3.1</TargetFramework>  
  <AspNetCoreHostingModel>OutOfProcess</AspNetCoreHostingModel>  
</PropertyGroup>
```

- Add <AspNetCoreHostingModel> element to the app's project file with a value of **OutOfProcess**

```

app.UseEndpoints(endpoints =>
{
    endpoints.MapGet("/", async context =>
    {
        await context.Response.
            WriteAsync(System.Diagnostics.Process.GetCurrentProcess().ProcessName);
    });
});

```

localhost:54222

localhost:54222

Apps Playgroup English L... Playgroup Art - You... Playgroup Craft Wo...

StaffManagement

Elements Console Sources **Network** Performance Memory Application

Filter Hide data URLs All XHR JS CSS Img Media Font Doc WS Manife

10 ms 20 ms 30 ms 40 ms 50 ms

Name Headers Preview Response Initiator Timing Cookies

localhost

General

Request URL: http://localhost:54222/
 Request Method: GET
 Status Code: 200 OK
 Remote Address: [::1]:54222
 Referrer Policy: no-referrer-when-downgrade

Response Headers view source

Date: Sun, 02 Aug 2020 10:25:25 GMT
Server: Kestrel
 Transfer-Encoding: chunked
 X-Powered-By: ASP.NET

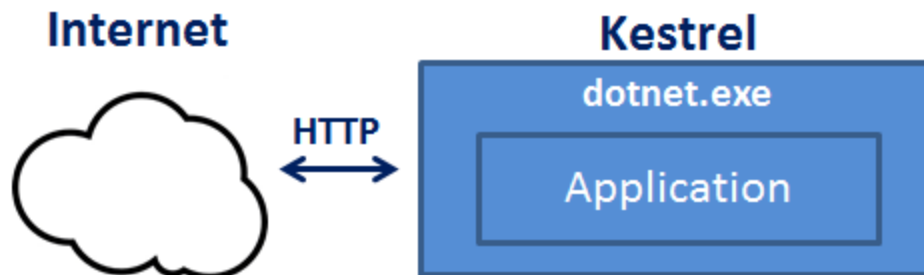
Request Headers view source

Accept: text/html,application/xhtml+xml,application/xml;q=...
 Accept-Encoding: gzip, deflate, br

1 requests 149 B transferred

Kestrel - a cross-platform web server

- **Kestrel can be used as the internet facing web server**
 - it process the incoming HTTP requests directly.
 - In this model we are not using an external web server.
 - Only Kestrel is used and it is this server that faces the internet, to directly handle the incoming HTTP requests.
 - When we run the asp.net core application using the .NET core CLI, Kestrel is the only web server that is used to handle and process the incoming HTTP request.



Cont.

- **Kestrel can also be used in combination with a reverse proxy server**, such as IIS, Nginx, or Apache.
- With a reverse proxy server in place, it receives incoming HTTP requests from the network and forwards them to the Kestrel server for processing.
- Upon processing the request, the Kestrel server sends the response to the reverse proxy server which then ultimately sends the response to the requested client over the network.



Cont.

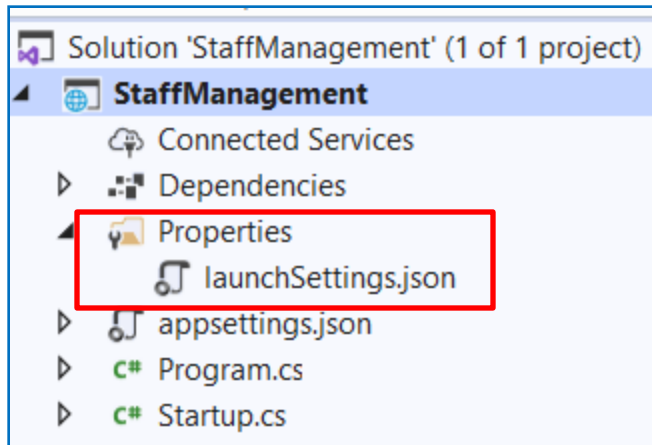
- If Kestrel can be used by itself as a web server, why do we need a reverse proxy server?
- Configuration, security and load balancing
- With **Out of Process Hosting**, whether you use a reverse proxy server or not, it is the Kestrel server that hosts the application and process the request.
- The reverse proxy server if used, takes the incoming HTTP request and forwards it to the Kestrel server. Similarly it takes the response from the Kestrel server and sends it to the client.

Can we run an asp.net core application without using the built in kestrel web server?



Squirrels plant thousands of new trees each year simply by forgetting where they put their acorns.

ASP NET Core launchsettings json file



```
launchSettings.json  Output  Microsoft.Common.C...entVersion.targets  Startup.cs  Progra
Schema: https://json.schemastore.org/launchsettings
1  {
2  }
3      "iisSettings": {
4          "windowsAuthentication": false,
5          "anonymousAuthentication": true,
6          "iisExpress": {
7              "applicationUrl": "http://localhost:54222",
8              "sslPort": 0
9          }
10     },
11     "profiles": {
12         "IIS Express": {
13             "commandName": "IISExpress",
14             "launchBrowser": true,
15             "environmentVariables": {
16                 "ASPNETCORE_ENVIRONMENT": "Development"
17             }
18         },
19         "StaffManagement": {
20             "commandName": "Project",
21             "launchBrowser": true,
22             "applicationUrl": "http://localhost:5000",
23             "environmentVariables": {
24                 "ASPNETCORE_ENVIRONMENT": "Development"
25             }
26         }
27     }
28 }
```

Cont.

- ASP NET Core launchsettings json file
 - This file is **only used on local development machine.**
 - We do not need it for publishing our asp.net core application.

Cont.

- If there are certain settings that you want your asp.net core application to use when you publish and deploy your app, store them in **appsettings.json file**.
 - We usually store our application configuration settings in this file.



The screenshot shows a code editor with the following elements:

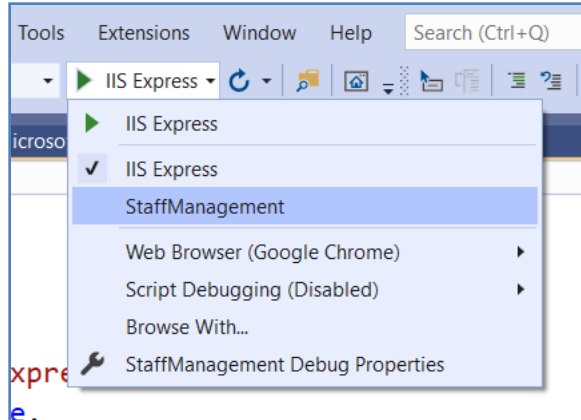
- Tab bar:** Contains four tabs: `appsettings.json` (selected), `launchSettings.json`, `Output`, and `Microsoft.Common.C...entVersion.targets`.
- Schema:** A link to `https://json.schemastore.org/appsettings`.
- Tree View:** A vertical tree on the left side of the editor showing the JSON structure:
 - Root object (line 1)
 - Object `"Logging": {` (line 2)
 - Object `"LogLevel": {` (line 3)
 - String `"Default": "Information",` (line 4)
 - String `"Microsoft": "Warning",` (line 5)
 - String `"Microsoft.Hosting.Lifetime": "Information"` (line 6)
 - Close brace `}` (line 7)
 - Comma `,` (line 8)
 - String `"AllowedHosts": "*"` (line 9)
 - Close brace `}` (line 10)
- Code Editor:** Displays the JSON content:

```
1  {
2    "Logging": {
3      "LogLevel": {
4        "Default": "Information",
5        "Microsoft": "Warning",
6        "Microsoft.Hosting.Lifetime": "Information"
7      }
8    },
9    "AllowedHosts": "*"
10 }
11
```

Cont.

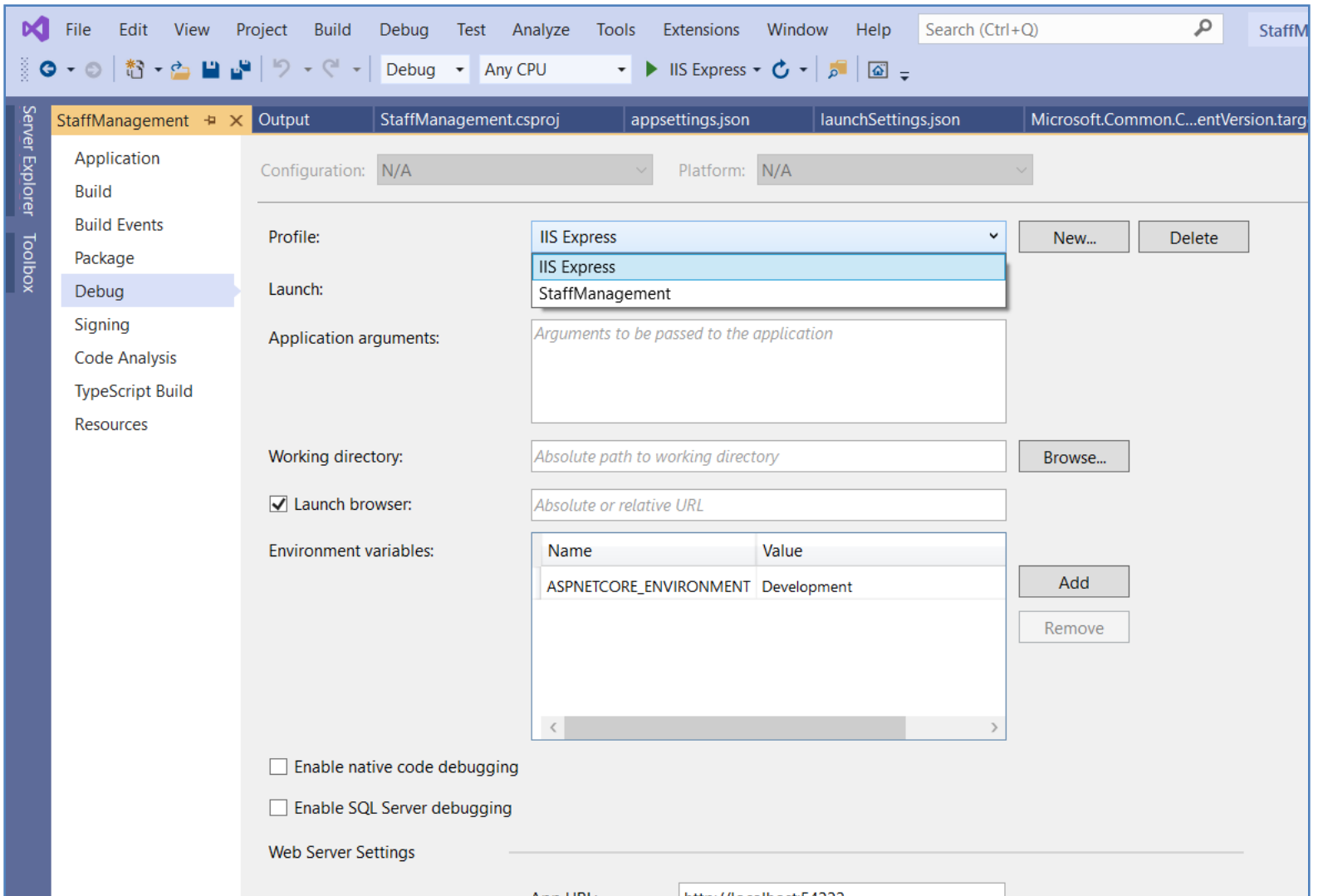
- We can also have **environment specific appsettings.json files**.
 - For example, appsettings.Staging.json for the staging environment.
- In ASP.NET Core, in addition to appsettings.json file, we also have other configuration sources like Environment variables, User Secrets, Command Line Arguments and even our own custom configuration source.

Launch Profiles in launchSettings.json file



```
{
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:54222",
      "sslPort": 0
    }
  },
  "profiles": {
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "StaffManagement": {
      "commandName": "Project",
      "launchBrowser": true,
      "applicationUrl": "http://localhost:5000",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```


commandName	AspNetCoreHosting Model	Internal Web Server	External Web Server
Project	Hosting Setting Ignored	Only one web server is used - Kestrel	
IISExpress	InProcess	Only one web server is used - IIS Express	
IISExpress	OutOfProcess	Kestrel	IIS Express
IIS	InProcess	Only one web server is used - IIS	
IIS	OutOfProcess	Kestrel	IIS



StaffManagement ✕

Output

StaffManagement.csproj

appsettings.json

launchSettings.json

Microsoft.Common.C...entVersion.tar

Application

Build

Build Events

Package

Debug

Signing

Code Analysis

TypeScript Build

Resources

Configuration: N/A

Platform: N/A

Profile: IIS Express

New...

Delete

Launch: IIS Express

Application arguments: Arguments to be passed to the application

Working directory: Absolute path to working directory

Browse...

☒ Launch browser: Absolute or relative URL

Environment variables:

Name	Value
ASPNETCORE_ENVIRONMENT	Development

Add

Remove

☐ Enable native code debugging

☐ Enable SQL Server debugging

Web Server Settings

App URL: http://localhost:54222

IIS Express Bitness: Default

Hosting Model: Default (In Process)

☐ Enable SSL

ASP NET Core appsettings json file

- In previous versions of ASP.NET, we store application configuration settings, like database connection strings for example, in **web.config** file.
- In ASP.NET Core application configuration settings can come from the following **different configurations sources**.
 - Files (appsettings.json, appsettings.{Environment}.json
 - where {Environment} is the app's current hosting environment)
 - User secrets
 - Environment variables
 - Command-line arguments

StaffManagement

Output

StaffManagement.csproj

appsettings.json

launchSettings.json

Schema: <https://json.schemastore.org/appsettings>

```
1  {
2    "Logging": {
3      "LogLevel": {
4        "Default": "Information",
5        "Microsoft": "Warning",
6        "Microsoft.Hosting.Lifetime": "Information"
7      }
8    },
9    "AllowedHosts": "*"
10 }
```

StaffManagement Output StaffManagement.csproj appsettings.json launchSettings

Schema: <https://json.schemastore.org/appsettings>

```
1  {
2  "Logging": {
3    "LogLevel": {
4      "Default": "Information",
5      "Microsoft": "Warning",
6      "Microsoft.Hosting.Lifetime": "Information"
7    }
8  },
9  "AllowedHosts": "*",
10 "MyKey": "Value of MyKey from appsettings.json"
11 }
```

```
StaffManagement.csproj  appsettings.json  launchSettings.json  Microsoft.Common.C...entVersion.targets  Startup.cs  X
  StaffManagement.Startup  _configuration

using System.Threading.Tasks;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

namespace StaffManagement
{
    2 references
    public class Startup
    {
        private IConfiguration _configuration;

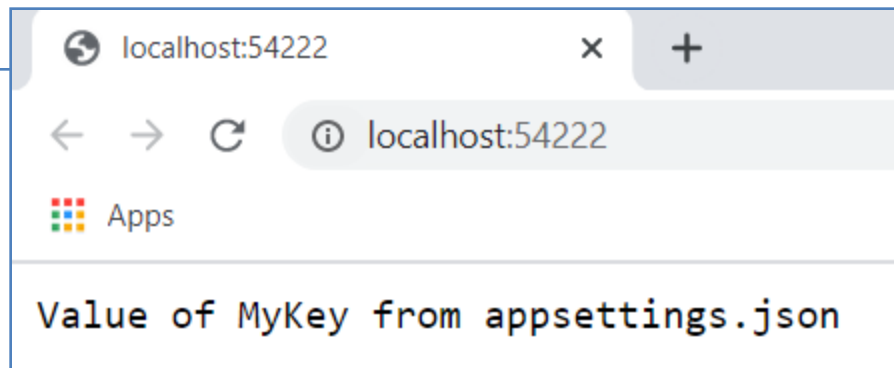
        // Notice we are using Dependency Injection here
        0 references
        public Startup(IConfiguration configuration)
        {
            _configuration = configuration;
        }

        // This method gets called by the runtime. Use this method to add services to the container.
    }
}
```

```
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(_configuration["MyKey"]);
        });
    });
}
```



ASP.NET Core IConfiguration service

- IConfiguration service is setup to read configuration information from all the various configuration sources in asp.net core
- If you have a configuration setting with the **same key in multiple configuration sources**, the later configuration sources override the earlier configuration sources
- CreateDefaultBuilder() method of the Host class which is automatically invoked when the application starts, reads the configuration sources in a specific order.

Cont.

- The following is the default order in which the various configuration sources are read
 1. appsettings.json,
 2. appsettings.{Environment}.json
 3. User secrets
 4. Environment variables
 5. Command-line arguments

ent.json | StaffManagement | appsettings.json | launchSettings.json

themastore.org/launchsettings

```
"windowsAuthentication": false,
"anonymousAuthentication": true,
"iisExpress": {
  "applicationUrl": "http://localhost:54222",
  "sslPort": 0
},
"profiles": {
  "IIS Express": {
    "commandName": "IISExpress",
    "launchBrowser": true,
    "environmentVariables": {
      "ASPNETCORE_ENVIRONMENT": "Development",
      "MyKey": "Value of MyKey from launchSettings.json"
    }
  }
},
"StaffManagement": {
  "commandName": "Project",
  "launchBrowser": true,
```

Solution Explorer

Search Solution Explorer (Ctrl+;)

- Solution 'StaffManagement' (1 of 1 projects)
- StaffManagement
 - Connected Services
 - Dependencies
 - Properties
 - launchSettings.json
 - appsettings.json
 - appsettings.Development.json
 - Program.cs
 - Startup.cs

Solution Explorer | Team Explorer

Properties

launchSettings.json File Properties

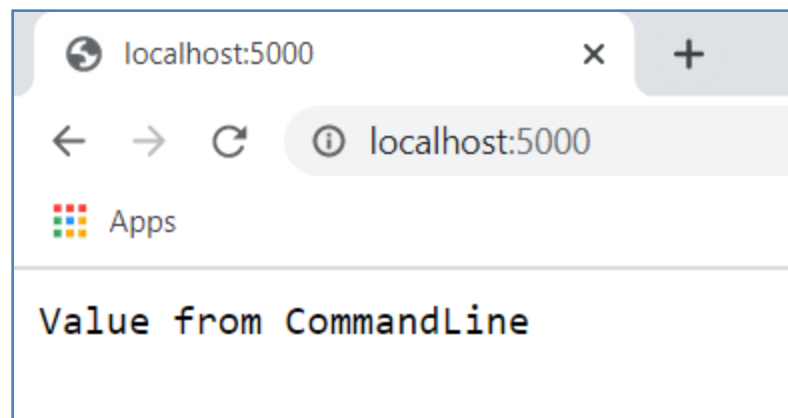
localhost:54222

localhost:54222

Apps

Value of MyKey from launchSettings.json

```
C:\Users\AAM\source\repos\StaffManagement\StaffManagement>dotnet run myKey="Value from CommandLine"
```



Human Body Facts!

- The left side of your body is controlled by the right side of your brain while the right side of your body is controlled by the left side of your brain.
- Antibiotics are only effective against bacteria, they won't help in fighting off a virus.
- Red blood cells carry oxygen around the body. They are created inside the bone marrow of your bones.

Middleware in ASP NET Core

- In ASP.NET Core, Middleware is a piece of software that can handle an HTTP request or response.
- A given middleware component in ASP.NET Core has a very specific purpose.
 - a middleware component that authenticates a user
 - another piece of middleware to handle errors
 - yet another middleware to serve static files such as JavaScript files, CSS files, Images etc.

Cont.

- It is these middleware components that we use to setup a request processing pipeline in ASP.NET Core.
- It is this pipeline that determines how a request is processed.
- The request pipeline is configured as part of the application startup by the `Configure()` method in `Startup.cs` file.

```
appsettings.json | launchSettings.json | Program.cs | appsettings.Development.json | Microsoft.Common.C...entVersion.targets | Startup.cs
```

StaffManagement.Startup

Configure(IApplicationBuilder app, IWebHostEnvironment env)

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

0 references

```
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(_configuration["MyKey"]);
    //    });
    //});

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




- In ASP.NET Core, a **Middleware component** has **access to both - the incoming request and the outgoing response.**
- So a Middleware component may process an incoming request and pass that request to the next piece of middleware in the pipeline for further processing.
- For example, if you have a **logging middleware**, it might simply log the time the request is made and pass the request to the next piece of middleware for further processing.



- **A middleware component may handle the request and decide not to call the next middleware in the pipeline.**
 - This is called short-circuiting the request pipeline.
- Short-circuiting is often desirable because it avoids unnecessary work.
 - For example, if the request is for a static file like an image or css file, the StaticFiles middleware can handle and serve that request and short-circuit the rest of the pipeline.
 - This means in our case, the StaticFiles middleware will not call the MVC middleware if the request is for a static file.



- A middleware component may handle an incoming HTTP request by generating an HTTP response.
 - For example, **mvcmiddleware** in the pipeline handles a request to the URL **/employees** and returns a list of employees.
- **A middleware component may also process the outgoing response.**
 - For example, the logging middleware component may log the time the response is sent.
 - In addition it may also calculate the over all time taken to process the request by computing the difference between request received and response sent times.

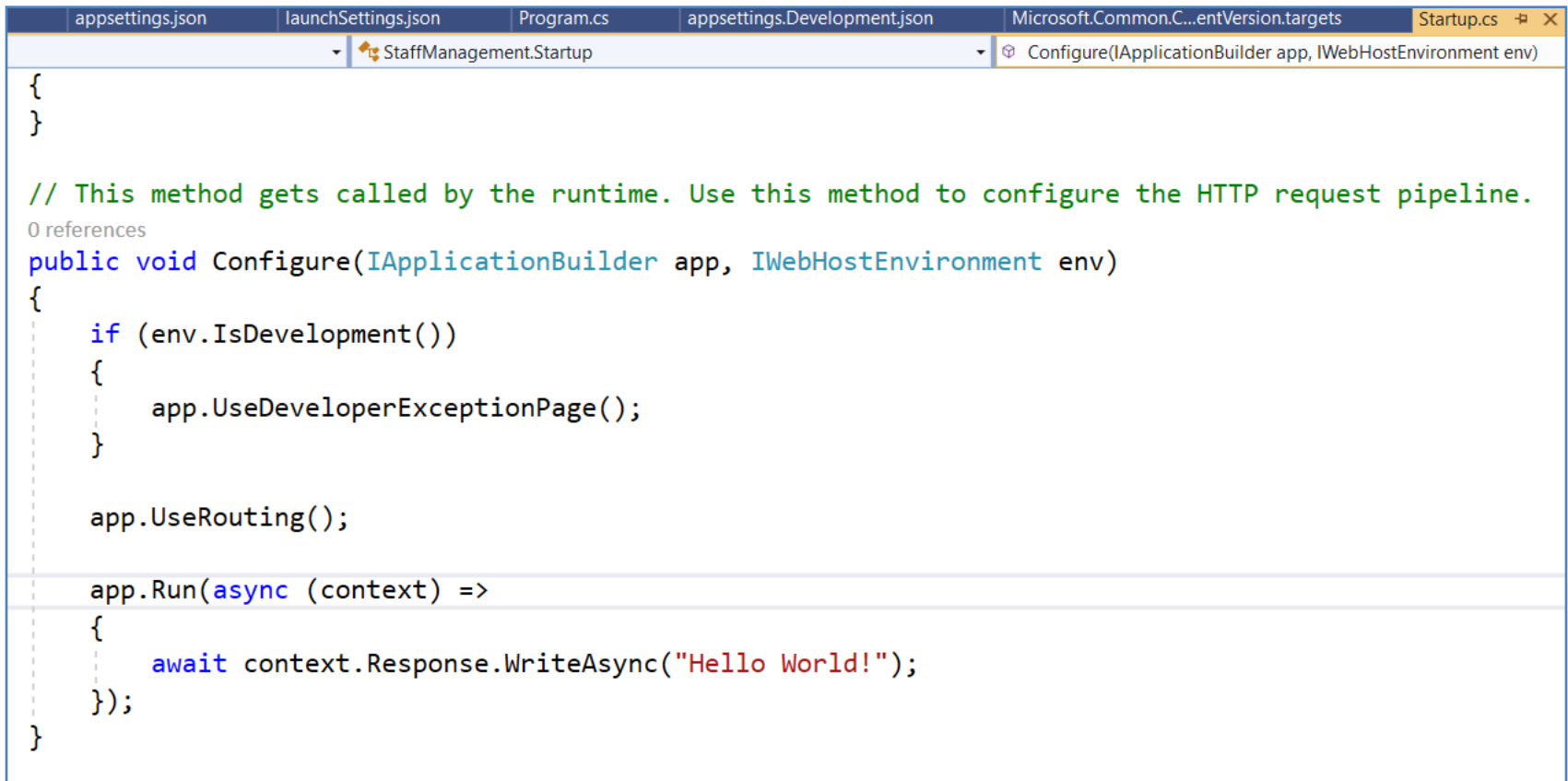


- **Middleware components are executed in the order they are added to the pipeline.**
 - Care should be taken to add the middleware in the right order, otherwise the application may not function as expected.
- **The middleware components are available as NuGet packages.**
 - This means updates are now handled by NuGet, providing the ability to update each middleware separately.



- Depending on your application requirements you may add as many or as few middleware components to the request processing pipeline.
 - For example, if you are developing simple web application with a few static HTML pages and images, then your request processing pipeline may contain just "**StaticFiles**" middleware.
 - On the other hand, if you are developing a secure data driven web application then you may need several middleware components like StaticFiles middleware, Authentication middleware, Authorization middleware, MVC middleware etc.

Configure ASP NET Core request processing pipeline

A screenshot of the Visual Studio IDE showing the 'Configure' method in 'Startup.cs'. The top of the window displays several open files: 'appsettings.json', 'launchSettings.json', 'Program.cs', 'appsettings.Development.json', 'Microsoft.Common.C...entVersion.targets', and 'Startup.cs'. The 'Startup.cs' file is selected, and the 'Configure' method is highlighted. The code is as follows:

```
{  
}  
  
// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.  
0 references  
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)  
{  
    if (env.IsDevelopment())  
    {  
        app.UseDeveloperExceptionPage();  
    }  
  
    app.UseRouting();  
  
    app.Run(async (context) =>  
    {  
        await context.Response.WriteAsync("Hello World!");  
    });  
}
```

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

- We are using Run() method to add middleware to our application's request processing pipeline
- If you hover the mouse over the Run() method, from the intellisense you can see that this Run() method is implemented as an extension method of IApplicationBuilder interface.
 - This is the reason we are able to invoke this Run() method on IApplicationBuilder object app.
- The parameter that we are passing to the Run() method is a RequestDelegate which we can see from the intellisense.
 - RequestDelegate is a delegate that has HttpContext object as a parameter.
- It is through this HttpContext object, the middleware gains access to both the incoming http request and outgoing http response.
- At the moment, we are passing request delegate inline as an anonymous method using a lambda.
- With this Run() extension method we can only add a terminal middleware to the request pipeline.
- A terminal middleware is a middleware that does not call the next middleware in the pipeline

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline

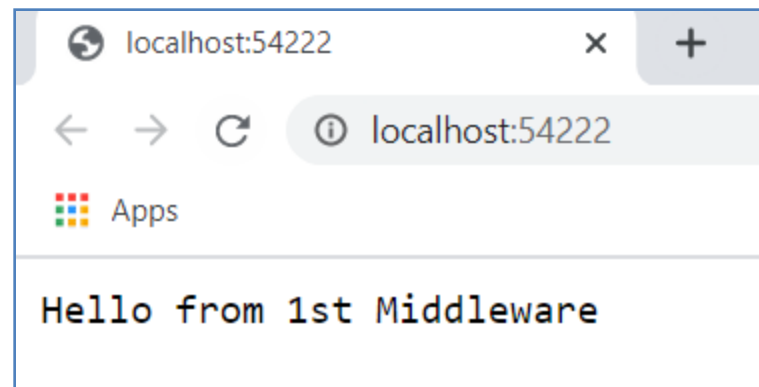
0 references

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

    app.UseRouting();

    app.Run(async (context) =>
    {
        await context.Response.WriteAsync("Hello from 1st Middleware");
    });

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




```

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

    app.UseRouting();

    app.Use(async (context, next) =>
    {
        await context.Response.WriteAsync("Hello from 1st Middleware");
        await next();
    });

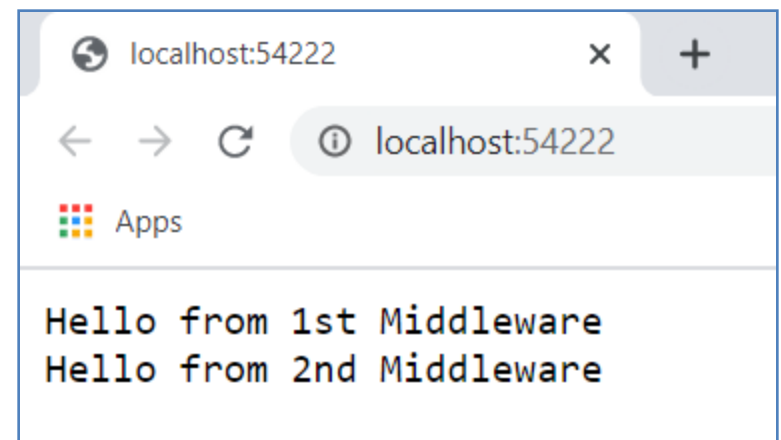
    app.Run(async (context) =>
    {
        await context.Response.WriteAsync("Hello from 2nd Middleware");
    });
}

```

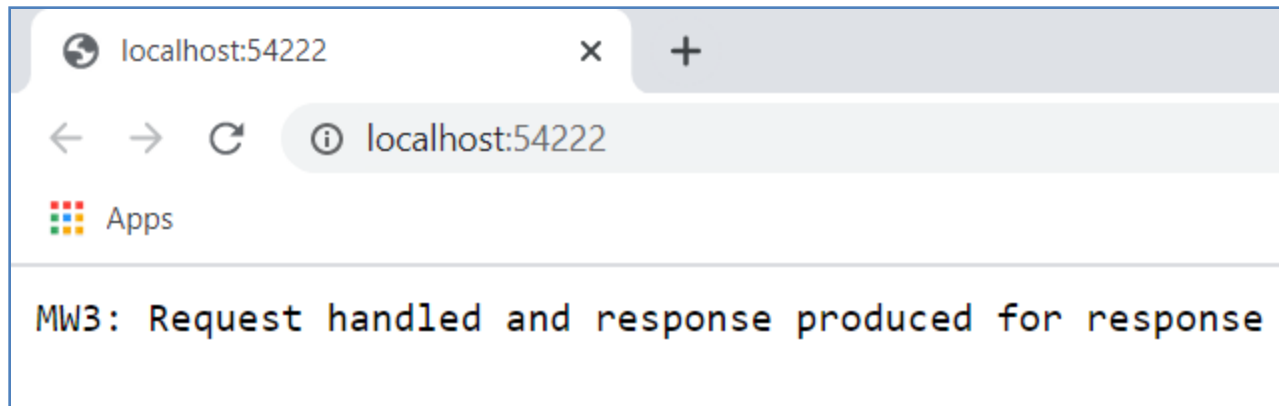
```

app.Use(async (context, next) =>
{
    await context.Response.WriteAsync("Hello from 1st Middleware");
    await next();
});

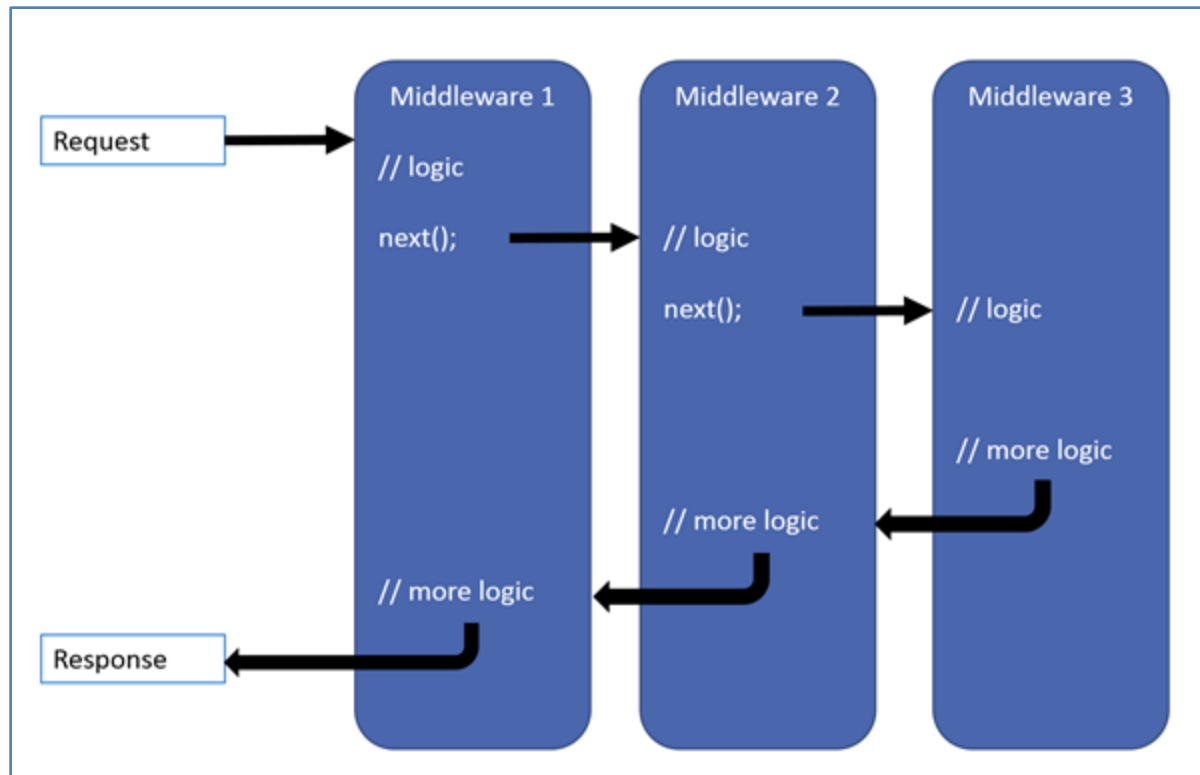
```

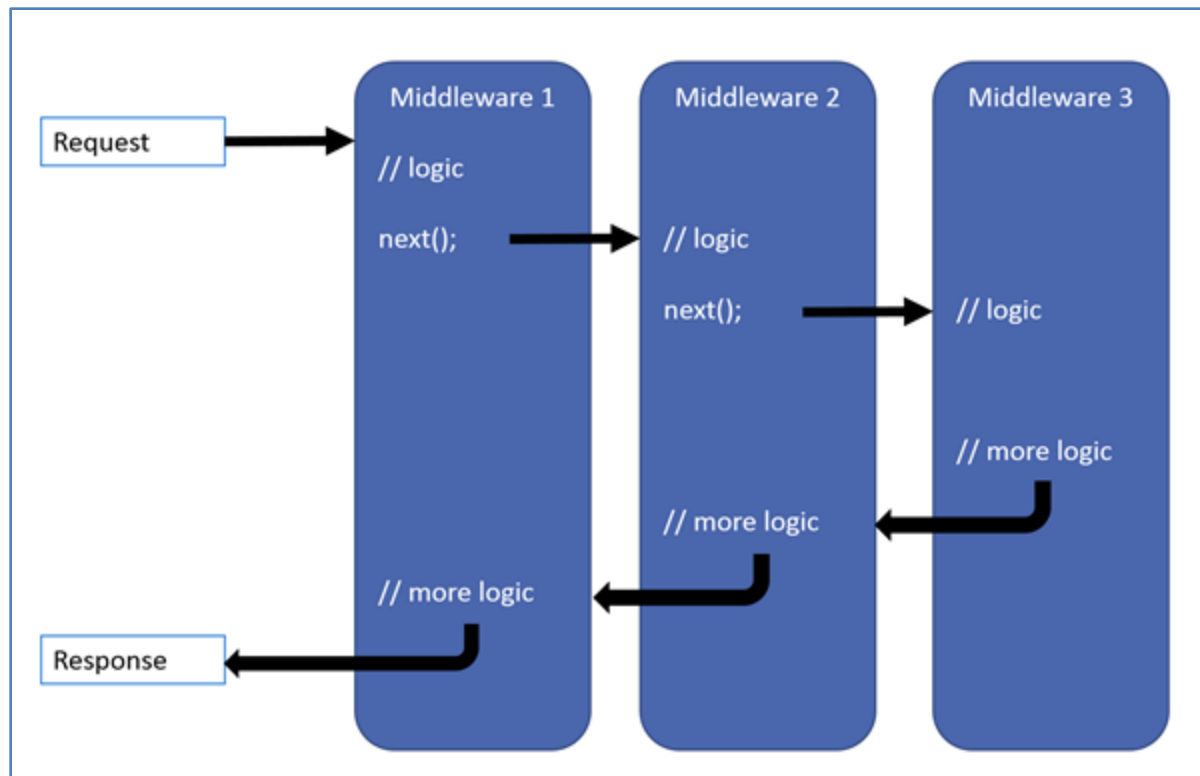


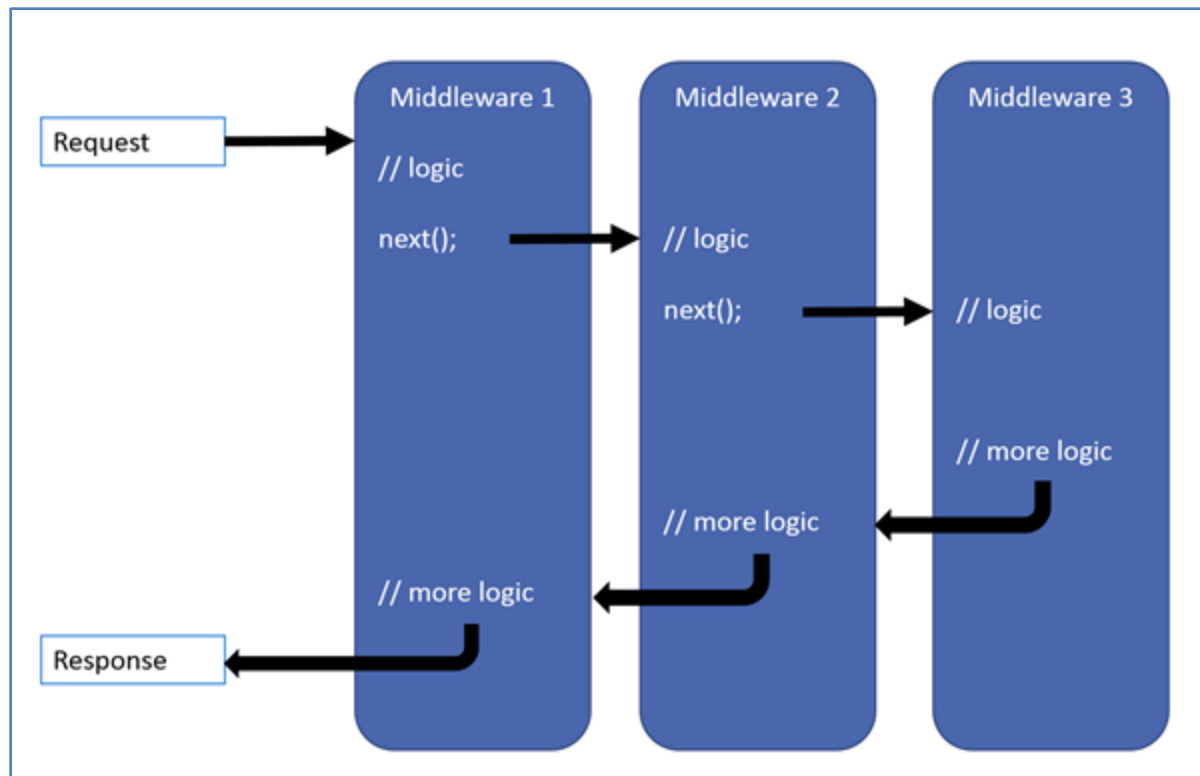
```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env, ILogger<Startup> logger)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    app.UseRouting();
    app.Use(async (context, next) =>
    {
        logger.LogInformation("MW1: Incoming Request");
        await next();
        logger.LogInformation("MW1: Outgoing Response");
    });
    app.Use(async (context, next) =>
    {
        logger.LogInformation("MW2: Incoming Request");
        await next();
        logger.LogInformation("MW2: Outgoing Response");
    });
    app.Run(async (context) =>
    {
        await context.Response.WriteAsync("MW3: Request handled and response produced for response");
        logger.LogInformation("MW3: Request handled and response produced");
    });
}
```

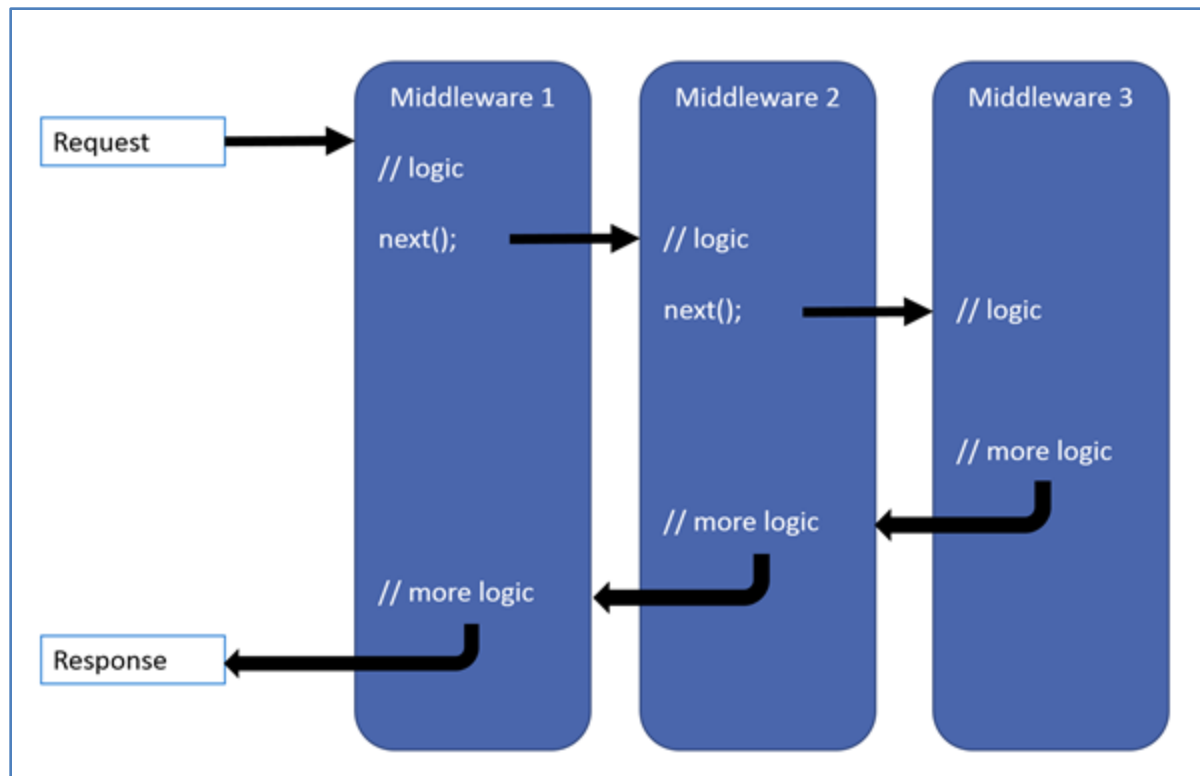


```
StaffManagement.Startup: Information: MW1: Incoming Request  
StaffManagement.Startup: Information: MW2: Incoming Request  
'iisexpress.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Micro  
StaffManagement.Startup: Information: MW3: Request handled and response produced  
StaffManagement.Startup: Information: MW2: Outgoing Response  
StaffManagement.Startup: Information: MW1: Outgoing Response
```



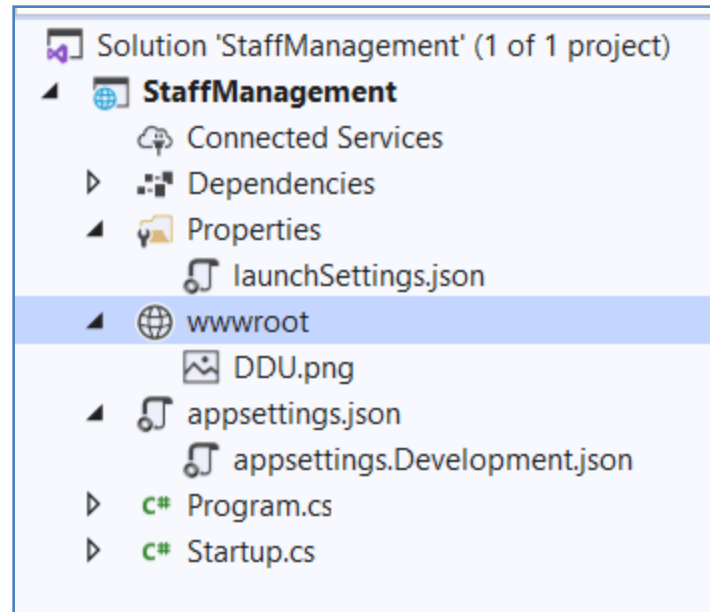


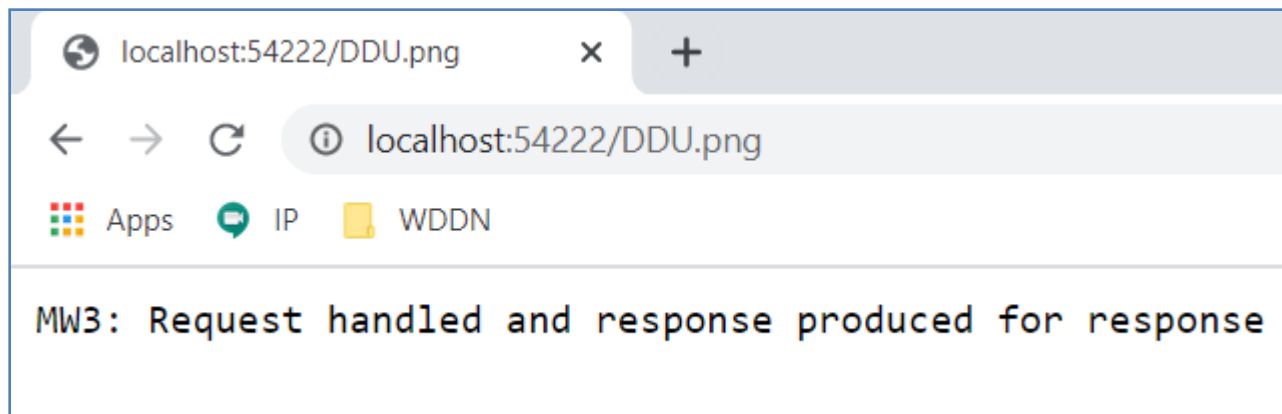




Static files in asp net core

- By default, an asp.net core application will not serve static files
- The default directory for static files is **wwwroot** and this directory must be in the root project folder

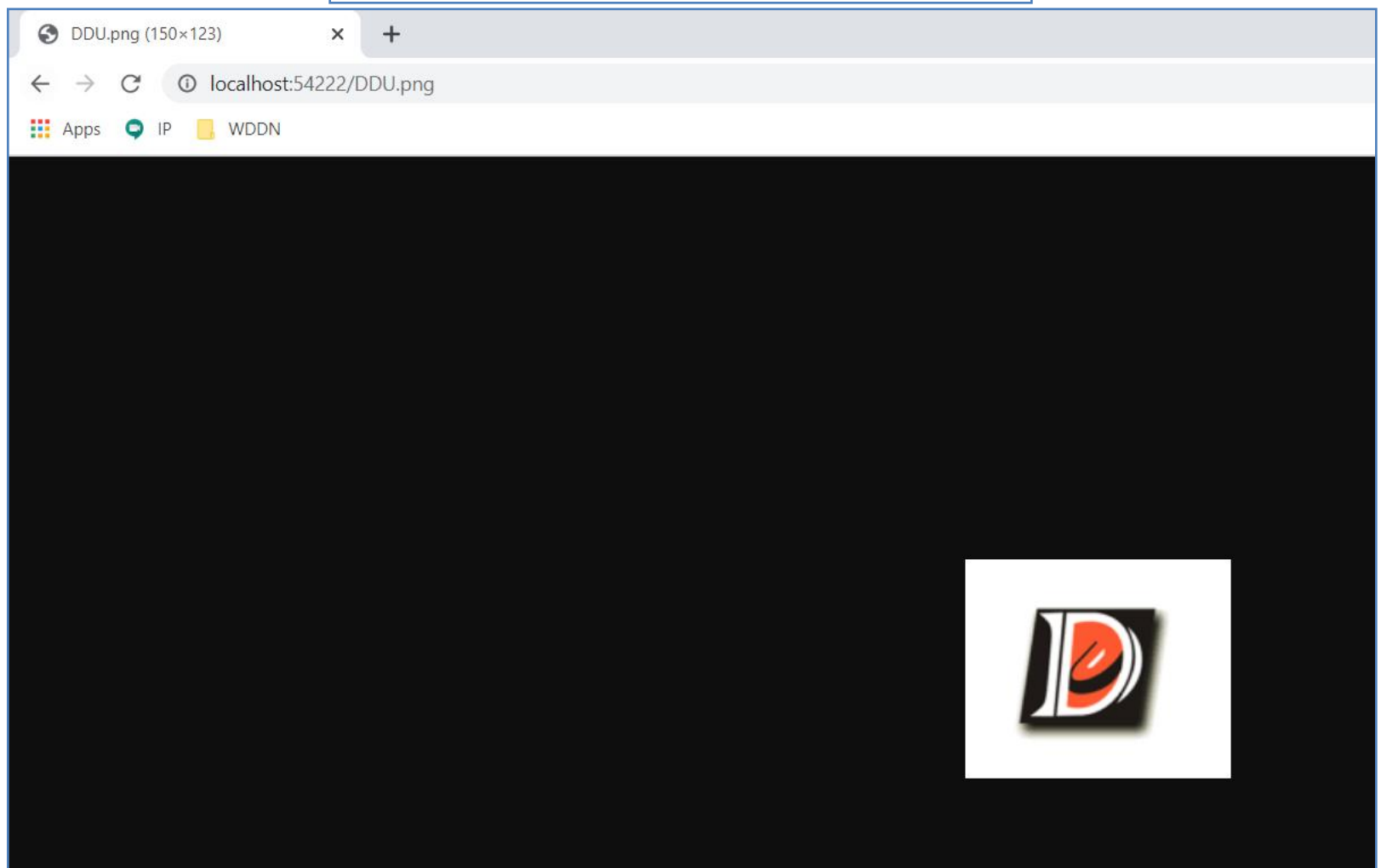
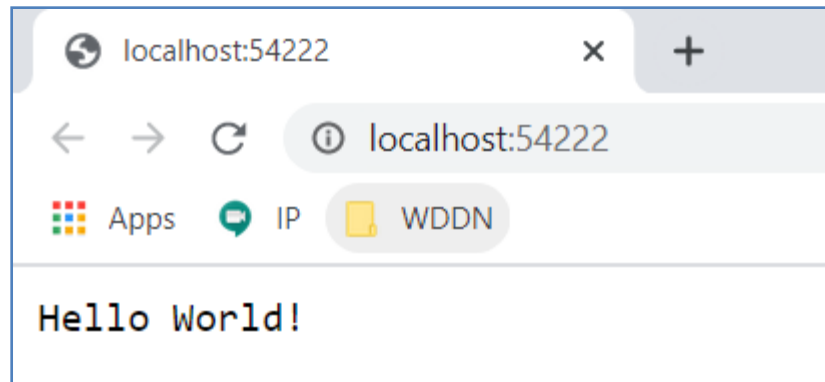


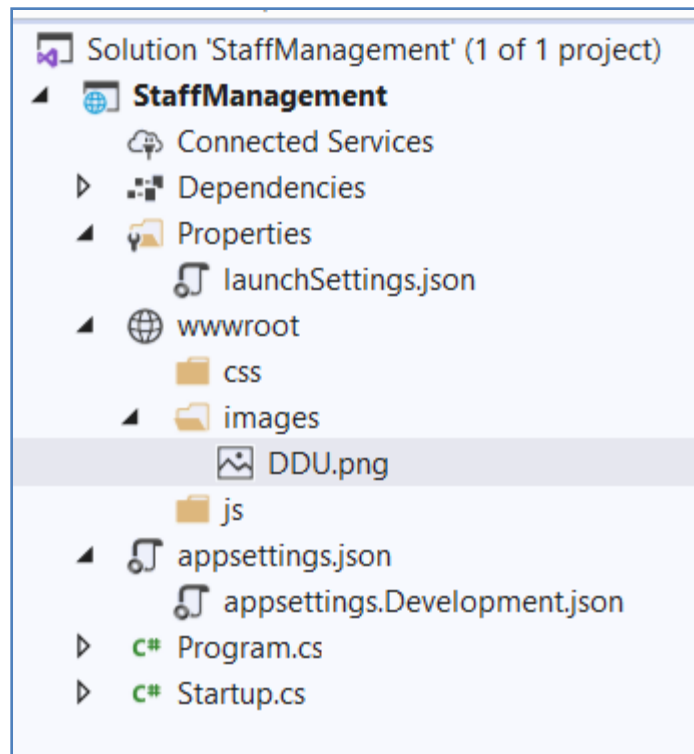


```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    app.UseStaticFiles();

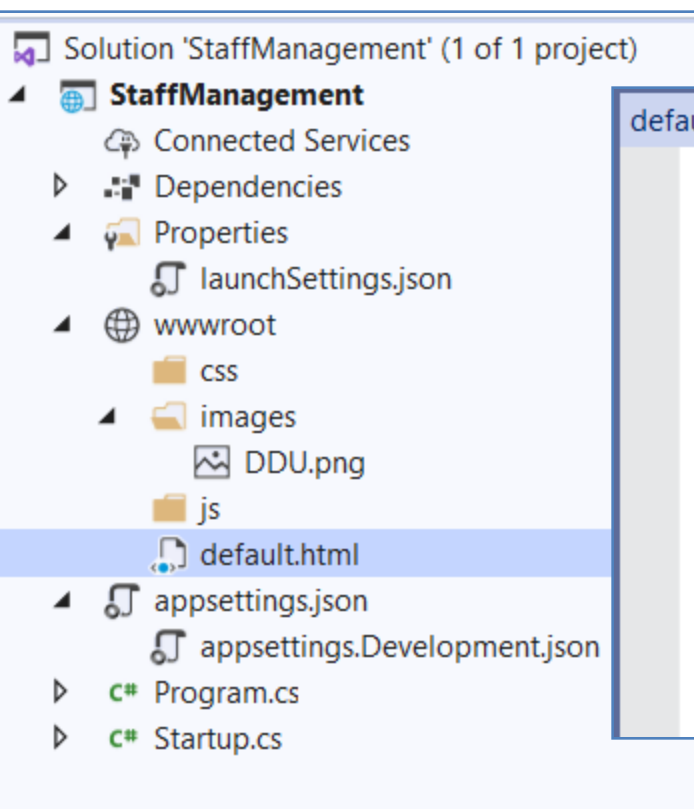
    app.UseRouting();

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





Serving a default document

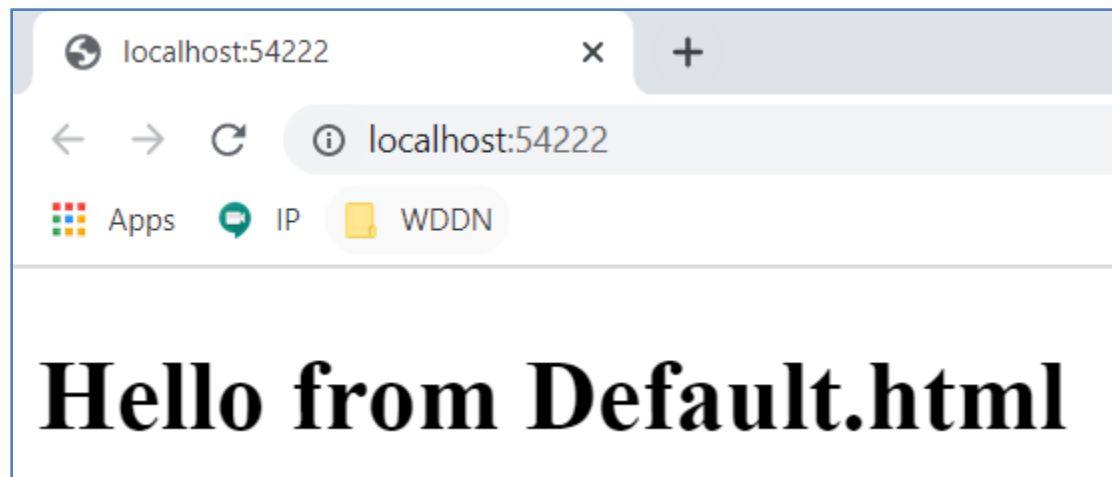


```
default.html  Output  appsettings.json  launchSettings.json  Pro
1      <!DOCTYPE html>
2      <html>
3      <head>
4          <meta charset="utf-8" />
5          <title></title>
6      </head>
7      <body>
8          <h1>Hello from Default.html</h1>
9      </body>
10     </html>
```

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    // Add Default Files Middleware
    app.UseDefaultFiles();
    // Add Static Files Middleware
    app.UseStaticFiles();

    app.UseRouting();

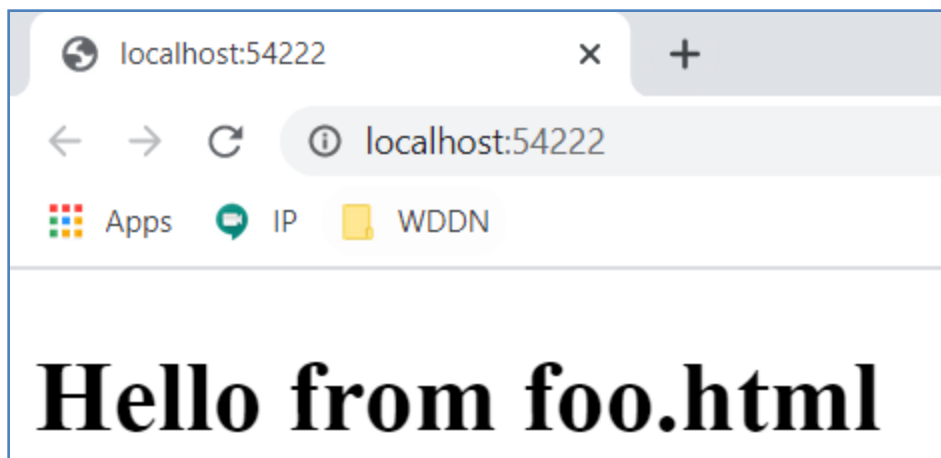
    app.Run(async (context) =>
    {
        await context.Response.WriteAsync("Hello World!");
    });
}
```



```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    // Specify foo.html as the default document
    DefaultFilesOptions defaultFilesOptions = new DefaultFilesOptions();
    defaultFilesOptions.DefaultFileNames.Clear();
    defaultFilesOptions.DefaultFileNames.Add("foo.html");
    // Add Default Files Middleware
    app.UseDefaultFiles(defaultFilesOptions);
    // Add Static Files Middleware
    app.UseStaticFiles();

    app.UseRouting();

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



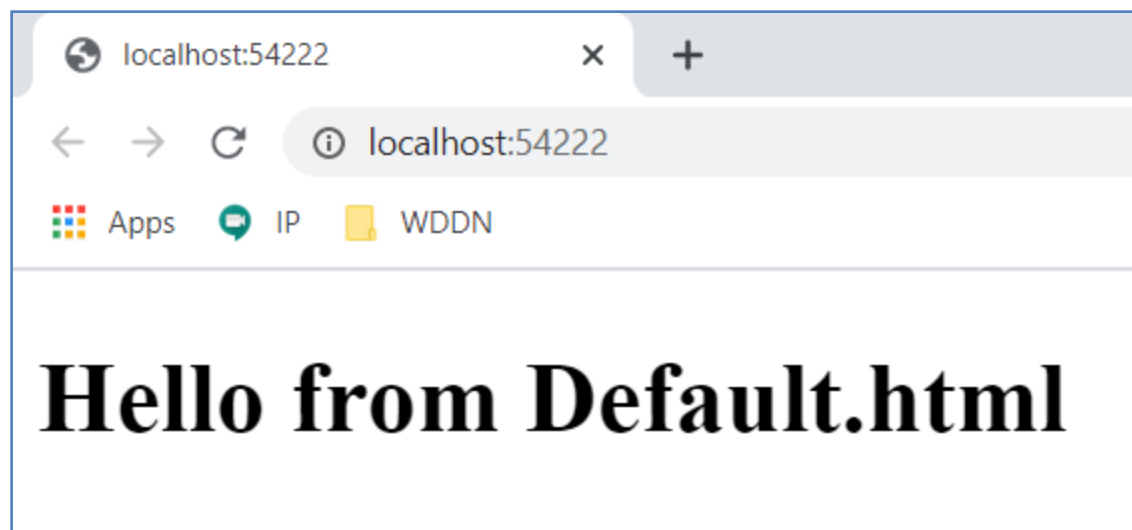
UseDeveloperExceptionPage Middleware

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

    app.UseFileServer();

    app.UseRouting();

    app.Run(async (context) =>
    {
        throw new Exception("Some error processing the request");
        await context.Response.WriteAsync("Hello World!");
    });
}
```



Internal Server Error

localhost:54222/xyz.html

AppsIPWDDN

An unhandled exception occurred while processing the request.

Exception: Some error processing the request

StaffManagement.Startup+<>c+<<Configure>b__3_0>d.MoveNext() in **Startup.cs**, line 44

Stack

QueryCookiesHeadersRouting

Exception: Some error processing the request

+

StaffManagement.Startup+<>c+<<Configure>b__3_0>d.MoveNext() in **Startup.cs**
44. throw new Exception("Some error processing the request");
Microsoft.AspNetCore.Diagnostics.DeveloperExceptionPageMiddleware.Invoke(HttpContext context)

Show raw exception details

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        DeveloperExceptionPageOptions developerExceptionPageOptions =
            new DeveloperExceptionPageOptions
            {
                SourceCodeLineCount = 10
            };
        app.UseDeveloperExceptionPage(developerExceptionPageOptions);
    }

    app.UseFileServer();

    app.UseRouting();

    app.Run(async (context) =>
    {
        throw new Exception("Some error processing the request");
        await context.Response.WriteAsync("Hello World!");
    });
}
```

Internal Server Error

localhost:54222/xyz.html

AppsIPWDDN

An unhandled exception occurred while processing the request.

Exception: Some error processing the request

StaffManagement.Startup+<>c+<<Configure>b_3_0>d.MoveNext() in **Startup.cs**, line 49

StackQueryCookiesHeadersRouting

Exception: Some error processing the request

StaffManagement.Startup+<>c+<<Configure>b_3_0>d.MoveNext() in **Startup.cs**

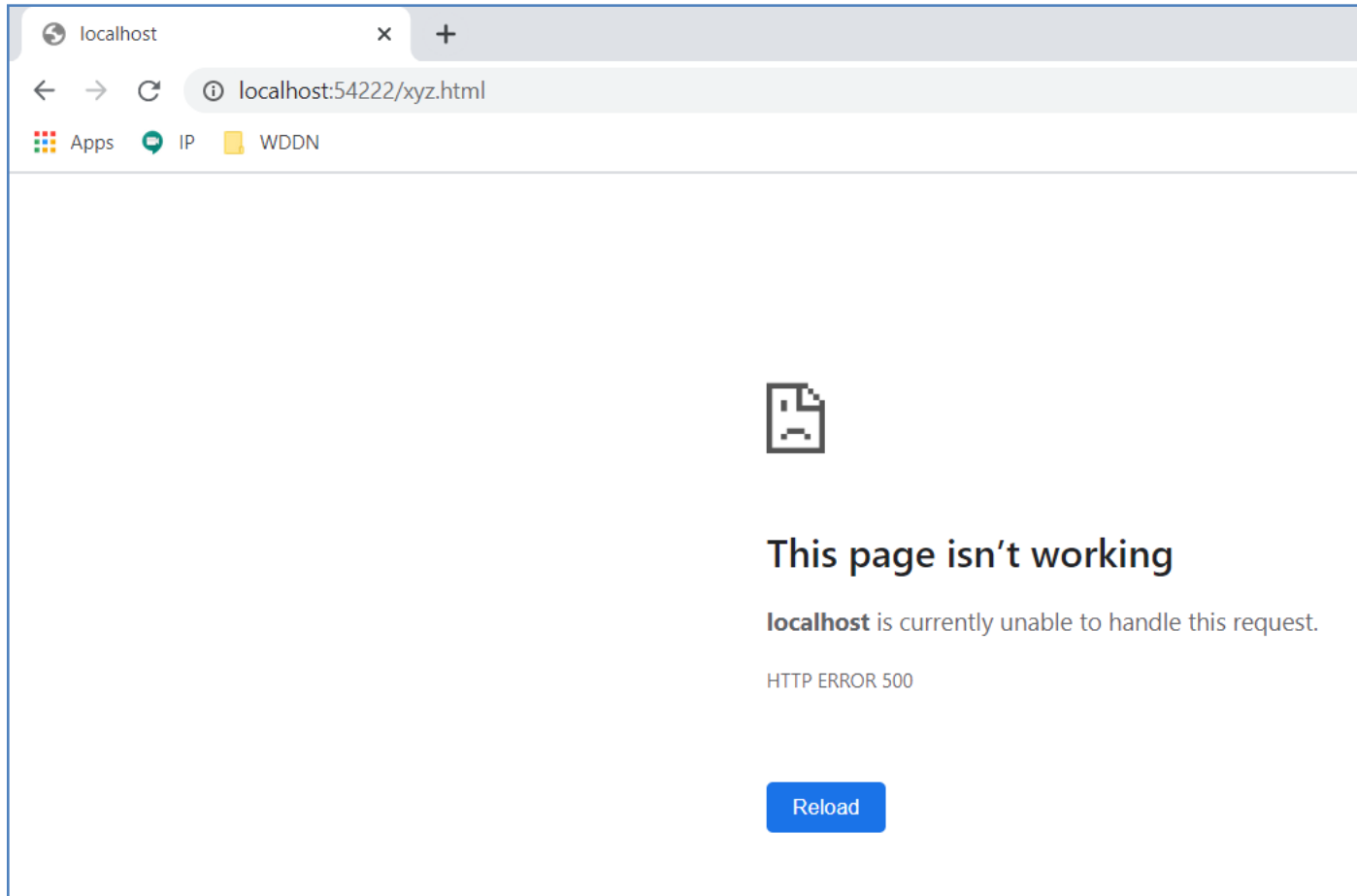
-

```
39.         };
40.         app.UseDeveloperExceptionPage(developerExceptionPageOptions);
41.     }
42.
43.     app.UseFileServer();
44.
45.     app.UseRouting();
46.
47.     app.Run(async (context) =>
48.     {
49.         throw new Exception("Some error processing the request");
50.         await context.Response.WriteAsync("Hello World!");
51.     });
52. }
53. }
54. }
```

Microsoft.AspNetCore.Diagnostics.DeveloperExceptionPageMiddleware.Invoke(HttpContext context)

How UseDeveloperExceptionPage Middleware works

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.UseFileServer();
    app.UseRouting();
    app.Run(async (context) =>
    {
        throw new Exception("Some error processing the request");
        await context.Response.WriteAsync("Hello World!");
    });
    if (env.IsDevelopment())
    {
        DeveloperExceptionPageOptions developerExceptionPageOptions =
            new DeveloperExceptionPageOptions
            {
                SourceCodeLineCount = 10
            };
        app.UseDeveloperExceptionPage(developerExceptionPageOptions);
    }
}
```



Software Development Environments



A diagram showing three software development environments. At the top, 'Development' (purple box) and 'Staging' (orange box) are side-by-side. Below them, centered, is 'Production' (green box).

Development

Staging

Production

Development Environment

- Software developers typically use this environment for day to day development work.
- We want non-minified JavaScript and CSS files to be loaded on a development environment for ease of debugging.
- Similarly we want developer exception page if there is an unhandled exception so we can understand the root cause of the exception and fix it if required.

Staging Environment

- Many organizations, try to keep their staging environment as identical as possible to the actual production environment.
- The primary reason for this environment is to identify any deployment related issues.
- Also, if you are developing a B2B (Business to Business) application, you may be interfacing with other service provider systems.
- Many organizations, usually setup their staging environment to interface with the service providers as well, for complete end to end testing.
- We usually do not troubleshoot and debug on a staging environment, so for better performance we want minified JavaScript and CSS files to be loaded.
- If there is an unhandled exception, display user friendly error page instead of the developer exception page. A user friendly error page will not contain any technical details.
 - It contains a generic message like the following
"Something has gone wrong, please email, chat or call our application support using the contact details below"

Production Environment

- The actual live environment, that we use for day to day business.
- Production environment should be configured for maximum security and performance.
 - So load minified JavaScript and CSS files to improve the performance.
- For better security, display a User Friendly Error Page instead of the Developer Exception Page.
 - The technical details on the Developer Exception Page does not make sense to the end user and they can be used by malicious user to break into your application.

Program.cs launchSettings.json appsettings.Development.json Microsoft.Common.C...entV

son.schemastore.org/launchsettings

```
}  
},  
"profiles": {  
  "IIS Express": {  
    "commandName": "IISExpress",  
    "launchBrowser": true,  
    "environmentVariables": {  
      "ASPNETCORE_ENVIRONMENT": "Development",  
      "MyKey": "Value of MyKey from launchSettings.json"  
    }  
  }  
}
```

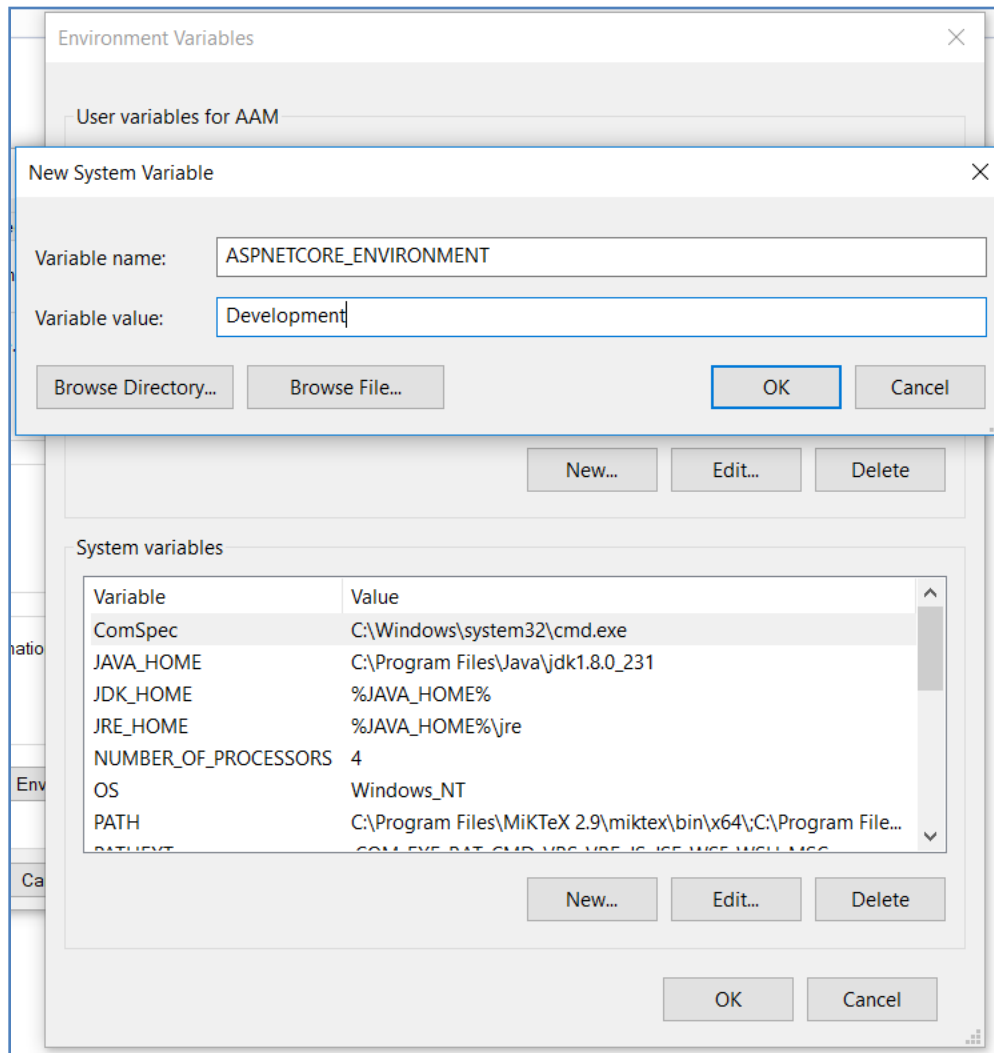
localhost:54222

localhost:54222

Apps IP WDDN

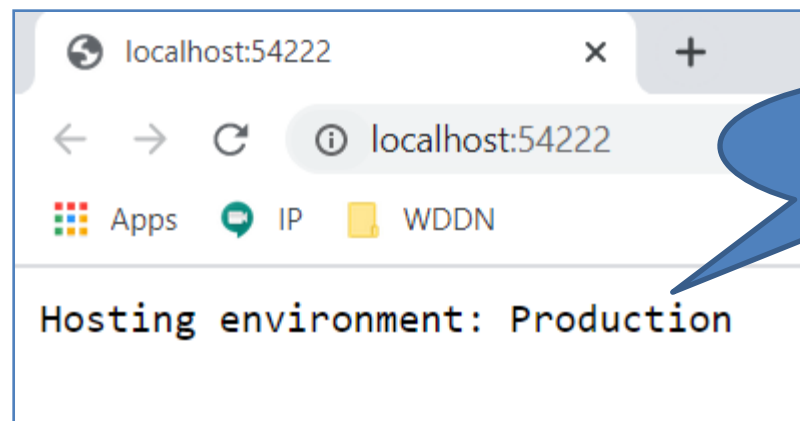
Hosting environmentDevelopment

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)  
{  
    if (env.IsDevelopment())  
    {  
        app.UseDeveloperExceptionPage();  
    }  
    app.UseStaticFiles();  
    app.UseRouting();  
    app.Run(async (context) =>  
    {  
        await context.Response.WriteAsync("Hosting environment" + env.EnvironmentName);  
    });  
}
```

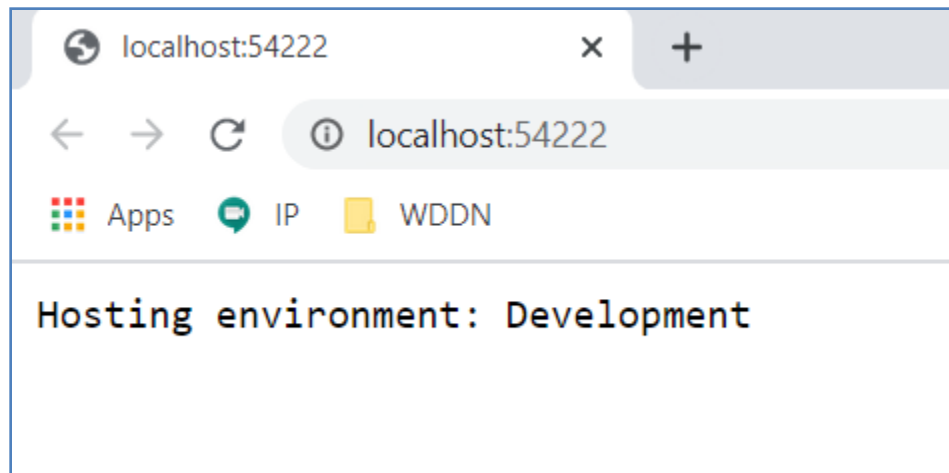


```
launchSettings.json  appsettings.Development.json  Microsoft.Common.C...entV
mstore.org/launchsettings

}
},
"profiles": {
  "IIS Express": {
    "commandName": "IISExpress",
    "launchBrowser": true,
    "environmentVariables": {
      //"ASPNETCORE_ENVIRONMENT": "Development",
      "MyKey": "Value of MyKey from launchSettings.json"
    }
  }
}
```



Why?



Useful methods of IHostingEnvironment service

- Use the following methods of IHostingEnvironment service to identify the environment in which our application is running.
 - IsDevelopment()
 - IsStaging()
 - IsProduction()
- Custom environment like UAT (User Acceptance Testing) or QA (Quality Assurance) environment.
 - env.IsEnvironment("UAT")

What is the difference between .NET
Core and .NET Framework?

What is Kestrel?

What is Startup.cs file in dot net core?

What is the ConfigureServices method
of the Startup.cs file?

What is the Configure method of the
Startup.cs file?

What is Middleware?

How to handle errors in middleware?

Where static files contains in asp.net
core application?

What is the difference between
`app.Run` and `app.Use` in asp.net core
middleware pipeline?