ASP.NET Core

TIME FOR CASE STUDY

User Story – Sprint 1

- As a **Customer** of Shopon, I should be allowed to view all products on my laptop, desktop or from mobile and from anywhere.

Thought

- Customer should be allowed to view all products on different devices.
- Customer can view these products from anywhere.





Solution

Solution

- Create new ASP.NET Core project

Knowledge Byte | Create ASP.NET Core Project

ASP.NET Core

- ASP.NET Core is a **cross-platform**, **High-performance**, **Open-Source framework** for building modern, cloud-based, internet-connected applications.
- ASP.NET Core is redesign of ASP.NET 4.x.
- Using ASP.NET Core we can
  - Build web apps and services, **Internet of Things** (IoT) apps, and **mobile** backends.
  - Use our favorite development tools on **Windows**, **macOS**, and **Linux**.
  - Deploy to the **cloud** or **on-premises**.
  - Run on **.NET Core**.

Get to know

Features of ASP.NET Core

Setting up ASP.NET Core Development

Creating ASP.NET Core Project

ASP.NET Core Project Structure

Next Step

Features of ASP.NET Core

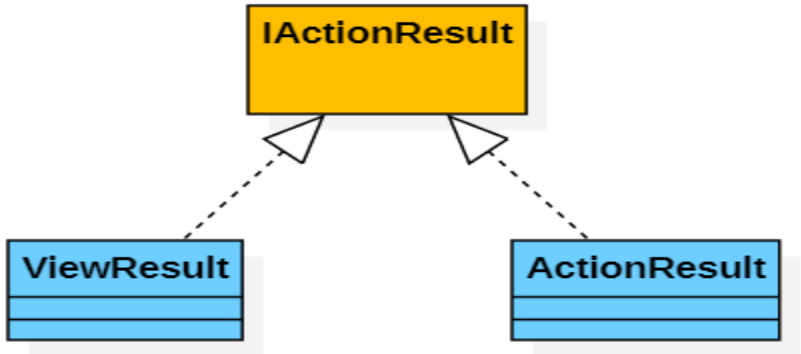
- Cross Platform
- Dependency Injection
- Open-Source
- One programming model for MVC and Web API
- Testability
- Modular

Cross Platform

- ASP.NET Core applications can be developed and run across different platforms like
  - Windows
  - macOS
  - Linux
- ASP.NET Core applications can be hosted on
  - IIS
  - Apache
  - Docker
  - Self-host in our own process
- For development point of view we can use
  - Visual Studio
  - Visual Studio Code
  - Sublime

One programming model for MVC and Web API

- With ASP.NET Core we use the same unified programming model to create MVC style web applications and ASP.NET Web API's.
- Both the MVC Controller class and the ASP.NET Web API Controller class inherit from the same **Controller** base class and returns **IActionResult**.



Dependency Injection

- ASP.NET Core supports the dependency injection (DI) software design pattern, which is a technique for achieving **Inversion of Control** (IoC) between classes and their dependencies.
- A *dependency* is an object that another object depends on.
- DI makes it easy to change the implementation without modifying the controller.
- Doesn't create an instance of Dependency, it's created by the DI container.

Testability

- Unit tests involve testing a part of an app in isolation from its infrastructure and dependencies. When unit testing controller logic, only the contents of a single action are tested, not the behavior of its dependencies or of the framework itself.
- With DI support and unified creating model for creating web application and Web API's, unit testing ASP.NET Core applications is easy.

Open-Source

- Open-Source and community focused
- ASP.NET Core is fully open source and is been actively developed by .NET team in collaboration with vast community of developers.
- To know more visit
  - https://dotnet.microsoft.com/en-us/platform/community
  - https://github.com/dotnet/aspnetcore

Modular

- ASP.NET Core provides modularity with Middleware components.
- Both the request and response pipelines are composed using the middleware components.
- Rich set of built-in middleware components are provided out of the box.
- We can create our own Custom Middleware components.

Setting up ASP.NET Core Development

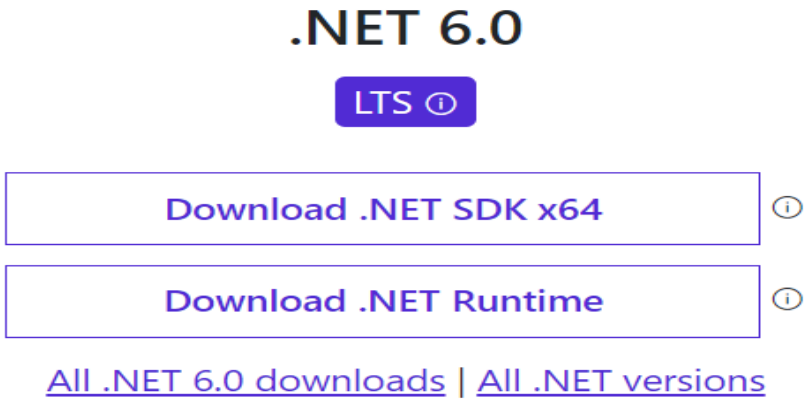
- Editor
  - Visual Studio
  - Visual Studio Code
  - Sublime
  - Atom
- .NET Core SDK(Software Development Kit)

Editor

- Visual Studio and Visual Studio Code
  - https://visualstudio.microsoft.com/
- Sublime
  - https://www.sublimetext.com/download
- Atom
  - https://atom.io/

.NET Core SDK

- We can download .NET Core SDK from
  - https://dotnet.microsoft.com/en-us/download



To list all the SDK's available in the system:

```
C:\>dotnet --list-sdks
5.0.202 [C:\Program Files\dotnet\sdk]
6.0.101 [C:\Program Files\dotnet\sdk]
```

Creating ASP.NET Core Project

- ASP.NET Core project can be created using
  - .NET CLI
  - Visual Studio (2019)

.NET CLI

- The .NET command-line interface (CLI) is a cross-platform tool chain for developing, building, running, and publishing .NET applications.
- The .NET CLI is included with the .NET SDK.
- To get help about CLI commands, open command prompt execute: `dotnet --help`
- To get .NET Project templates: `dotnet new --list`
- To create ASP.NET MVC project: `dotnet new web`

NOTE: If we don't specify name of the project, by default CLI will pick the name of the folder in which the command is executed and creates the project with the same **name**. To specify name, we must include –name attribute with **new** command.

```
dotnet new web --name=MyShopon
```

.NET CLI

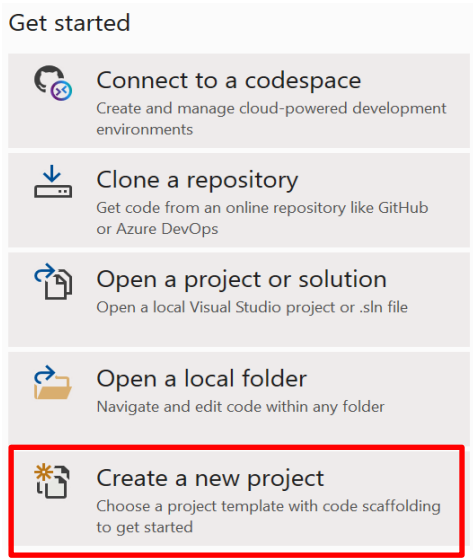
- Some useful commands
  - **dotnet build** – is used to build the application
  - **dotnet restore –** is used to restores the dependencies and tools of a project
  - **dotnet publish –** is used to publishes the application and its dependencies to a folder for deployment to a hosting system.
  - **dotnet run** – runs source code without any explicit compile or launch commands
  - **dotnet clean –** is used to clean the output of a project

.NET CLI

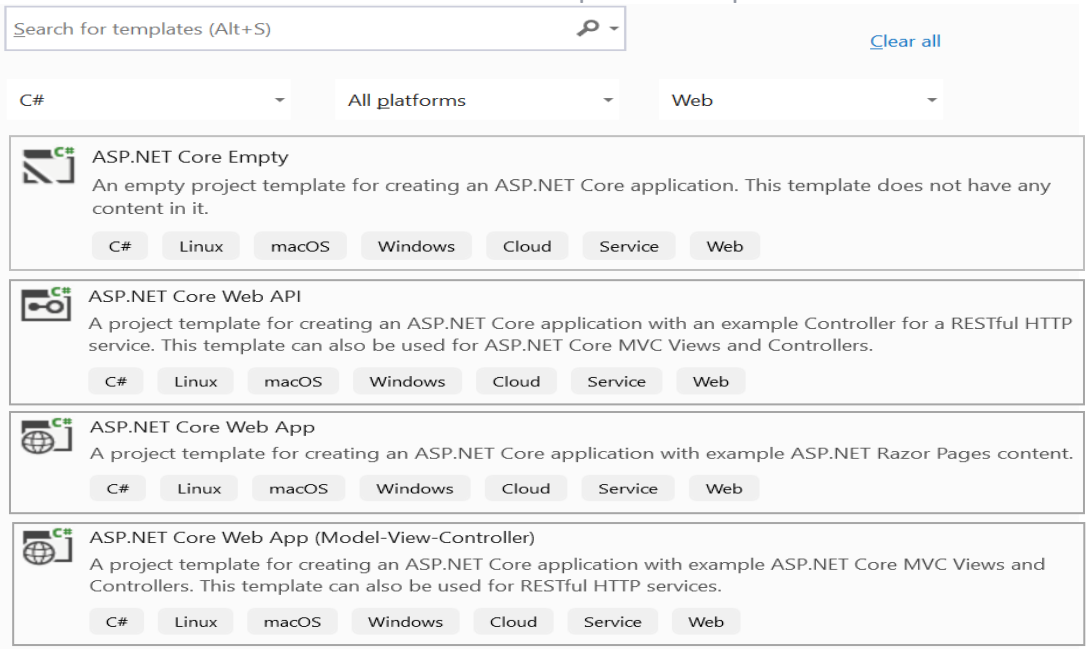
- Commands to open project created using CLI in VS
  - **dotnet new sln –n mySolution** – Create new solution with name mySolution
  - **dotnet new console –o myApp** – Create new console app with name myApp
  - **dotnet new classlib –o mylib1** – Create new class library with name mylib1
  - **dotnet sln mySolution.sln add myapp\myapp.csproj** – to add project to solution
  - **dotnet sln mySolution.sln add mylib1\mylib1.csproj** –solution-folder-mylibs – to add class library to project

Visual Studio (2019)

- Creating new project using Visual Studio 2019
  - Create a new project
  - Select project Template

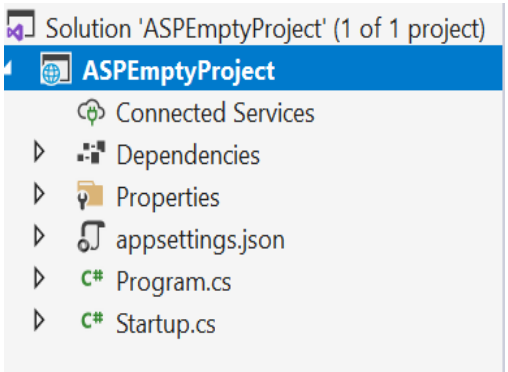


Click on respective template to know more



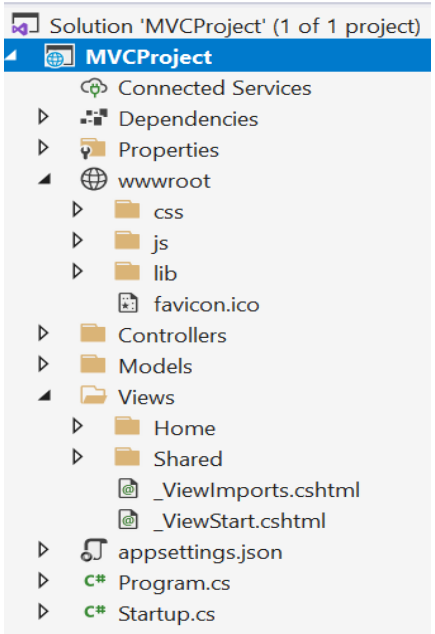
ASP.NET Core Empty Project

- ASP.NET Core Empty Project – This template does not contain much content. This has minimum content to display “Hello World” on the web browser. This will have minimum configuration which will be setup by default.



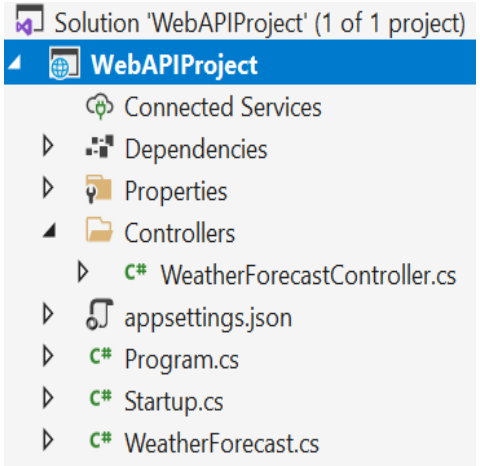
ASP.NET Core Web App(Model View Controller)

- This template has all the required contents to create any standard MVC project like Models, Views and Controllers. It also contains all required JavaScript, CSS files related to base code.



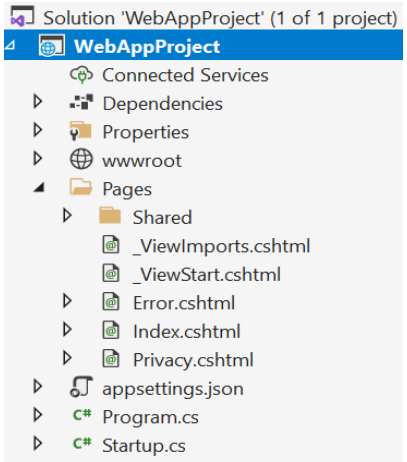
ASP.NET Core Web API

- This template is used to create **HTTP RESTfull** Service. This will not have user interface so all website specified contents like JavaScript files, CSS files, View files, layout files will not be part of this template. This template creates the Controllers folder.

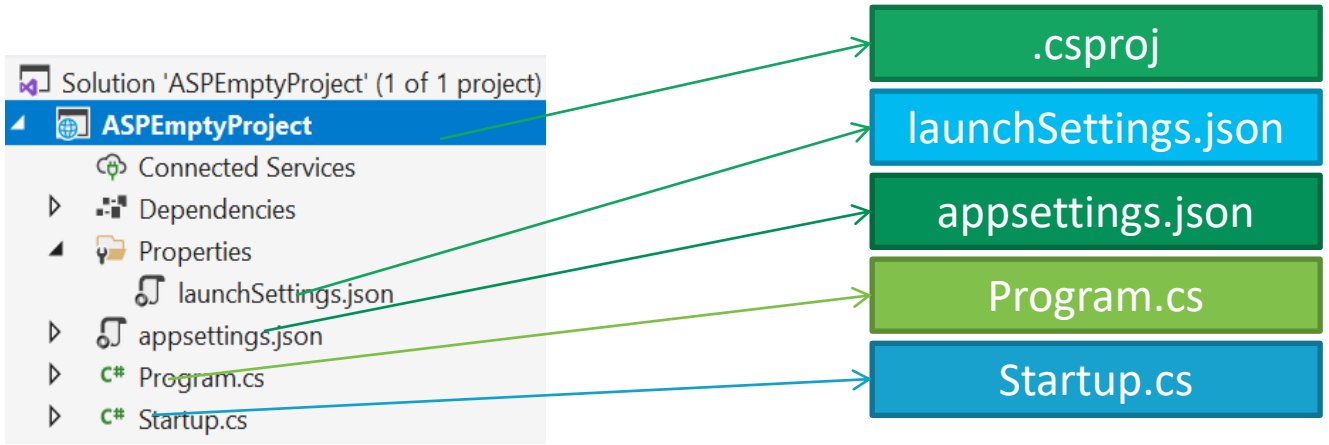


ASP.NET Core Web App

- This template uses the **Razor Pages Framework** for building Web Applications. We use Razor pages only when we are building simple web application which is not having complex business logics. We can consider this as a slimmer version of MVC framework.

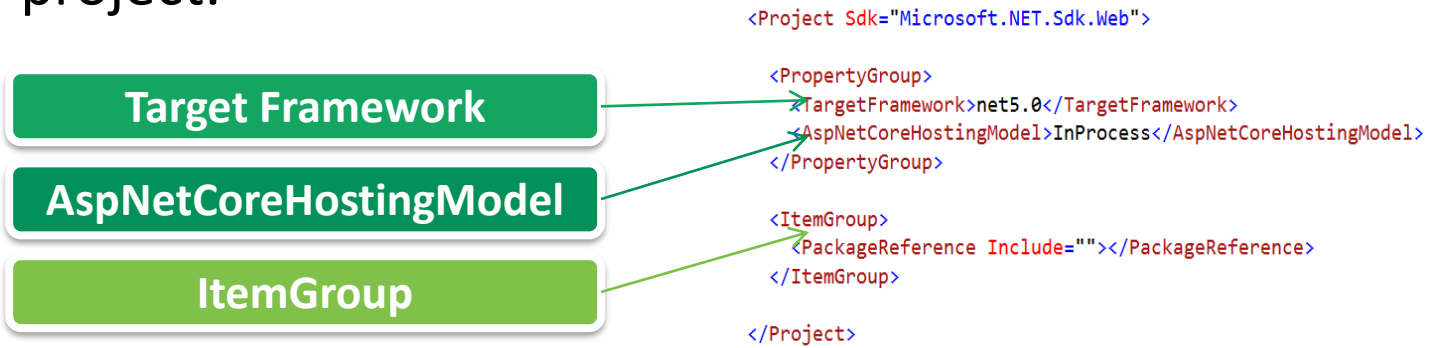


ASP.NET Core Project Structure



.csproj

- **.csproj** or **.vbproj** depending on the programming language used.
- To modify project file, no need to unload the project.
- File or folder references are no longer included in the project file. Whenever there is a new file or the folder is added to the project, it reflects immediately.
- The File System determines what files and folders belong to the project.



Target Framework

- The Target Framework specifies the target framework for the application.
- To specify a target framework we use **Target Framework Moniker(TFM).**
- Naming of TFM

```
<PropertyGroup>
  <TargetFramework>net5.0</TargetFramework>
</PropertyGroup>
```

Name	Abbreviation	Name
.NET Framework	net	net451 net472
.NET Core	netcoreapp	netcoreappapp1.0 netcoreapp2.2 netcoreapp3.1 net5.0

AspNetCoreHostingModel

- AspNetCoreHostingModel specifies how the application should be hosted.
- It can be **InProcess** or **OutOfProcess.**
- **InProcess** hosts the app inside of the IIS worker process(w3sp.exe).
- **OutOfProcess** is the *default hosting process*.
- OutOfProcess hosting model forward web requests to a backend ASP.NET Core app running the **Kestrel server**.

ItemGroup

- **ItemGroup** has **PackageReference** as one of the element in it.
- **PackageReference** is used to include a reference to the Nuget package that is installed for the application.
- PackageReference include **Metapackage**(Microsoft.AspNetCore.App)
- A Metapackage has no content of its own. It just contains a list of

NOTE: A PackageReference to **Microsoft.AspNetCore.App** is not necessary when targeting .NET Core 3.0 or higher. If Microsoft.NET.Sdk.Web is used, the shared framework will be referenced automatically. Otherwise, the PackageReference should be replaced with a FrameworkReference.

```
<ItemGroup>
  <PackageReference Include="Microsoft.AspNetCore.App"></PackageReference>
  <PackageReference Include="Microsoft.AspNetCore.Razor.Design"></PackageReference>
</ItemGroup>
```

Program.cs

- Program.cs has **Main** method.
- ASP.NET Core application initially starts as a Console application and the Main() is the entry point into the application.
- The **Main()** configures ASP.NET Core and starts it and at that point it becomes an ASP.NET Core web application.

- **CreateHostBuilder()** takes **args** as parameter passing it to **Host.CreateDefaultBuilder** method and returns **IHostBuilder** reference.

```
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>();
            });
}
```

Startup.cs

- This is one of the file which is generated by the ASP.NET Core template designer.
- This class has
  - ConfigureServices – is used to configure services which are required for our application.
  - Configure – is used to configure our request processing pipeline.

```
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)...
}
```

The Configure()

- The Configure method is a place where we can configure application request pipeline for our application using **IApplicationBuilder** instance that is provided by the built-in IoC container.
- ASP.NET Core introduced the **middleware** components to define a request pipeline, which will be executed on every request. We include only those middleware components which are required by our application and thus increase the performance of our application.

Middleware

- Middleware is software that's assembled into an app pipeline to handle requests and responses. Each component:
  - Chooses whether to pass the request to the next component in the pipeline.
  - Can perform work before and after the next component in the pipeline.
- Request delegates are used to build the request pipeline. The request delegates handle each HTTP request.
- Request delegates are configured using Run, Map, and Use extension methods.

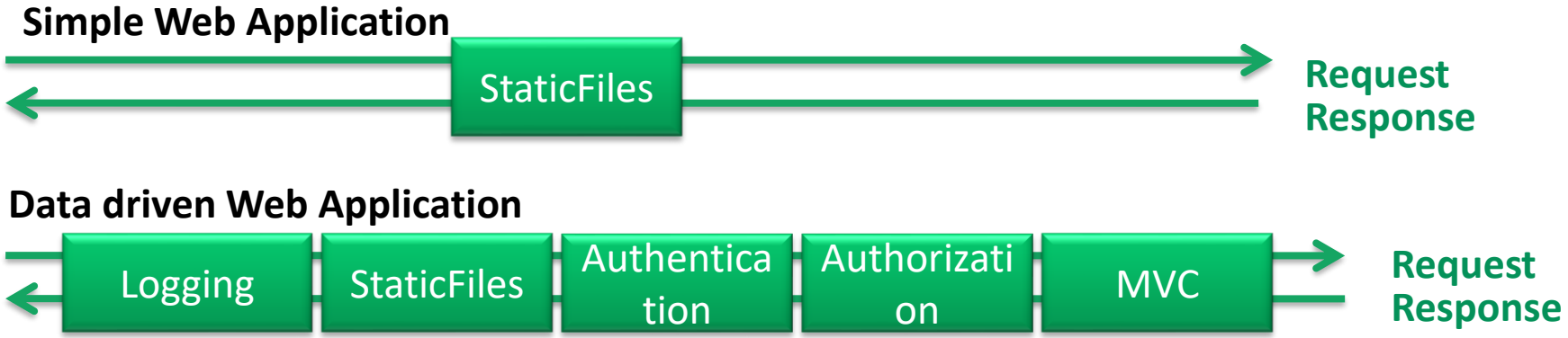
Processing of Middleware

Processing of Middleware

- Middleware has access to both Request and Response.
- An individual request delegate can be specified *in-line* as an anonymous method (called **in-line middleware**), or it can be defined in a *reusable class*. These reusable classes and in-line anonymous methods are ***middleware***, also called ***middleware components***.
- Each middleware component in the request pipeline is responsible for invoking the next component in the pipeline or *short-circuiting the pipeline*. When a middleware short-circuits, it's called a ***terminal middleware*** because it prevents further middleware from processing the request.
- May process the outgoing Response
- Exception-handling delegates should be called early in the pipeline, so they can catch exceptions that occur in later stages of the pipeline.
- Middleware components are executed in the order they are added.

Processing of Middleware

- This middleware are executed in the order in which they are placed. If not placed properly, application may not work as expected.
- These middleware components are available as Nuget packages.
- We always have control on configuring the middleware based on our request.



Run

- **RunExtensions.Run(IApplicationBuilder, RequestDelegate) Method**
  - Adds a **terminal middleware** delegate to the application's request pipeline.
  - The Run method will not execute the next middleware, to process the next middleware, we have to use **Use()**.

Example1

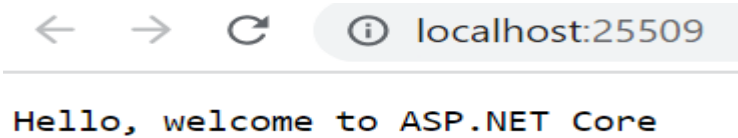
Example2

Run Example1

Startup.cs

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

    app.Run(async (context) =>
    {
        await context.Response.WriteAsync(
            "Hello, welcome to ASP.NET Core");
    });
}
```

Browser Output



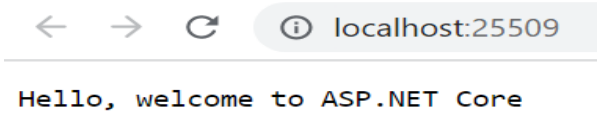
Run Example2

Startup.cs

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

    app.Run(async (context) =>
    {
        await context.Response.WriteAsync(
            "Hello, welcome to ASP.NET Core");
    });
    app.Run(async (context) =>
    {
        await context.Response.WriteAsync(
            "Next message to print");
    });
}
```

This will not get called

Browser Output



Use() Method

UseExtensions.Use Method

- **Use()** method is used to invoke next middleware.
- It has 2 overloads

Use(IApplicationBuilder, Func<HttpContext,RequestDelegate,Task>)	Adds a middleware delegate defined in-line to the application's request pipeline. If you aren't calling the next function, use Run(IApplicationBuilder, RequestDelegate) instead.
Use(IApplicationBuilder, Func<HttpContext,Func<Task>,Task>)	Adds a middleware delegate defined in-line to the application's request pipeline. If you aren't calling the next function, use Run(IApplicationBuilder, RequestDelegate) instead. Prefer using Use(IApplicationBuilder, Func<HttpContext,RequestDelegate,Task>) for better performance as shown below:

Example1

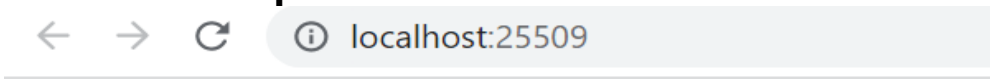
Use() – Example

Startup.cs

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

    app.Use(async (context, next) =>
    {
        await context.Response.WriteAsync(
            "Hello, welcome to ASP.NET Core. ");

        await next.Invoke();
    });
    app.Run(async (context) =>
    {
        await context.Response.WriteAsync(
            "Next message to print");
    });
}
```

next()

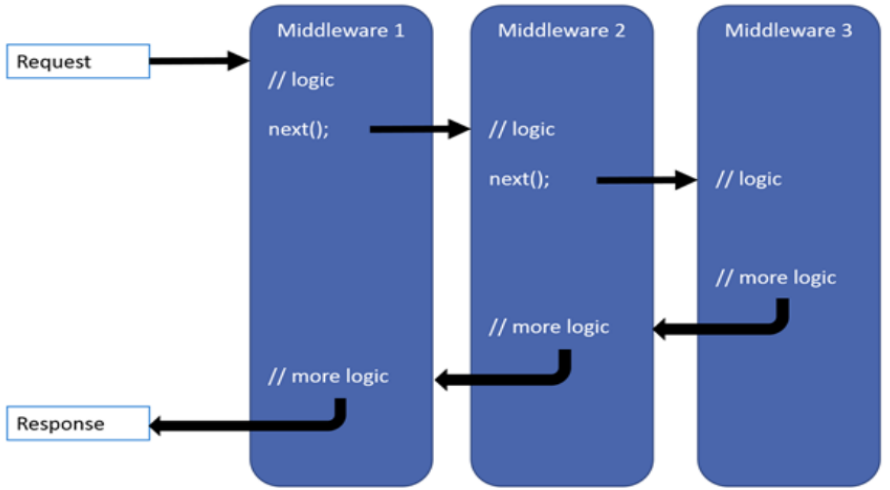
Browser Output



next() Method

- next will return **Func<Task>**. This will generate delegate which will hold next middleware. The **next.Invoke()** method will invokes the next middleware.
- We can also use `await next();` invoke the next middleware too.

Example



next() Example

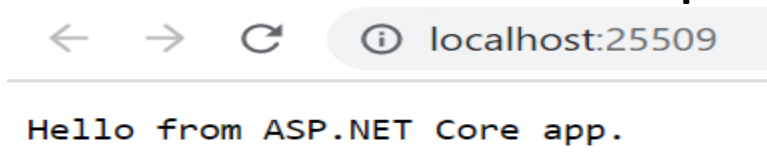
Startup.cs

```
public void Configure(IApplicationBuilder app,
    IWebHostEnvironment env,
    ILogger<Startup> logger)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    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(
            "Hello from ASP.NET Core app.");
        logger.LogInformation("MW3: Request handled and response generated.");
    });
}
```

Debug window

```
ASPEmptyProject.Startup: Information: MW1: Incoming request.
ASPEmptyProject.Startup: Information: MW2: Incoming request.
'iisexpress.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files
'iisexpress.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files
ASPEmptyProject.Startup: Information: MW3: Request handled ar
ASPEmptyProject.Startup: Information: MW2: Outgoing response.
ASPEmptyProject.Startup: Information: MW1: Outgoing response.
'iisexpress.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files
'iisexpress.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files
```

Browser Output



NOTE: Run application in **DEBUG** mode

Map() method

MapExtensions.Map Method

- Runs middleware if path requested by user equals path provided in parameter.

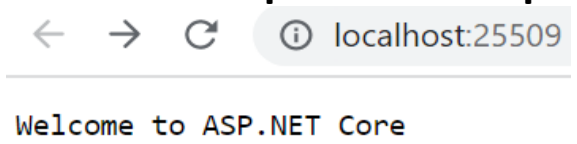
Example

Map() - Example

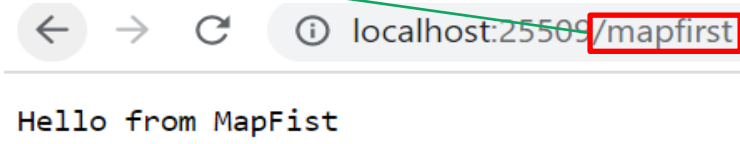
Startup.cs

```
public void Configure(IApplicationBuilder app,
    IWebHostEnvironment env,
    ILogger<Startup> logger)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.Map("/mapFirst", MapFirst);
    app.Run(async (context) =>
    {
        await context.Response.WriteAsync("Welcome to ASP.NET Core");
    });
}

1 reference
private void MapFirst(IApplicationBuilder obj)
{
    obj.Use(async(context, next) =>
    {
        await context.Response.WriteAsync("Hello from MapFist");
    });
}
```

Browser Output with out parameter



Browser Output with parameter



Host.CreateDefaultBuilder

- **CreateDefaultBuilder** performs several tasks. Some of the tasks are
  - Setting up the web server
  - Loading the host and application configuration from various configuration sources and
  - Configuring logging

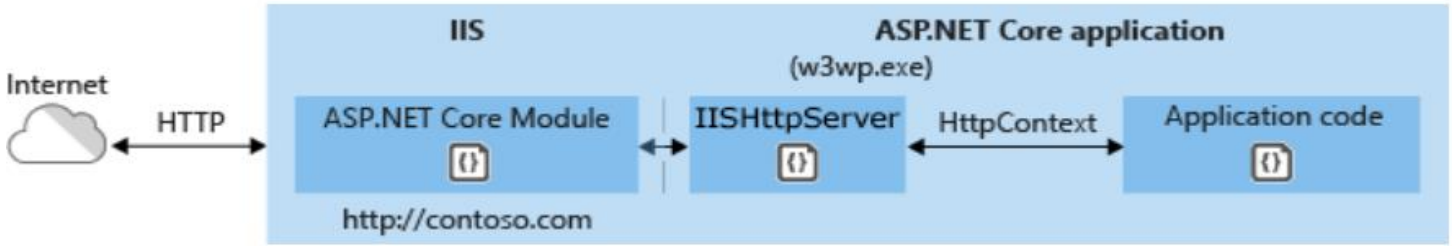
Setting up the web server

- An ASP.NET core application can be hosted using
  - Hosting an ASP.NET Core app inside of the IIS worker process (w3wp.exe), called the **in-process hosting model**.
  - Forwarding web requests to a backend ASP.NET Core app running the Kestrel server, called the **out-of-process hosting model**.

InProcess v/s OutOfProcess

InProcess Cont…

- In-process hosting runs an ASP.NET Core app in the same process as its IIS worker process.
- In-process hosting provides improved performance over out-of-process hosting because requests aren't proxied over the loopback adapter, a network interface that returns outgoing network traffic back to the same machine.



InProcess Cont…

- Since ASP.NET Core 3.0, in-process hosting has been ***enabled by default*** for all app deployed to IIS.
- To explicitly configure an app for in-process hosting, set the value of the <AspNetCoreHostingModel> property to InProcess in the project file (.csproj):

```
<PropertyGroup>
  <AspNetCoreHostingModel>InProcess</AspNetCoreHostingModel>
</PropertyGroup>
```

To get the process name executing the app

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

Example

InProcess - Example

Startup.cs

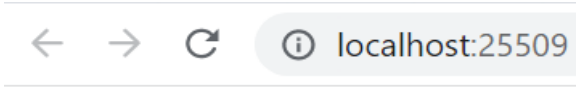
```
public void Configure(IApplicationBuilder app,
    IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.UseRouting();

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

Executing mode

ASPCoreApp.csproj

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

  <PropertyGroup>
    <TargetFramework>net5.0</TargetFramework>
    <AspNetCoreHostingModel>InProcess</AspNetCoreHostingModel>
  </PropertyGroup>

</Project>
```

Browser Output

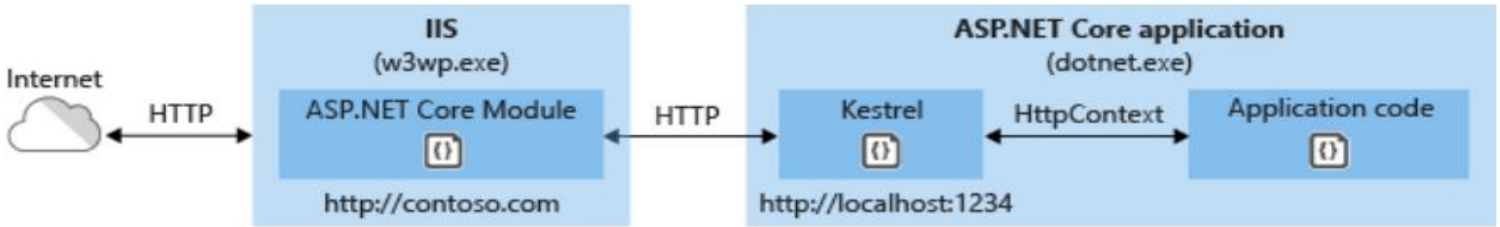


iisexpress

NOTE: By default, Visual studio uses IISExpress to host and run our application.

OutOfProcess Cont…

- There are 2 server which will be included in OutOfProcess hosting
  - Internal – internal web server is **Kestrel**
  - External – external web server can be IIS, Nginx or Apache based on the OS.



OutOfProcess Cont…

- Configuring OutOfProcessing hosing
  - Change <AspNetCoreHostingModel>**OutOfProcess**</AspNetCoreHostingModel>

Example

OutOfProcess - Example

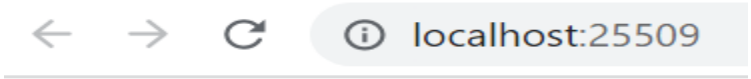
Startup.cs

```
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);
        });
    });
}
```

ASPCoreApp.csproj

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

  <PropertyGroup>
    <TargetFramework>net5.0</TargetFramework>
    <AspNetCoreHostingModel>OutOfProcess</AspNetCoreHostingModel>
  </PropertyGroup>

</Project>
```

Browser Output



ASPEmptyProject

Executing mode

Kestrel

- Kestrel is cross-platform web server for ASP.NET Core. It is including as default server in ASP.NET Core.
- Kestrel can be used by itself as an edge server which can handle HTTP service request from the client.
- The process used to host the app is **dotnet.exe.**
- Running app in Kestrel will use **OutOfProcess** by default. It ignores the setting mentioned in project file.
- When **run** command is executed from .NET Core CLI, Kestrel server

Example

Kestrel - Example

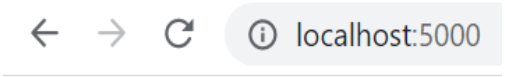
Startup.cs

```
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);
        });
    });
}
```

ASPCoreApp.csproj

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

  <PropertyGroup>
    <TargetFramework>net5.0</TargetFramework>
    <AspNetCoreHostingModel>InProcess</AspNetCoreHostingModel>
  </PropertyGroup>

</Project>
```

Browser Output



ASPEmptyProject

CLI Commands

```
dotnet clean
dotnet build
dotnet run
```

NOTE: The output is the process name which is serving the request.

InProcess v/s OutOfProcess

In Process	Out of Process
Process name is w3wp.exe or iisexpress.exe	Process name is dotnet.exe
Only one web server	Two web server - Internal – Kestrel server - External – Reverse proxy server which can be IIS, Apache or Nginx server
Better for performance	Penalty of proxying requests between internal and external web server

Can we run ASP.NET Core without Kestrel server?

launchSettings.json

- This is the configuration file. The configurations specified in this file is used while running .NET Core application either by Visual Studio or by .NET Core CLI.
- This file is required in local development machine.

launchSettings.json Cont…

- This file is used to store all *application configuration information* which will be used by ASP.NET Core application while publishing and deploying our app.
- We can also use different appsetting files based on environment like
  - appsettings.stagging.json
  - appsettings.development.json
- This file consists of profilers which will be used while running our application.

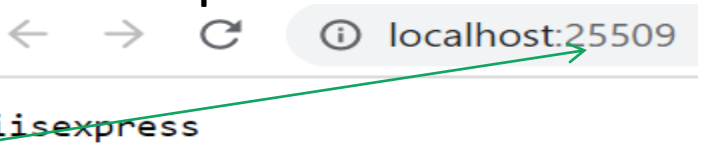
```
{
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:25509",
      "sslPort": 0
    }
  },
  "profiles": {
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "ASPEmptyProject": {
      "commandName": "Project",
      "dotnetRunMessages": "true",
      "launchBrowser": true,
      "applicationUrl": "http://localhost:5000",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

launchSettings.json Cont…

launchSettings.json

```
{
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:25509",
      "sslPort": 0
    }
  },
  "profiles": {
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "ASPEmptyProject": {
      "commandName": "Project",
      "dotnetRunMessages": "true",
      "launchBrowser": true,
      "applicationUrl": "http://localhost:5000",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

Browser Output



Startup.cs

```
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);
        });
    });
}
```

NOTE: Based on the profiles, we can run different configuration.

launchSettings.json Cont…

- Profiles - **launchSettings.json** has different profiles like "IIS Express“ and "ASPEmptyProject“. When we run application using IIS Express, profile related to IIS Express gets executed and when we run the same application using CLI, [Project named] profile settings gets executed.

Example

launchSettings.json - Example

launchSettings.json

```
{
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:25509",
      "sslPort": 0
    }
  },
  "profiles": {
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "ASPEmptyProject": {
      "commandName": "Project",
      "dotnetRunMessages": "true",
      "launchBrowser": true,
      "applicationUrl": "http://localhost:5000",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

Browser Output

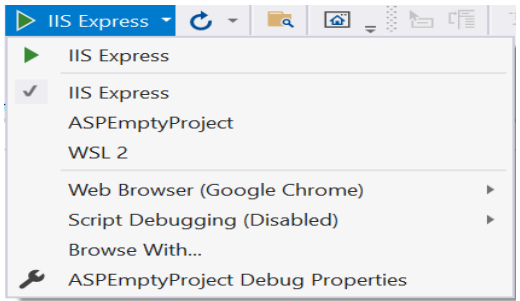
iisexpress

CLI Command to run the application

```
dotnet run
```

Browser Output

ASPEmptyProject

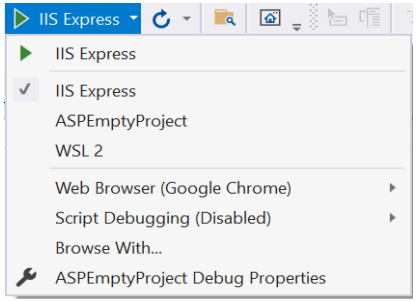


NOTE: By default when we run the application using VS, it will pick IIS Express profile, which we can change.

launchSettings.json Cont…

Execution model

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



The File System

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

  <PropertyGroup>
    <TargetFramework>net5.0</TargetFramework>
    <AspNetCoreHostingModel>InProcess</AspNetCoreHostingModel>
  </PropertyGroup>

</Project>
```

ASP.NET Configurations Sources

- **Configuration Sources in ASP.NET Core**
  - **Files(appsettings.json, appsettings.{Environment}.json)**
  - **User Secrets**
  - **Environment variables**
  - **Command-line arguments**

NOTE: All the configuration information will be stored in **Key** and **Value** pair. If the same key is used in different configuration settings, it will get override in the order given above. Priority flows from bottom to top.

appsettings.json

- **appsettings.json** file stores application configuration information like DB connection string.
- Configuration information in this file is stored in the form of **key** and **value** pair.
- ASP.NET Core provides **IConfiguration** Service which will allows us to access all these configuration information.

Example

appsettings.json - Example

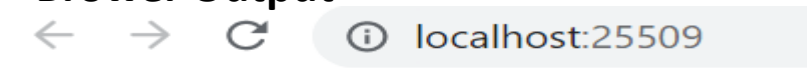
Startup.cs

```
using Microsoft.Extensions.Configuration;

public class Startup
{
    private readonly IConfiguration _config;
    public Startup(IConfiguration configuration)
    {
        _config = configuration;
    }

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

appsettings.json

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "MyKey": "Value of MyKey from config file"
}
```

Brower Output



Value of MyKey from config file

Environment variables

- This configuration inform is set in “launchSettings.json” file.
- The configuration is set using **Key** and **Value** pair.
- To read the configuration, we will use IConfiguration service.

Example

Environment variables - Example

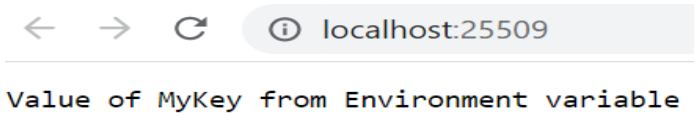
Startup.cs

```
using Microsoft.Extensions.Configuration;

public class Startup
{
    private readonly IConfiguration _config;
    public Startup(IConfiguration configuration)
    {
        _config = configuration;
    }

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

launchSettings.json

```
{
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:25509",
      "sslPort": 0
    }
  },
  "profiles": {
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development",
        "MyKey": "Value of MyKey from Environment variable"
      }
    },
    "ASPEmptyProject": {
      "commandName": "Project",
      "dotnetRunMessages": "true",
      "launchBrowser": true,
      "applicationUrl": "http://localhost:5000",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

Brower Output



Command-line arguments

- We can pass command line argument while executing the application.

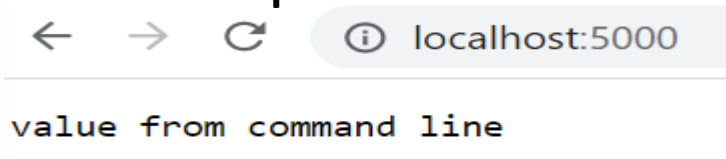
Example

Command-line arguments - Example

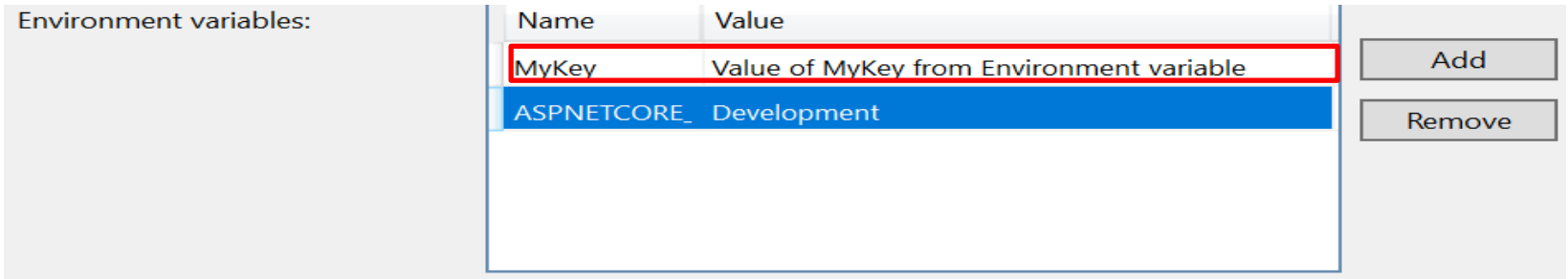
- Running the CLI command

```
dotnet run MyKey="value from command line"
```

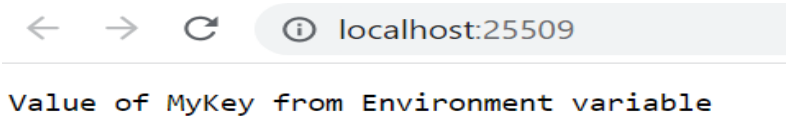
Brower Output



- From Project Property→Debug window



Brower Output



Next Step

Recap

Useful links

Thank you

Recap

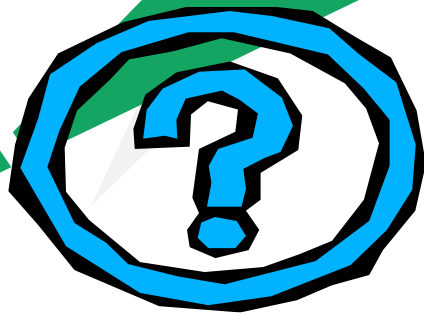
- Till now we have understood
  - Understanding of .NET Core
  - .NET Core features
  - Creating ASP.NET Core app
    - Using CLI
    - Using Visual Studio
  - .NET Core project structure
  - .NET Core file structure
  - .NET Core deployment

Useful Links

- https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-6.0
- https://docs.microsoft.com/en-us/aspnet/core/fundamentals/?view=aspnetcore-6.0&tabs=windowshttps://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-6.0
- https://docs.microsoft.com/en-us/aspnet/core/fundamentals/host/generic-host?view=aspnetcore-6.0
- https://dotnet.microsoft.com/en-us/platform/community
- https://github.com/dotnet/aspnetcore
- https://dotnet.microsoft.com/en-us/download
- https://docs.microsoft.com/en-us/dotnet/core/tools/
- https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/aspnet-core-module?view=aspnetcore-6.0#:~:text=ASP.NET%20Core%20apps%20default,used%20instead%20of%20Kestrel%20server.
- https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/iis/in-process-hosting?view=aspnetcore-6.0

Thank YOU

Any Questions?



US – CORPORATE HEADQUARTERS

1248 Reamwood Avenue,
Sunnyvale, CA 94089
Phone: (408) 743 4400

343 Thornall St 720
Edison, NJ 08837
Phone: (732) 395 6900

UK

57 Rathbone Place,
4th Floor, Holden House,
London, W1T 1JU , UK

89 Worship Street Shoreditch,
London EC2A 2BF, UK
Phone: (44) 2079 938 955

INDIA

Mumbai
4th Floor, Nomura
Powai , Mumbai 400 076
Phone: +91 (22) 3051 1000

Pune
5th floor, Amar Paradigm Baner Road
Baner, Pune 411 045
Phone: +91 (20) 6604 6000

Bangalore
4th Floor, Kabra Excelsior,
80 Feet Main Road, Koramangala 1st Block,
Bengaluru (Bangalore) 560034
Phone: +91 (80) 4666 1666

www.xoriant.com