MCSE 541:Web Computing and Mining

ASP.NET Core-Middleware and Static files

Prof. Dr. Shamim Akhter
shamimakhter@iubat.edu

Extension Method

- Additional method to be injected into a class/interface
  - without modifying, deriving or recompiling the original.
  - own custom class, .NET framework classes, or third party classes or interfaces.

```
int i = 10;
bool result = i.IsGreaterThan(100); //returns false
```

Define an Extension Method

```
namespace ExtensionMethods {
    public static class IntExtensions {
        public static bool IsGreaterThan(this int i, int value) {
            return i > value;
        }
    }
}
```

Binding parameter to bind these methods wit
int i class.

An extension method can take extra parameters, in addition to an instance of the type that it is extending.

```
using ExtensionMethods;
class Program {
    static void Main(string[] args) {
        int i = 10;
        bool result = i.IsGreaterThan(100);
        Console.WriteLine(result);
    }
}
```

Class Example with Extension Method

```
Using System;
namespace ExtensionMethod {
// Here Geek class contains three methods. Now we want to add two more new methods in it without re-compiling this class
class Geek {
              public void M1()
              {
                             Console.WriteLine("Method Name: M1");
               }
               public void M2()
              {
                             Console.WriteLine("Method Name: M2");
              }
              public void M3()
              {
                            Console.WriteLine("Method Name: M3");
              }
     }
}
```

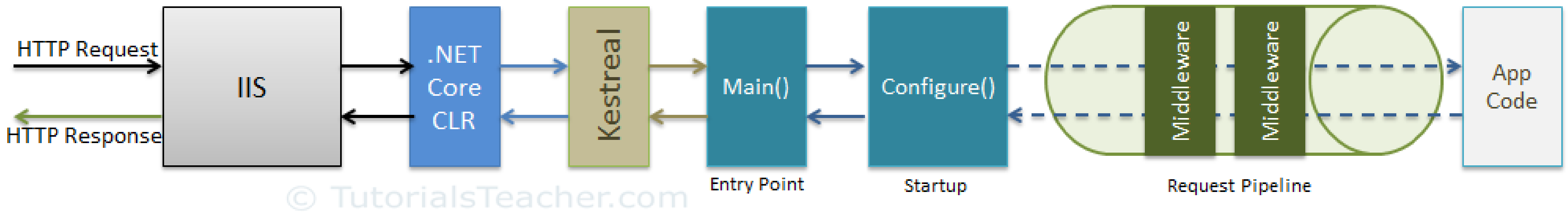
Defining Extension Methods

```
using System;

namespace ExtensionMethod {

// This class contains M4 and M5 methods. Which we want to add in
//Geek class. NewMethodClass is a static class
static class NewMethodClass {

    public static void M4(this Geek g)
    {
        Console.WriteLine("Method Name: M4");
    }

    public static void M5(this Geek g, string str)
    {
        Console.WriteLine(str);
    }
}

}
```

Call Extension Method

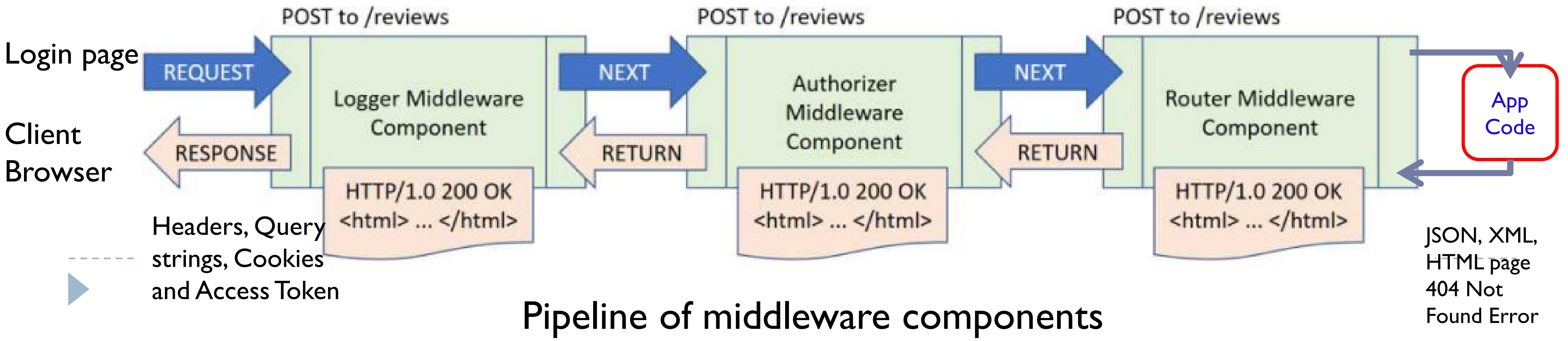
```
using System;

namespace ExtensionMethod {

public class GFG {

    // Main Method
    public static void Main(string[] args)
    {
        Geek g = new Geek();
        g.M1();
        g.M2();
        g.M3();
        g.M4();
        g.M5("Method Name: M5");
    }
}
}
```

ASP.NET Core Request Processing



Middleware

- A new concept in ASP.NET Core
- A component (class) which is executed on every http request in ASP.NET Core application.
  - How the application behaves if there is an error.
- In the classic ASP.NET,
  - HttpHandlers and HttpModules were part of the request pipeline.
- Middleware is similar to HttpHandlers and HttpModules
  - performs user authentication and authorization.


Login page
Client Browser
Headers, Query strings, Cookies and Access Token
POST to /reviews
REQUEST
RESPONSE
Logger Middleware Component
HTTP/1.0 200 OK <html> ... </html>
POST to /reviews
NEXT
RETURN
Authorizer Middleware Component
HTTP/1.0 200 OK <html> ... </html>
POST to /reviews
NEXT
RETURN
Router Middleware Component
HTTP/1.0 200 OK <html> ... </html>
App Code
JSON, XML, HTML page 404 Not Found Error

Pipeline of middleware components

Configure Middleware

- Middleware is configured into the Startup class's Configure method where an IApplicationBuilder interface instance is injected.
- IApplicationBUilder Interface, which is used to defines a class that provides the mechanisms to configure an application's request pipeline.

```
public class Startup {
   public Startup() { }
   public void Configure(IApplicationBuilder app, IHostingEnvironment env,
                         ILoggerFactory loggerFactory)
   { //configure middleware using IApplicationBuilder here..
       app.Run(async (context) => { await context.Response.WriteAsync("Hello World!");
                                   });
          // other code removed for clarity..
    }
}
```

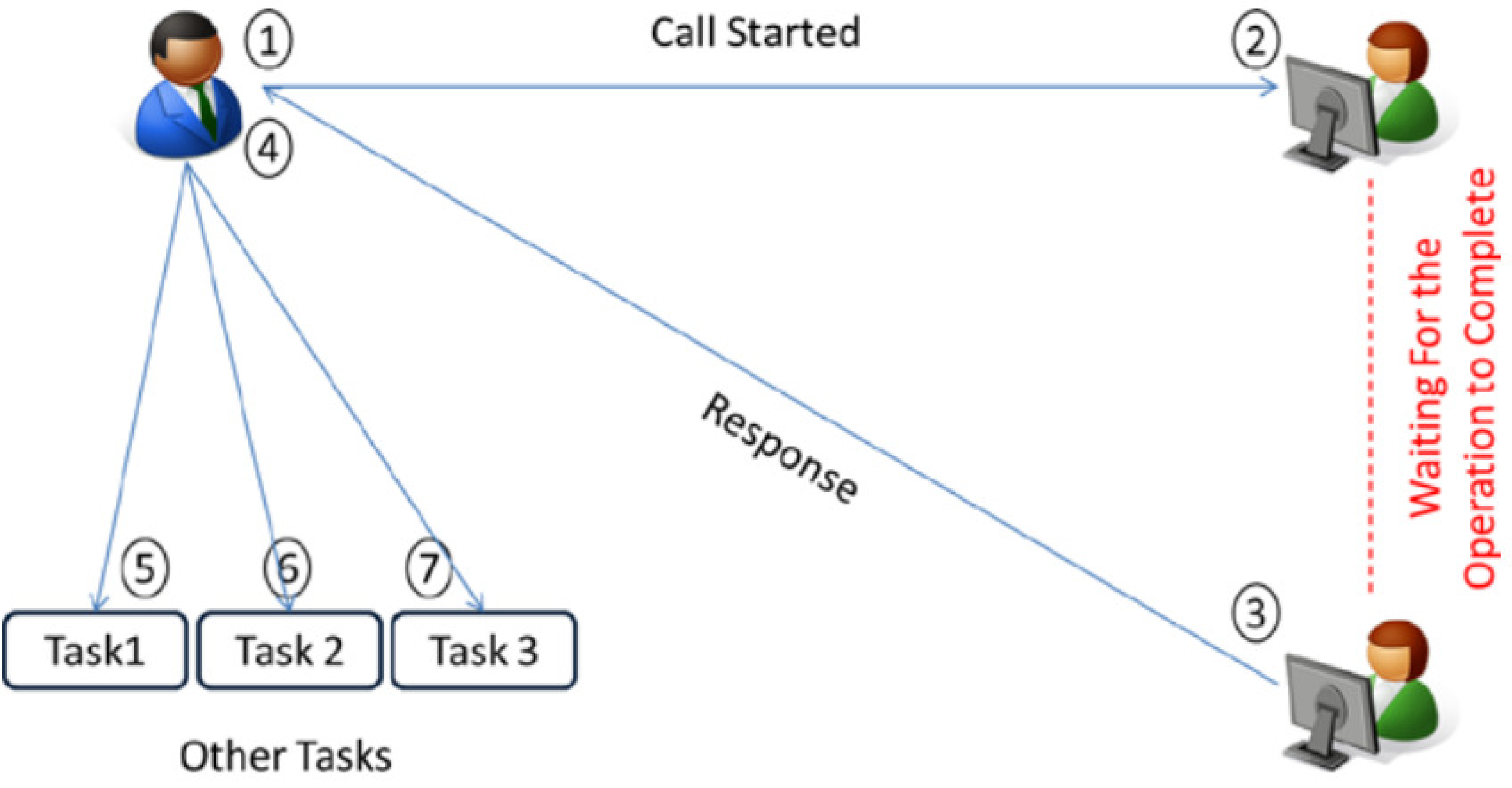
Run() is an extension method on IApplicationBuilder instance
which adds a terminal middleware to the application's request pipeline.

The above configured middleware returns a response with a string
"Hello World!" for each request.

Synchronous vs Asynchronous

Assume that a few days ago, after I bought a product from Company A, I began having a problem with it. I called the company's support center to explain and sort out the problem. After listening to my explanation, the customer service representative asked me to wait a few minutes. While he tried to solve the problem, I was left hanging on the telephone. The important part here is that I couldn't do anything else until the representative got back to me. Any other tasks I needed to perform were, like me, left on hold while I waited for my call to end.

My situation in this scenario can be related to **synchronous** processing of a long-running operation (Figure 2-1).



Let's construct another scenario. This time I bought a product from Company B; again, there was a problem. I called Company B's support center and explained the problem to a customer service representative. This time, however, since the representative said she would call me back as soon as the problem got sorted out, I could hang up the phone. This allowed me to see to other tasks while Company B's people worked on my problem. Later, the representative called me back and informed me of the problem's resolution. This scenario resembles how an **asynchronous** operation works (see Figure 2-2).

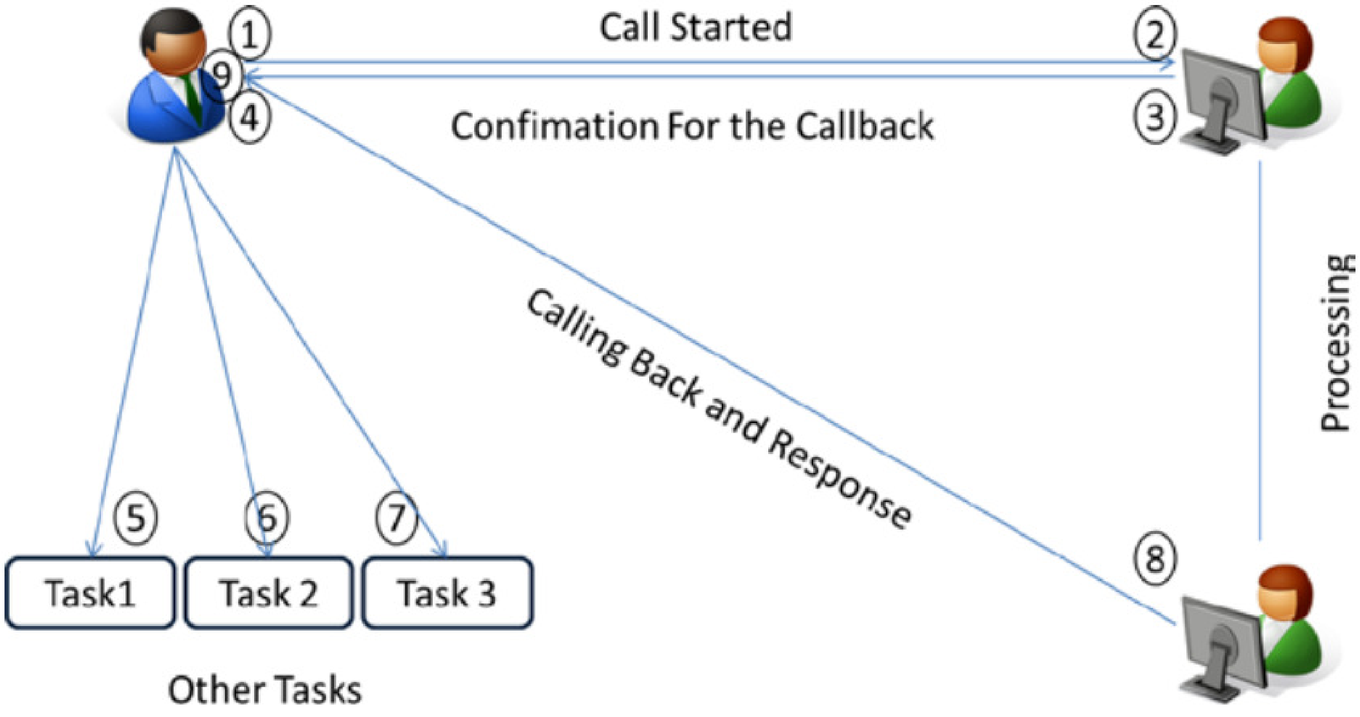


Figure 2-2. *An asynchronous phone call between me and Company B's customer service representative*

Run Method Signature

public static void Run(this IApplicationBuilder app, RequestDelegate handler)

Delegate Signature

Request delegates are used to build the request pipeline.
The request delegates handle each HTTP request.

public delegate Task RequestDelegate(HttpContext context);

```
public class Startup {
    public Startup() { }
    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        app.Run(MyMiddleware);
    }

    private Task MyMiddleware(HttpContext context)
    {
        return context.Response.WriteAsync("Hello World! ");
    }
}
```

MyMiddleware function is not asynchronous
Thus will block the thread till the time it completes the execution.
So, make it asynchronous by using async and await to improve performance and scalability.

Asynchronous MyMiddleware

```
private async Task MyMiddleware(HttpContext context)
{
    await context.Response.WriteAsync("Hello World! ");
}
```

- The await operator suspends evaluation of the enclosing async method until the asynchronous operation represented by its operand completes.
- When the asynchronous operation completes, the await operator returns the result of the operation, if any.

Run method calls with Delegate

```
public class Startup {
    public Startup() { }
    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        Microsoft.AspNetCore.Http.RequestDelegate Rd = MyMiddleware;
             //public delegate Task RequestDelegate(HttpContext context);
         app.Run(Rd);
    }

    private Task MyMiddleware(HttpContext context)
    {
       return context.Response.WriteAsync("Hello World! ");
    }
}
```

```
Microsoft.AspNetCore.Http.RequestDelegate Rd = (HttpContext context)
=>  context.Response.WriteAsync("Hello World10! ");
App.Run(Rd);
```

```
app.Run((HttpContext context) => context.Response.WriteAsync("Hello World10! "));
```

Configure Multiple Middlewares

There will be multiple middleware components in ASP.NET Core application which will be executed sequentially.

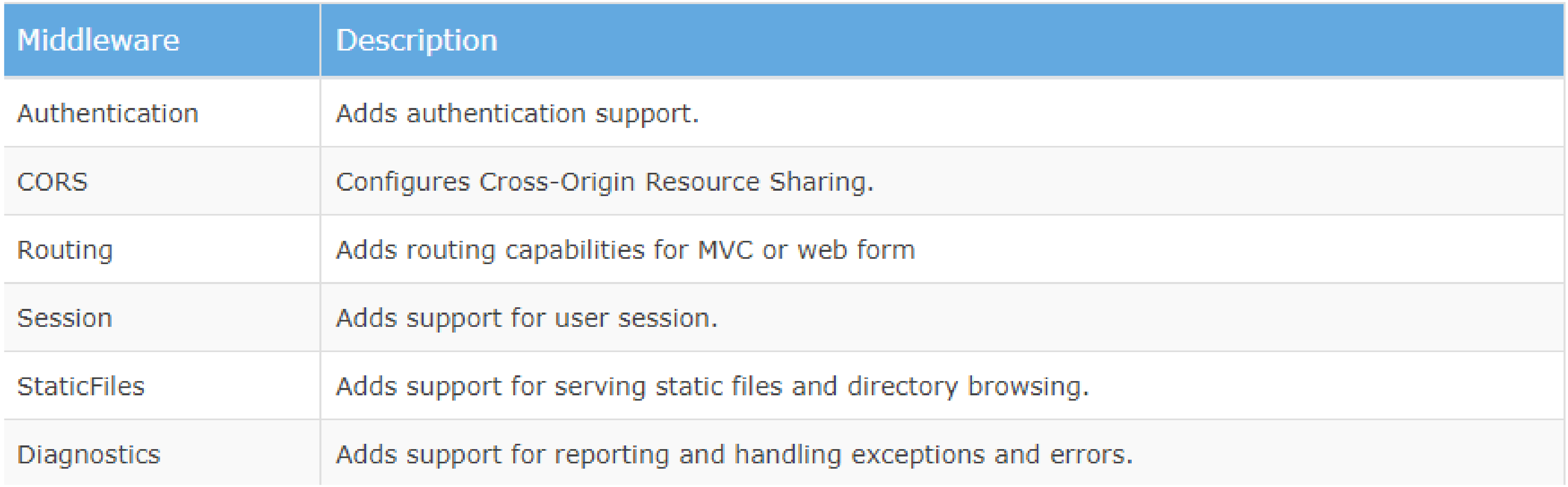
```
public class Startup {
   public Startup() { }
   public void Configure(IApplicationBuilder app, IHostingEnvironment env,
                         ILoggerFactory loggerFactory)
   { //configure middleware using IApplicationBuilder here..
       app.Run(async (context) => { await context.Response.WriteAsync("Hello World1!");
                                  });
       // the following will never be executed
       app.Run(async (context) => { await context.Response.WriteAsync("Hello World2!");
                                  });

    }
}
```

Use() Extension Method

We can use **Use()** method to configure multiple middleware in the order we like.

```
public class Startup {
   public Startup() { }
   public void Configure(IApplicationBuilder app, IHostingEnvironment env,
                         ILoggerFactory loggerFactory)
   { //configure middleware using IApplicationBuilder here..
      app.Use(async (context, next) => { await context.Response.WriteAsync("Hello World1!");

      await  next();
                                           });

       // the following will never be executed
      app.Run(async (context) => { await context.Response.WriteAsync("Hello World2!");
                                     });

    }
}
```

Add Built-in Middleware Via NuGet



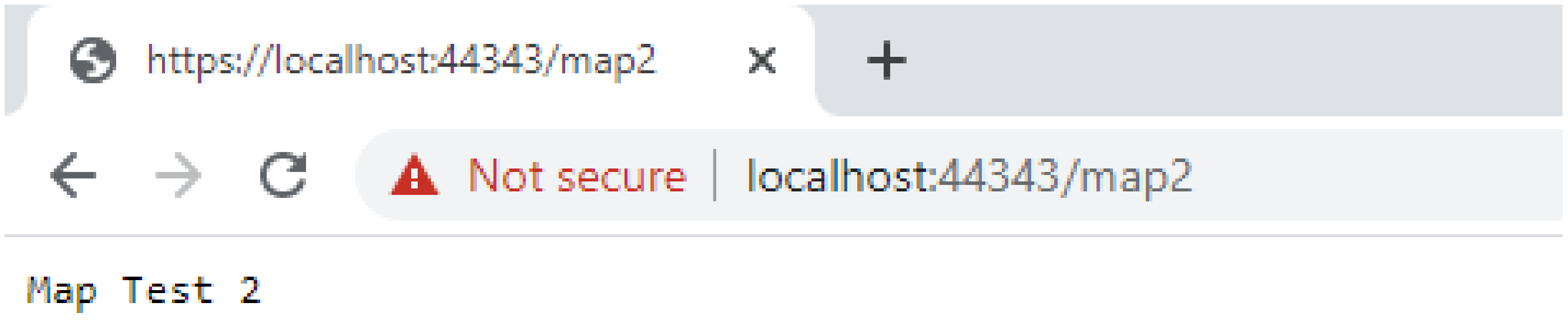
Middleware	Description
Authentication	Adds authentication support.
CORS	Configures Cross-Origin Resource Sharing.
Routing	Adds routing capabilities for MVC or web form
Session	Adds support for user session.
StaticFiles	Adds support for serving static files and directory browsing.
Diagnostics	Adds support for reporting and handling exceptions and errors.

Map Extension Method

- Map extensions are used as a convention for branching the pipeline.
- Map branches the request pipeline based on matches of the given request path.
  - If the request path starts with the given path, the branch is executed.

```
private static void HandleMapTest1(IApplicationBuilder app) {
            app.Run(async context => {
                await context.Response.WriteAsync("Map Test 1"); });
            }
private static void HandleMapTest2(IApplicationBuilder app) {
            app.Run(async context => {
                 await context.Response.WriteAsync("Map Test 2"); }); }

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

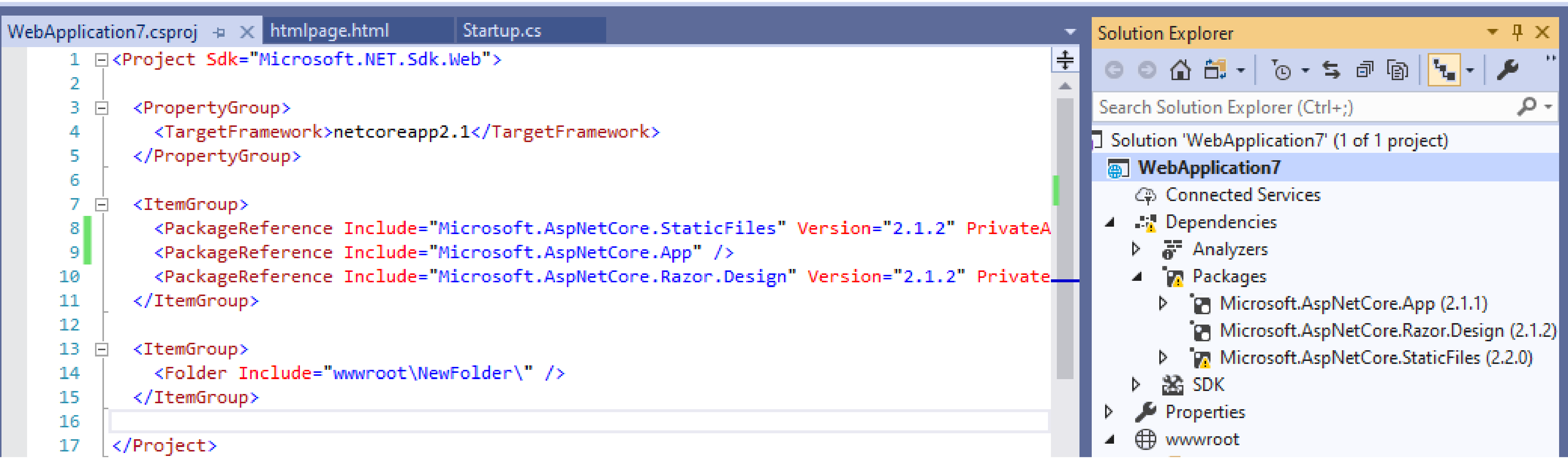
Request	Response
localhost:1234	Hello from non-Map delegate.
localhost:1234/map1	Map Test 1
localhost:1234/map2	Map Test 2
localhost:1234/map3	Hello from non-Map delegate.



```
Map Test 2
```

Serving Static Files

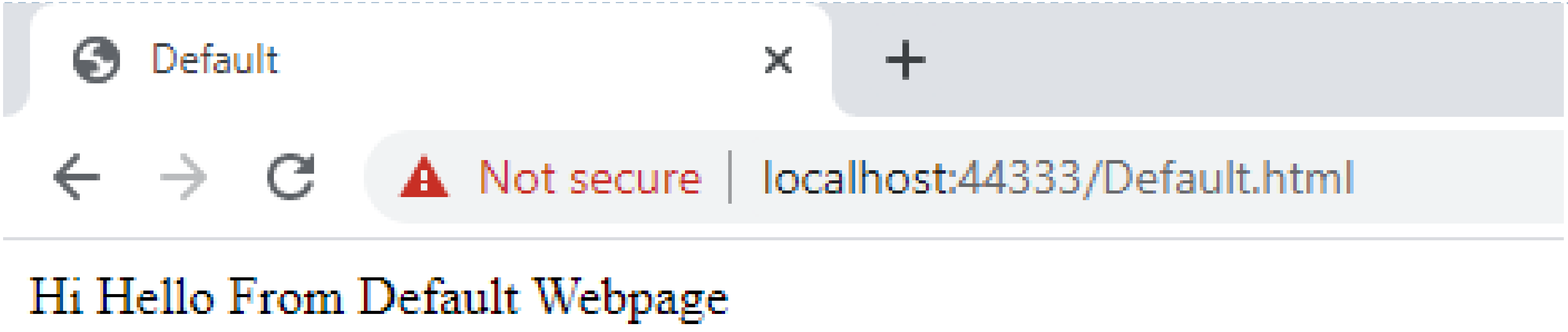
- We must include Microsoft.AspNetCore.StaticFiles middleware in the request pipeline.
- However, Microsoft.AspNetCore.All includes the middleware. Thus, we do not need to install for ASP.NET Core 2.x/3.1



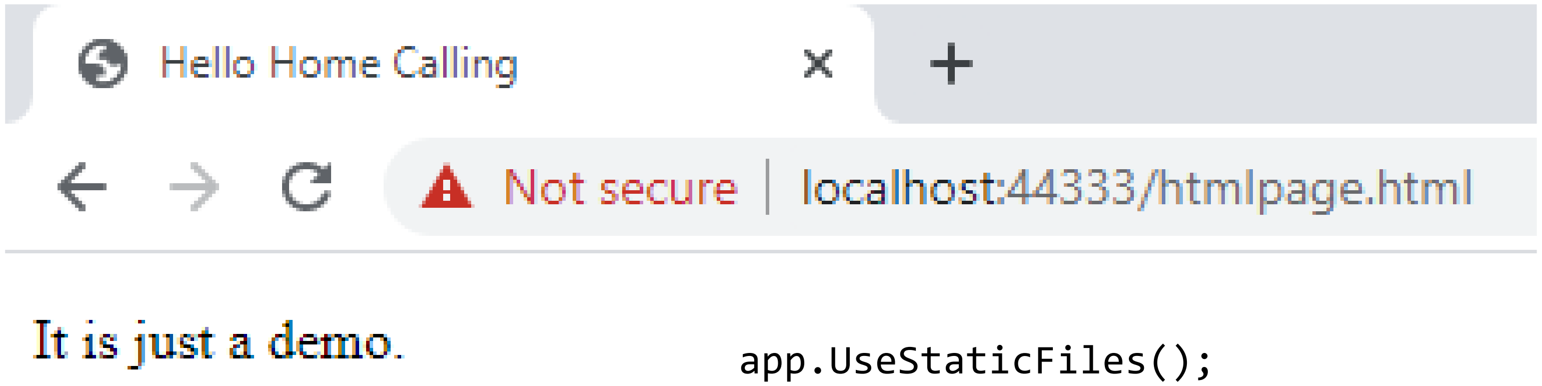
```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>netcoreapp2.1</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.StaticFiles" Version="2.1.2" PrivateA
    <PackageReference Include="Microsoft.AspNetCore.App" />
    <PackageReference Include="Microsoft.AspNetCore.Razor.Design" Version="2.1.2" Private
  </ItemGroup>

  <ItemGroup>
    <Folder Include="wwwroot\NewFolder\" />
  </ItemGroup>

</Project>
```

```
Solution 'WebApplication7' (1 of 1 project)
  WebApplication7
    Connected Services
    Dependencies
      Analyzers
      Packages
        Microsoft.AspNetCore.App (2.1.1)
        Microsoft.AspNetCore.Razor.Design (2.1.2)
        Microsoft.AspNetCore.StaticFiles (2.2.0)
      SDK
    Properties
    wwwroot
```

Default html at wwwroot folder



Hi Hello From Default Webpage

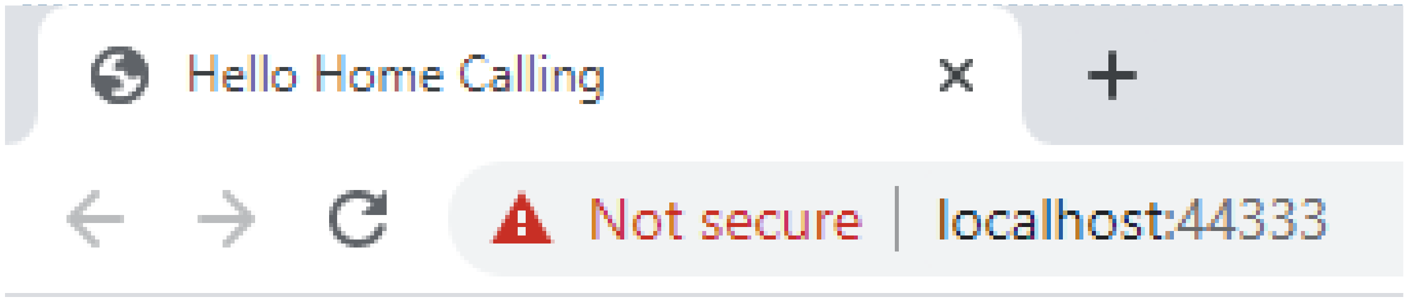


It is just a demo.

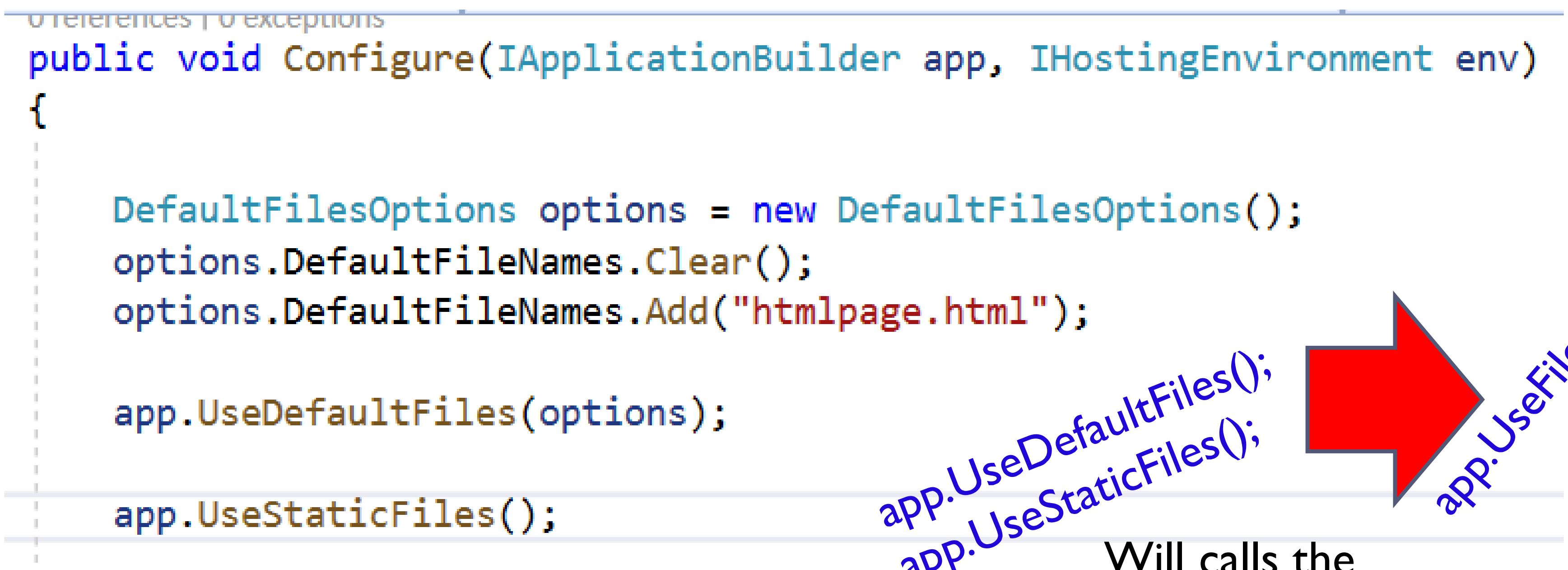
app.UseStaticFiles();

```
app.UseStaticFiles();

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

Setting htmlpage.html is default page



It is just a demo.



```
0 references | 0 exceptions
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{

    DefaultFilesOptions options = new DefaultFilesOptions();
    options.DefaultFileNames.Clear();
    options.DefaultFileNames.Add("htmlpage.html");

    app.UseDefaultFiles(options);

    app.UseStaticFiles();
```

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

app.UseFileServer();

Will calls the Default.html file

Change the htmlpage.html to the following

```html
<!DOCTYPE html>
<html><body>
<p>Enter a number and click OK:</p>
<input id="id1" type="number" min="100" max="300" required>
<button onclick="myFunction()">OK</button>
<p>If the number is less than 100 or greater than 300, an error message
   will be displayed.</p>
<p id="demo"></p>
<script>
function myFunction() {
  var inpObj = document.getElementById("id1");
  if (!inpObj.checkValidity()) {
    document.getElementById("demo").innerHTML =inpObj.validationMessage;
  } else {
    document.getElementById("demo").innerHTML = "Input OK";
  }
}
</script></body></html>
```

rangeOverflow

- <input id="id1" type="number" max="100">
- <button onclick="myFunction()">OK</button>

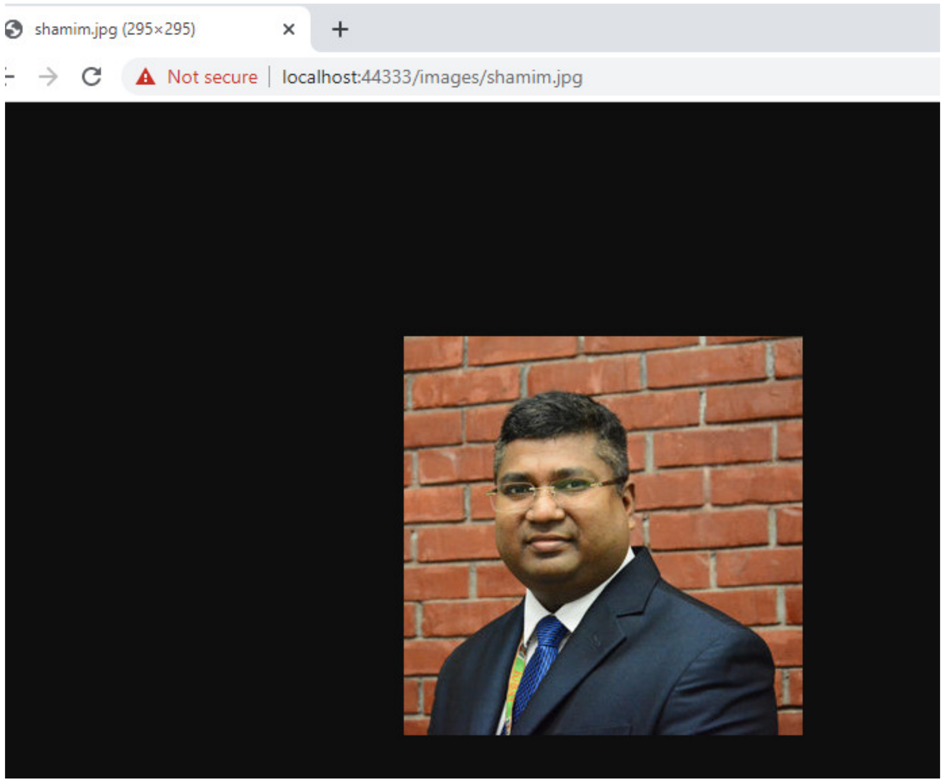
- <p id="demo"></p>

- <script>
- function myFunction() {
-   let text = "Value OK";
-   if (document.getElementById("id1").validity.rangeOverflow) {
-     text = "Value too large";
-   }
- }
- </script

rangeUnderflow

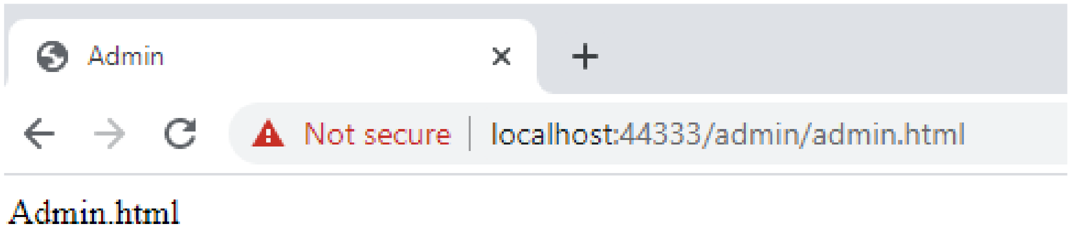
```
<input id="id1" type="number" min="100">
<button onclick="myFunction()">OK</button>

<p id="demo"></p>

<script>
function myFunction() {
  let text = = "Value OK";
  if (document.getElementById("id1").validity.
rangeUnderflow) {
    text = "Value too small";
  }
}
</script>
```

Java scripts calls from JS file



```
app.UseStaticFiles(new StaticFileOptions()
{
    FileProvider = new PhysicalFileProvider(
        Path.Combine(Directory.GetCurrentDirectory(), @"Images")),
    RequestPath = new PathString("/images")

});
```

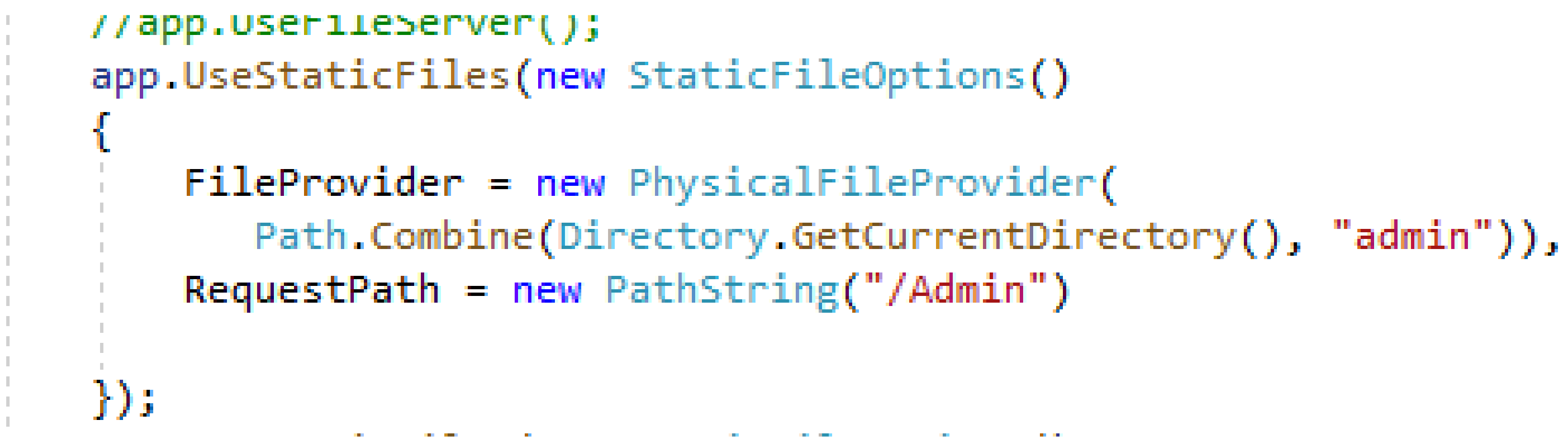
Get file from Admin folder



Admin

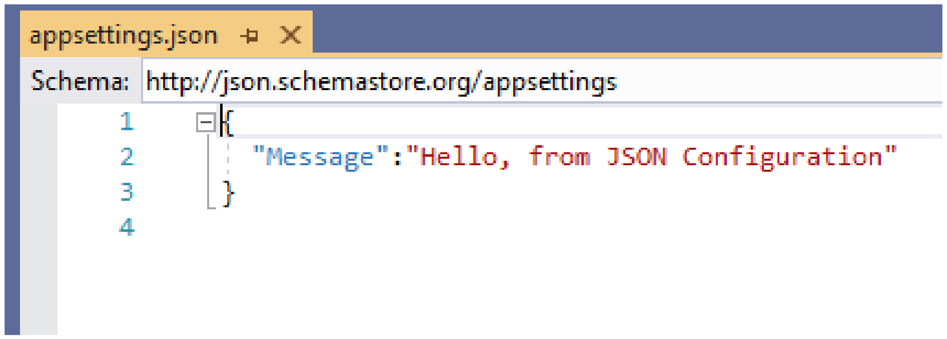
Not secure | localhost:44333/admin/admin.html

Admin.html



```
//app.UseFileServer();
app.UseStaticFiles(new StaticFileOptions()
{
    FileProvider = new PhysicalFileProvider(
       Path.Combine(Directory.GetCurrentDirectory(), "admin")),
    RequestPath = new PathString("/Admin")

});
```

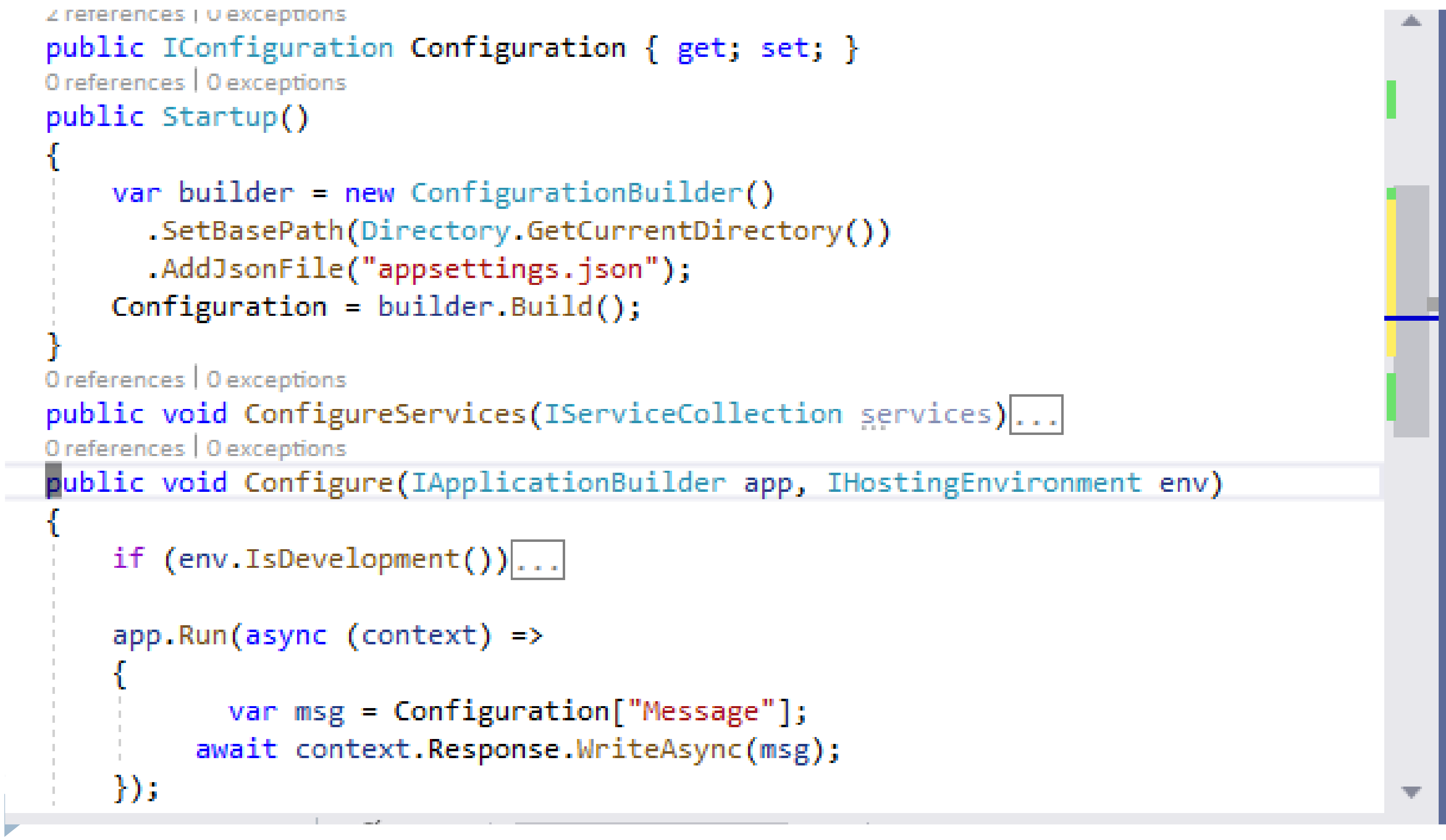
Adding a JSON file



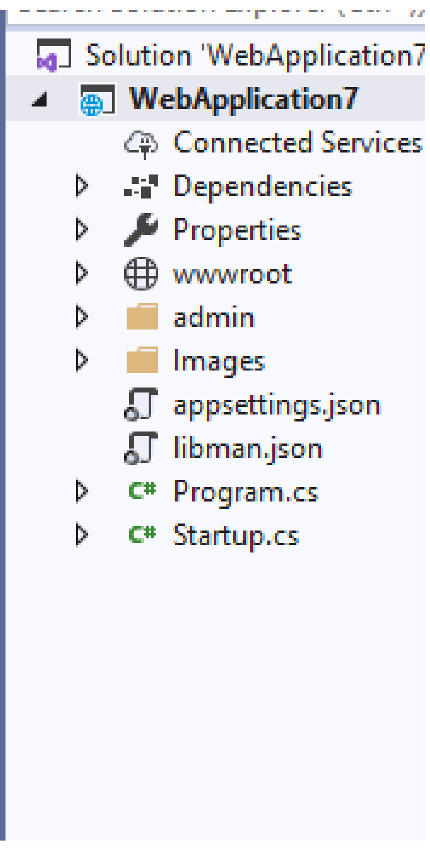
appsettings.json

Schema: http://json.schemastore.org/appsettings

```json
{
  "Message":"Hello, from JSON Configuration"
}
```

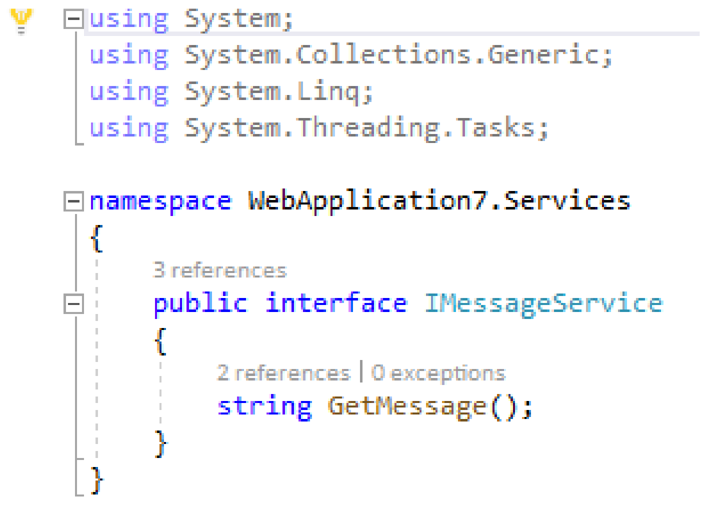


```csharp
2 references | 0 exceptions
public IConfiguration Configuration { get; set; }
0 references | 0 exceptions
public Startup()
{
    var builder = new ConfigurationBuilder()
      .SetBasePath(Directory.GetCurrentDirectory())
      .AddJsonFile("appsettings.json");
    Configuration = builder.Build();
}
0 references | 0 exceptions
public void ConfigureServices(IServiceCollection services)...
0 references | 0 exceptions
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())...

    app.Run(async (context) =>
    {
            var msg = Configuration["Message"];
          await context.Response.WriteAsync(msg);
    });
```

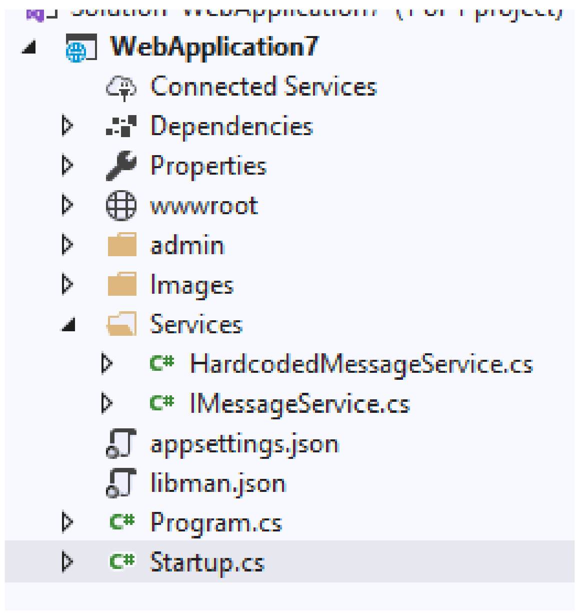


Adding a Service

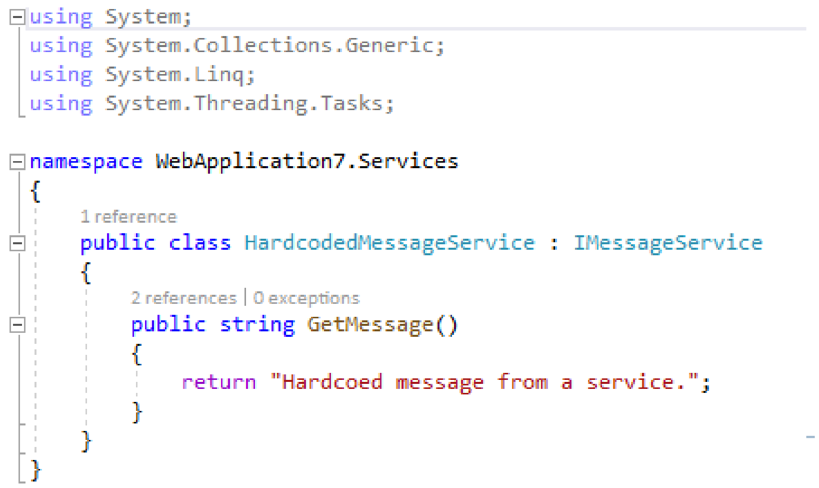
- Add one interface @ Service folder



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace WebApplication7.Services
{
    3 references
    public interface IMessageService
    {
        2 references | 0 exceptions
        string GetMessage();
    }
}
```

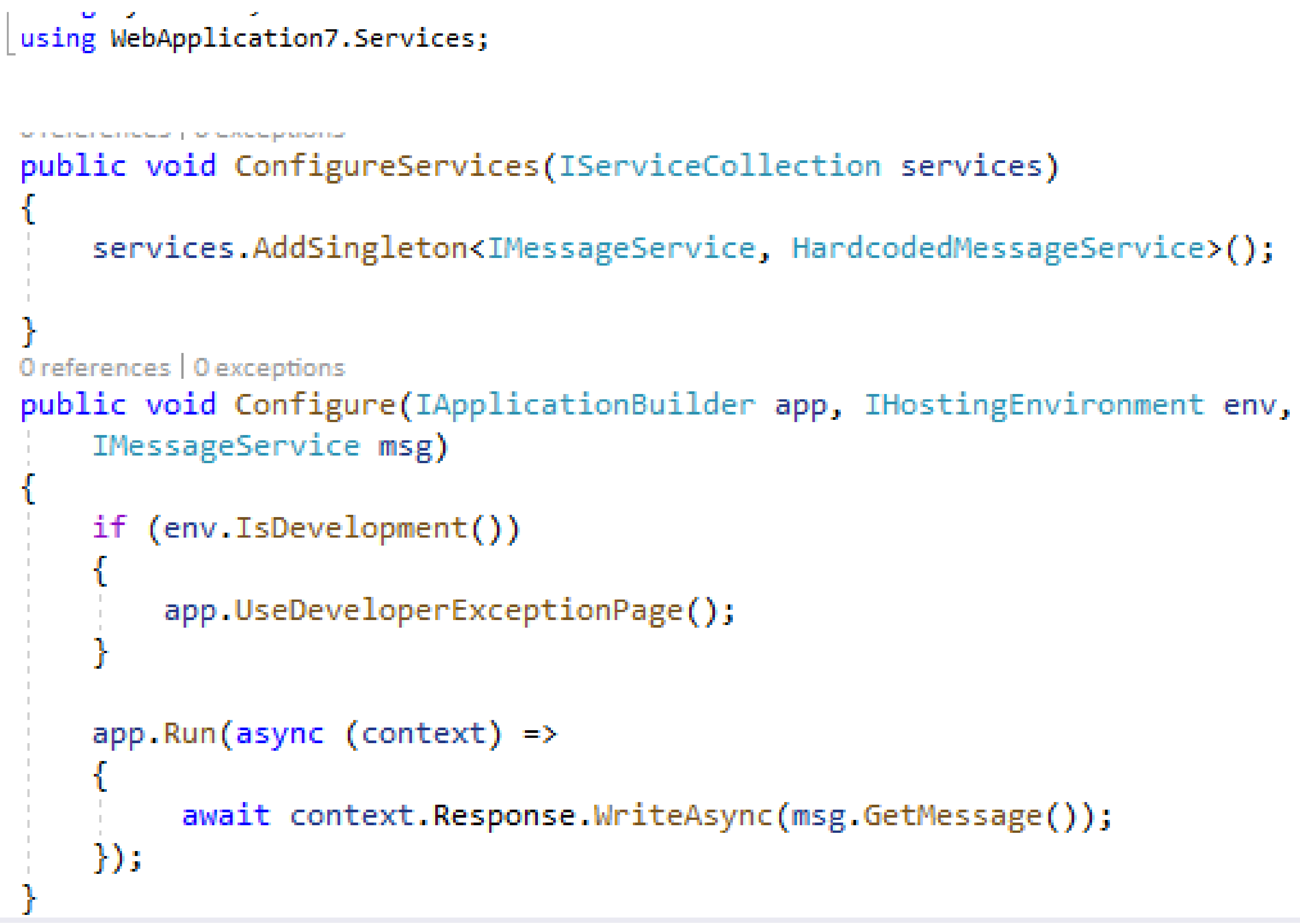


- Add one class @HardcodedMessageService



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace WebApplication7.Services
{
    1 reference
    public class HardcodedMessageService : IMessageService
    {
        2 references | 0 exceptions
        public string GetMessage()
        {
            return "Hardcoed message from a service.";
        }
    }
}
```

Configure A Service



```
using WebApplication7.Services;


public void ConfigureServices(IServiceCollection services)
{
    services.AddSingleton<IMessageService, HardcodedMessageService>();

}
0 references | 0 exceptions
public void Configure(IApplicationBuilder app, IHostingEnvironment env,
    IMessageService msg)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.Run(async (context) =>
    {
         await context.Response.WriteAsync(msg.GetMessage());
    });
}
```