Configuring Middleware Components || Understanding app.Use and app.Run

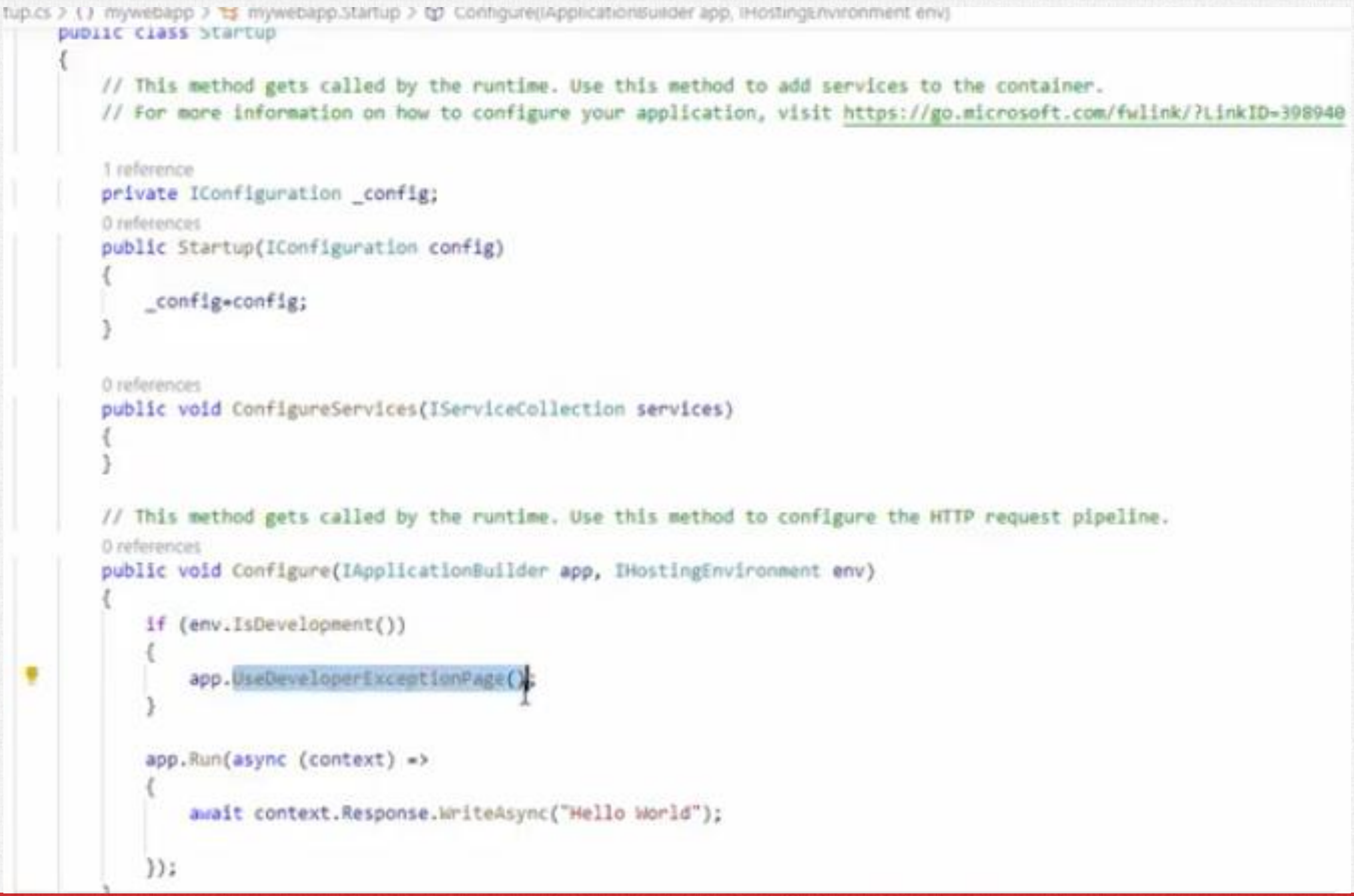
What are Request Delegates in ASP.NET Core?

- In ASP.NET Core, Request delegates are used to build the request pipeline i.e. request delegates are used to handle each incoming HTTP request. In ASP.NET Core, you can configure the Request delegates using the **Run, Map, and Use** extension methods.
- You can specify a request delegate using an in-line anonymous method (called in-line middleware) or you can specify the request delegates using a reusable class. These reusable classes and in-line anonymous methods are called as middleware or middleware components.
- Each middleware component in the request processing pipeline is responsible for invoking the next component in the pipeline or short-circuiting the pipeline by not calling the next middleware component.

What is the use of Use and Run methods?

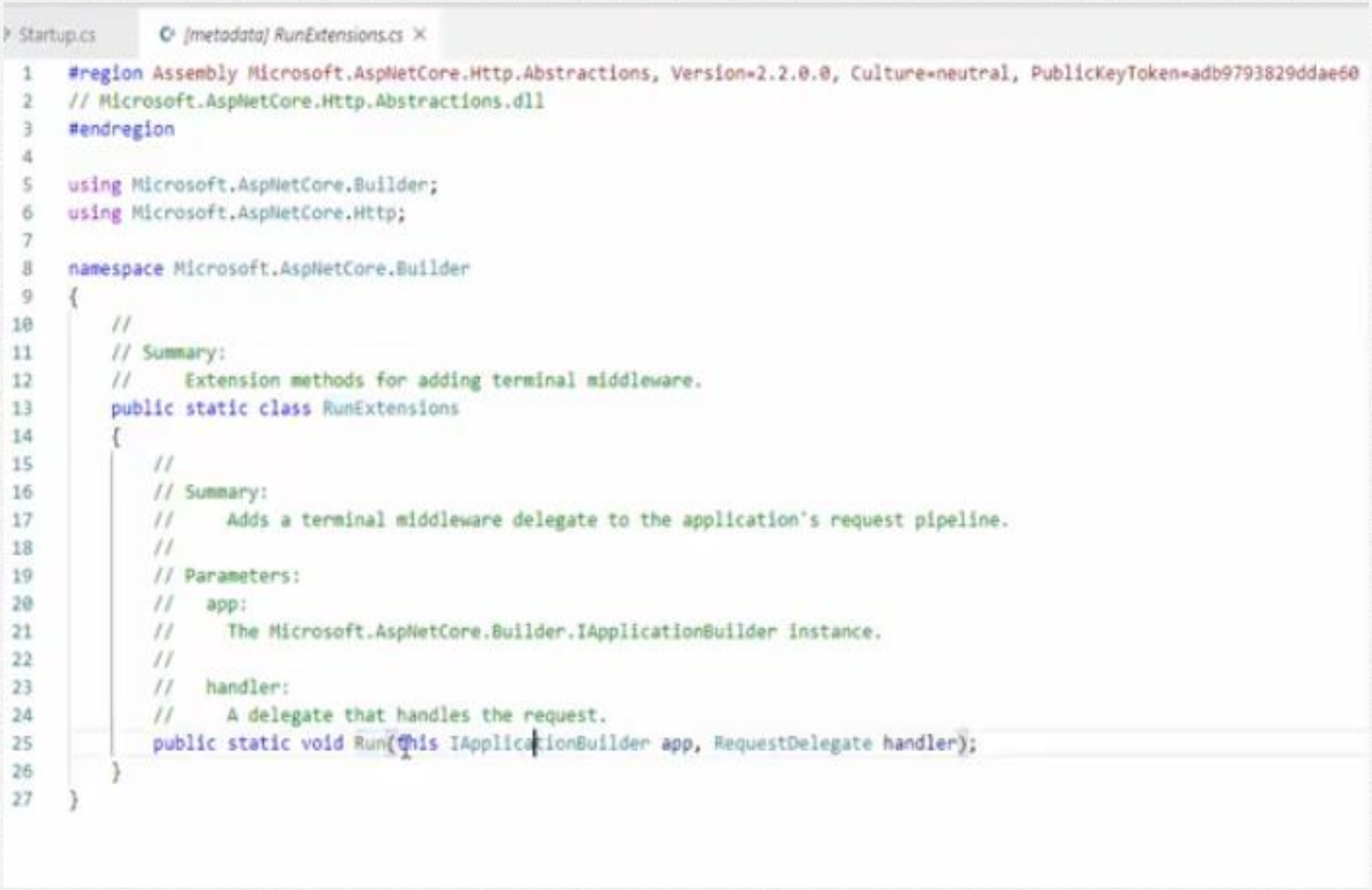
- In ASP.NET Core, you can use the "**Use**" and "**Run**" extension methods to register the Inline Middleware component into the Request processing pipeline.
- The "Run" extension method allows us to add the terminating middleware (the middleware which will not call the next middleware components in the request processing pipeline).
- On the other hand, the "Use" extension method allows us to add the middleware components which may call the next middleware component in the request processing pipeline.
- If you observe the Configure method, then you will see that it gets an instance of the IApplicationBuilder interface and using that instance along with the extension methods such as Use and Run, it configures the Middleware components.
- As you can see, in the configure method, two middleware components are registered in the request processing pipeline using the IApplicationBuilder instance i.e. app.
- They are as follows
  - **UseDeveloperExceptionPage() Middleware Component**
  - **Middleware Component Registered using Run() method**

Two middleware component inside Configure Method:-
1)app.useDeveloperExceptionPage() ; 2)app.Run() Extension methods.



```
tup.cs > {} mywebapp > mywebapp.Startup > Configure(IApplicationBuilder app, IHostingEnvironment env)
    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

        1 reference
        private IConfiguration _config;
        0 references
        public Startup(IConfiguration config)
        {
            _config=config;
        }

        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, IHostingEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }

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

            });
```

Goto Definition of Run Method:-



```
#region Assembly Microsoft.AspNetCore.Http.Abstractions, Version=2.2.0.0, Culture=neutral, PublicKeyToken=adb9793829ddae60
// Microsoft.AspNetCore.Http.Abstractions.dll
#endregion

using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Http;

namespace Microsoft.AspNetCore.Builder
{
    //
    // Summary:
    //     Extension methods for adding terminal middleware.
    public static class RunExtensions
    {
        //
        // Summary:
        //     Adds a terminal middleware delegate to the application's request pipeline.
        //
        // Parameters:
        //   app:
        //     The Microsoft.AspNetCore.Builder.IApplicationBuilder instance.
        //
        //   handler:
        //     A delegate that handles the request.
        public static void Run(this IApplicationBuilder app, RequestDelegate handler);
    }
}
```

What is the use of DeveloperExceptionPage Middleware Component?

- As you can see, within the configure method, the **UseDeveloperExceptionPage()** middleware component is registered into the pipeline and this middleware will come into the picture only when the hosting environment is "development".
- This middleware component is going to execute when there is an unhandled exception occurred in the application and since it is in development mode, it is going to show you the culprit line of code.

How to Configure Middleware Components using the Run() extension method?

- The second middleware component is registered by using the **Run()** extension method. As this component is registered using the Run extension method, so it is going to be a terminating middleware component means it will not call the next middleware component in the request processing pipeline.
- This middleware component is simply going to write the "**Hello World!**" message on to the Response object. The points that you need to remember is, this is the component which will respond to each and every incoming HTTP request at the moment.
- The second middleware component is registered by using the Run() extension method. As this component is registered using the Run extension method, so it is going to be a terminating middleware component means it will not call the next middleware component in the request processing pipeline. It simply going to write the "Hello World!" message on to the Response object.
- At the moment if you run the application, then you will see the "Hello World!" message irrespective of the URL Path. That means all the incoming requests are handled by this middleware (Run) component only.

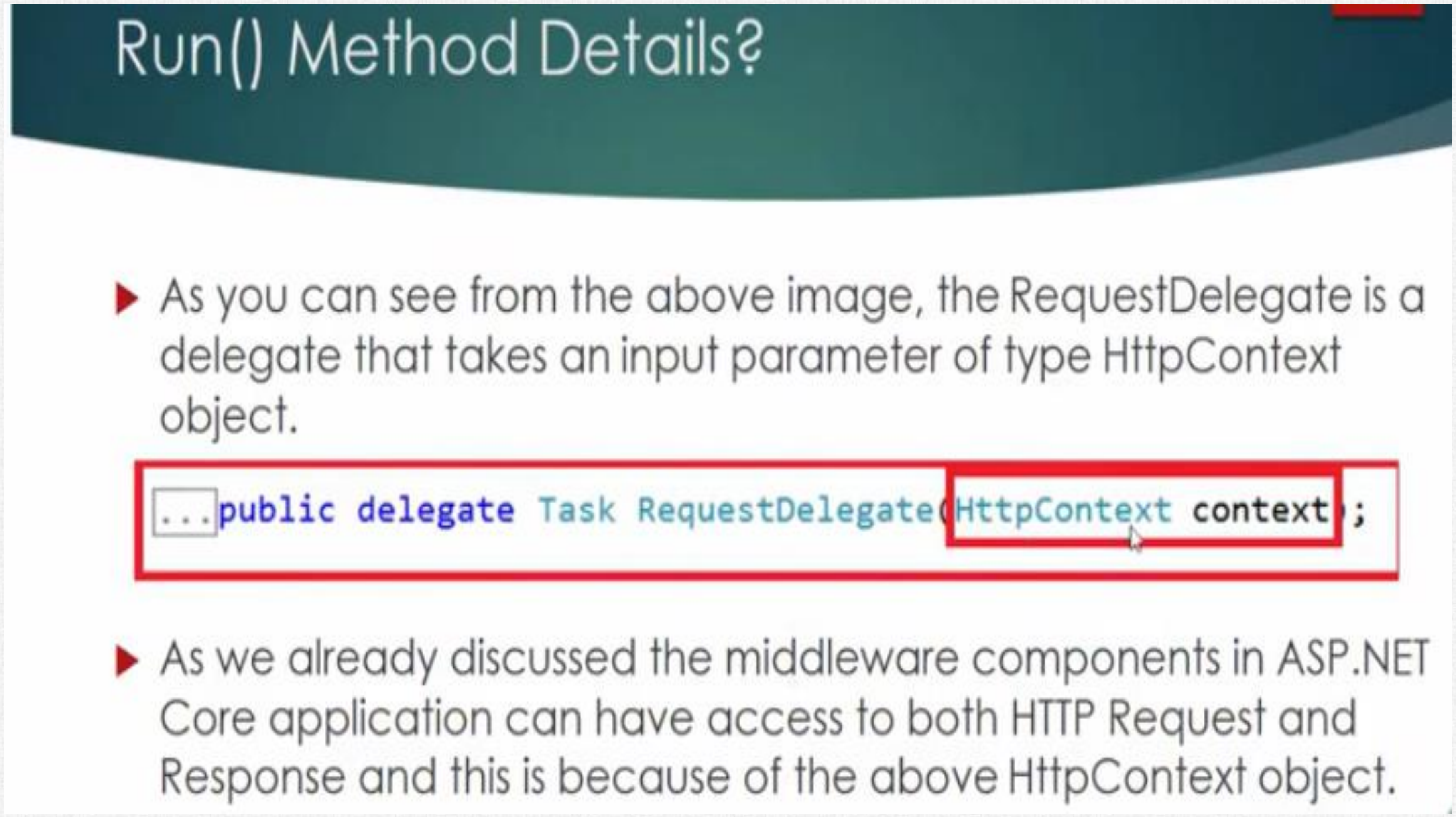
Run() Method Details?

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

- We are invoking the **Run()** method on the **IApplicationBuilder** instance (app) to register the middleware component into the request processing pipeline. Following is the definition of the **Run** method.

```
...public static class RunExtensions
{
    ...public static void Run(this IApplicationBuilder app, RequestDelegate handler);
}
```

- As you can see from the definition of the Run() method, it is implemented as an extension method of the IApplicationBuilder interface. This is the reason why we are able to call th Run() method using the IApplicationBuilder instance i.e. app



Run() Method Details?

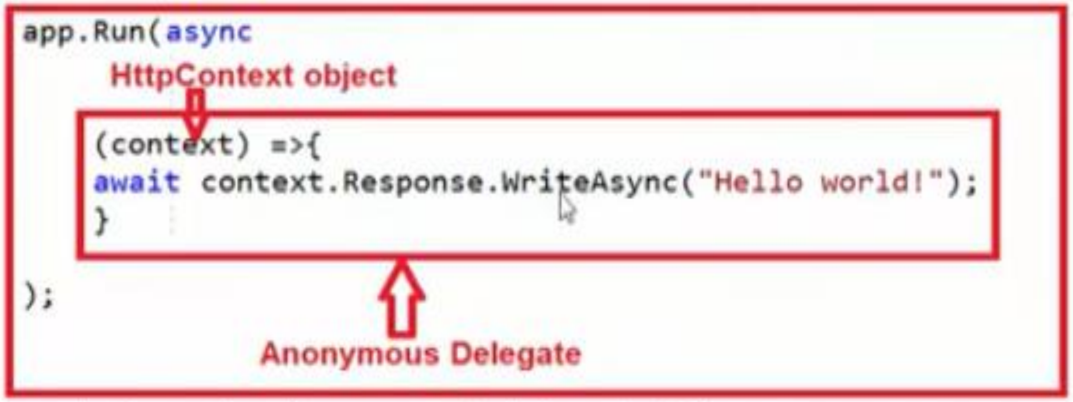
- As you can see from the above image, the RequestDelegate is a delegate that takes an input parameter of type HttpContext object.

```
...public delegate Task RequestDelegate(HttpContext context);
```

- As we already discussed the middleware components in ASP.NET Core application can have access to both HTTP Request and Response and this is because of the above HttpContext object.

Run() Method Details?

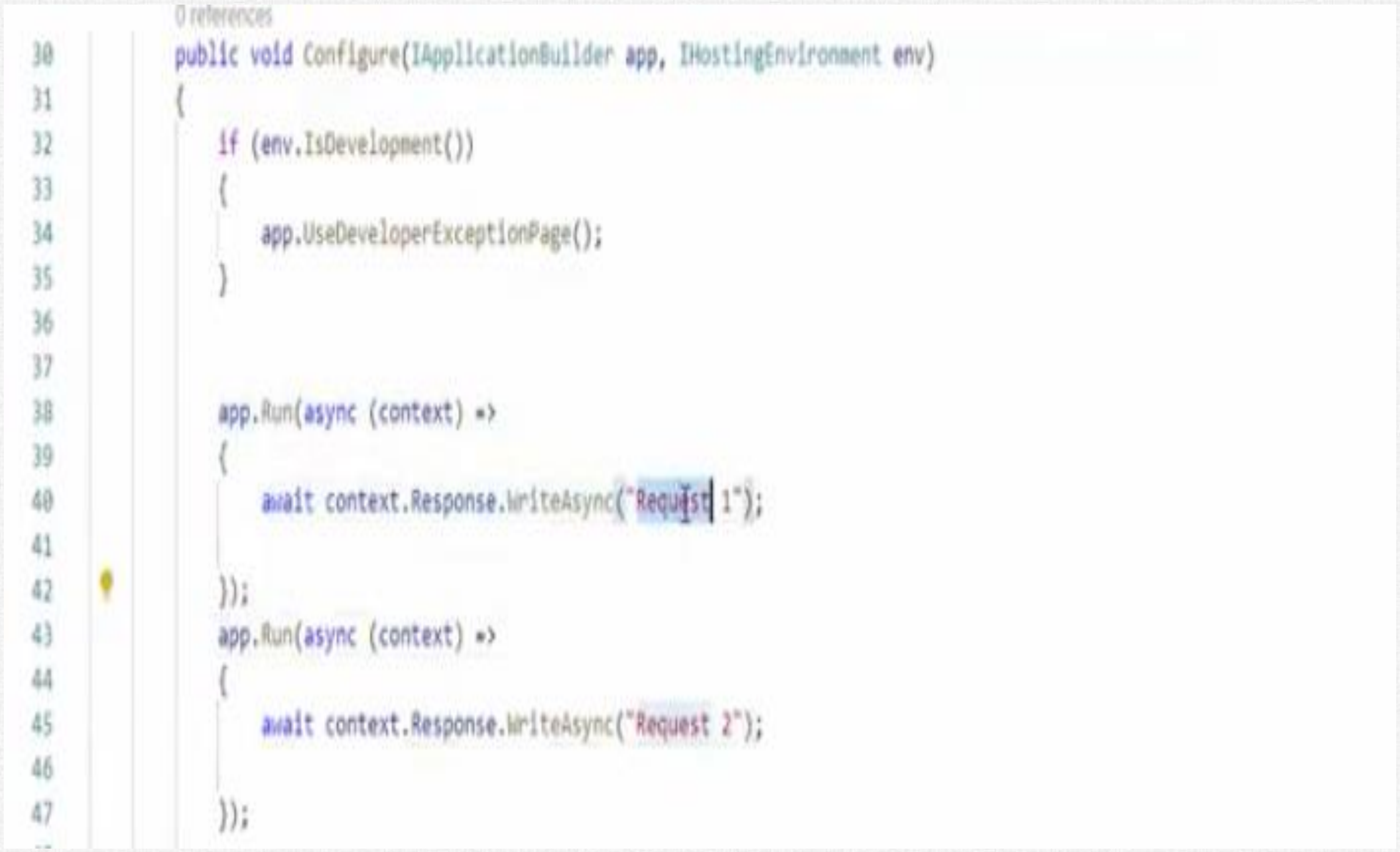
- In our example, we are passing the request delegate inline as an anonymous method using lambda expression and moreover we passing the HTTPContext object as an input parameter to the request delegate. The following diagram shows the above



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

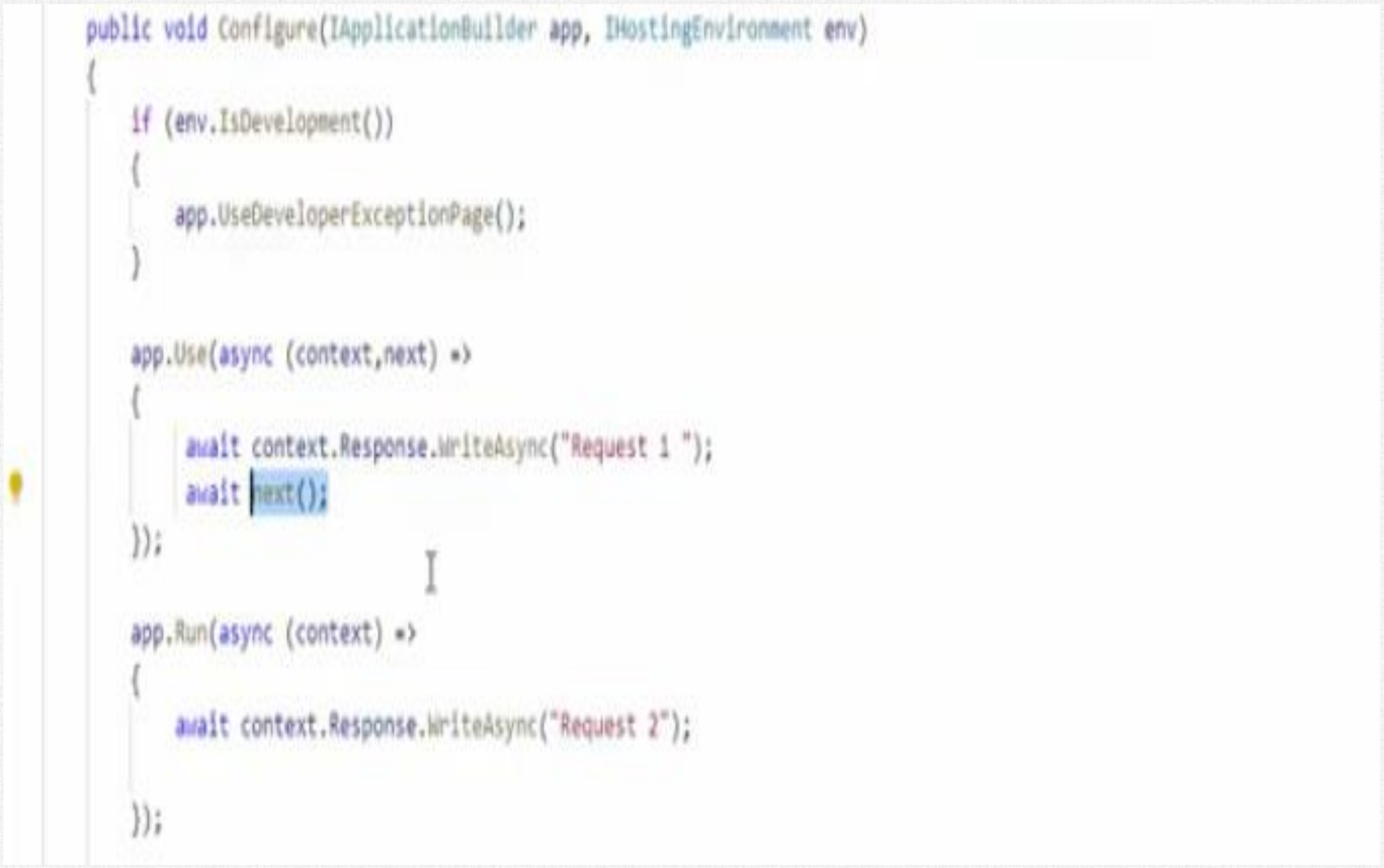
- **Note:** Instead of passing the request delegate inline as an anonymous method, you can also define the request delegate in a separate class and pass it here

Adding two run method inside the request pipeline and check it will termination the request pipeline or not.
Only print Request 1 (it’s a terminate the request pipeline by Run method) ;
Request 2 is not print.



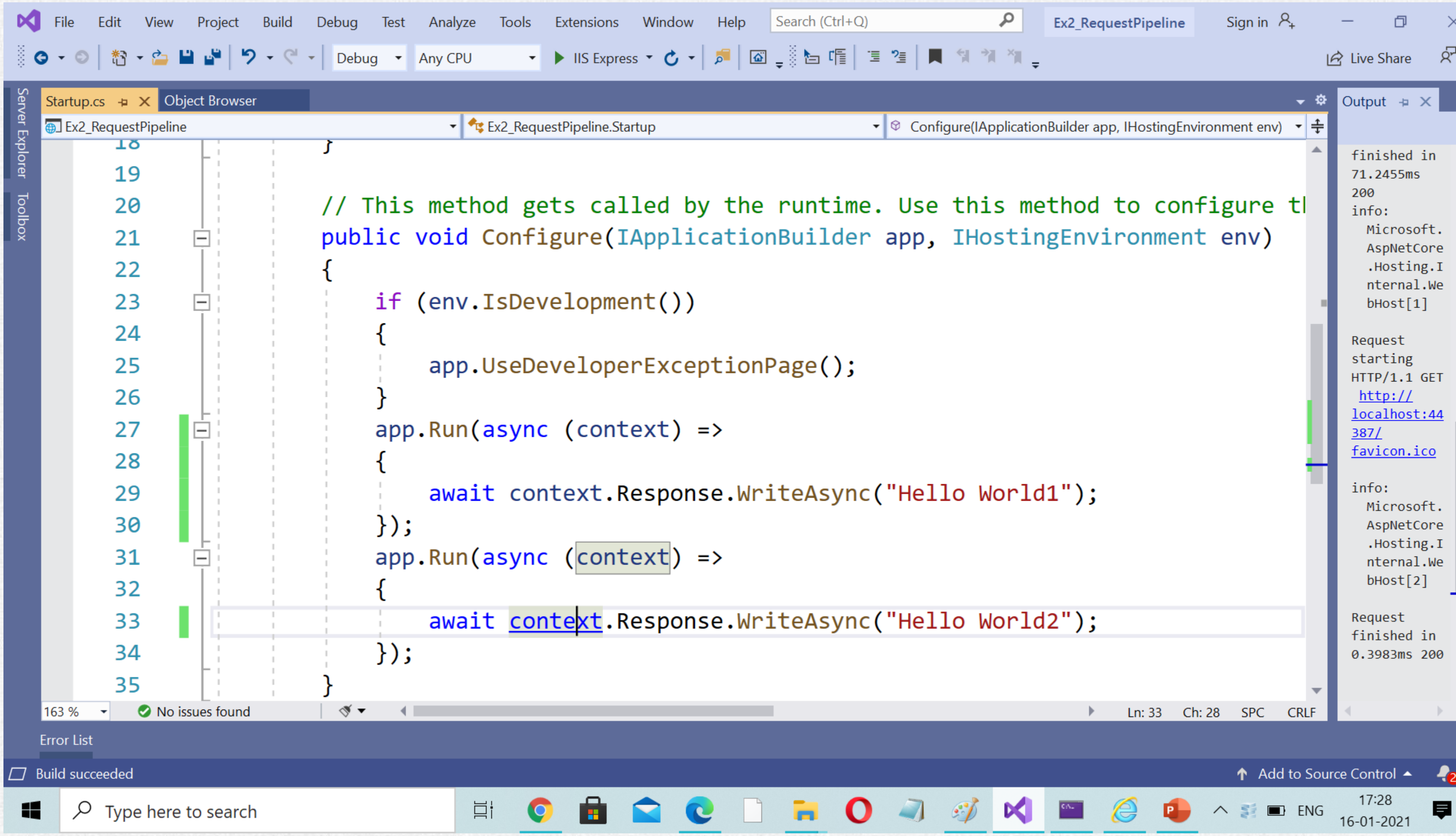
```
0 references
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }


    app.Run(async (context) =>
    {
        await context.Response.WriteAsync("Request 1");

    });
    app.Run(async (context) =>
    {
        await context.Response.WriteAsync("Request 2");

    });
```

How we use two request pipeline:- By using app.Use() Extension method inside this Use() Method call Next() method to go another request pipeline.



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

    app.Use(async (context,next) =>
    {
        await context.Response.WriteAsync("Request 1 ");
        await next();
    });

    app.Run(async (context) =>
    {
        await context.Response.WriteAsync("Request 2");

    });
```

Output is Hello World1 :-Run method terminate request pipeline process.

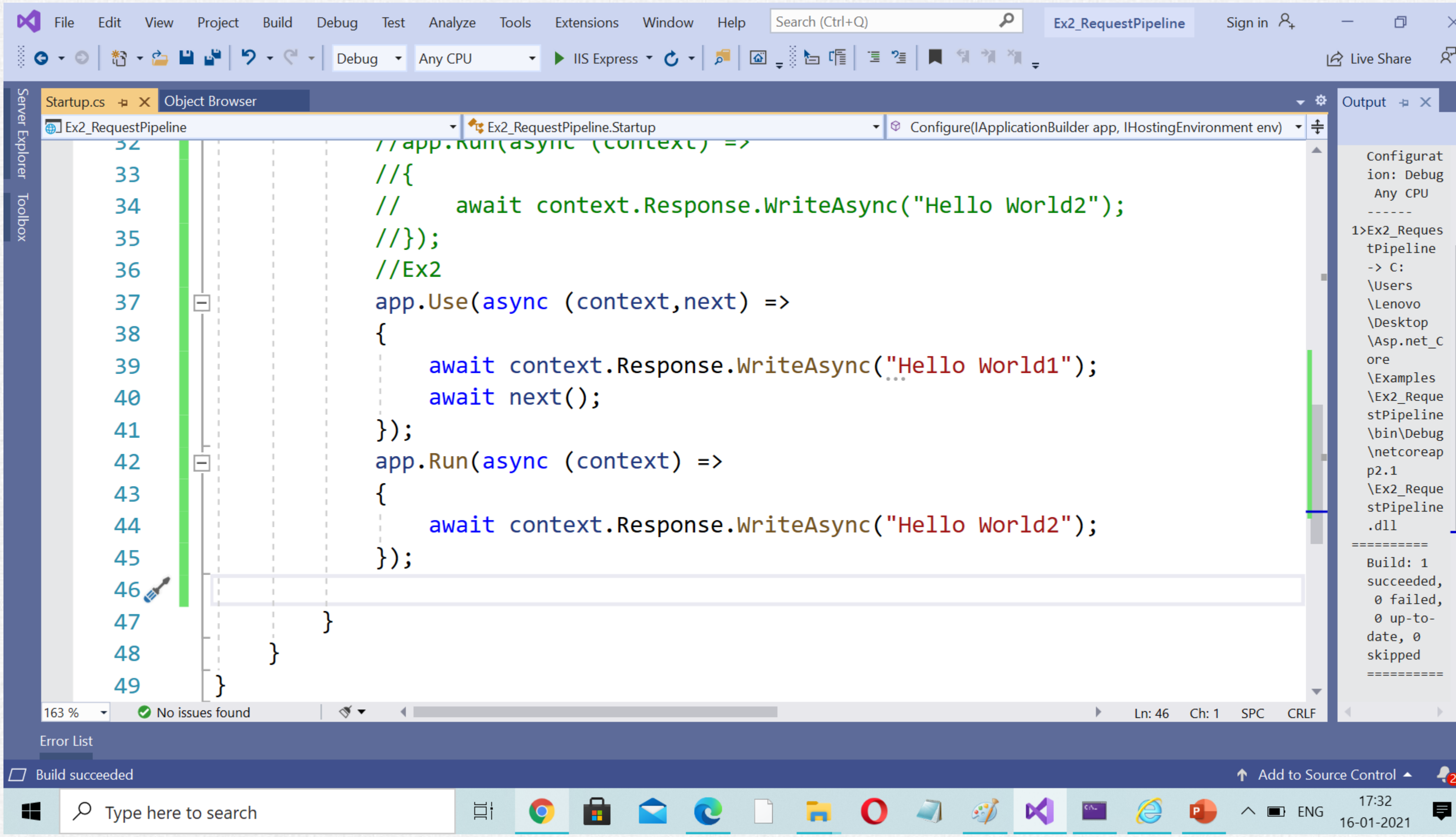


```
        }

        // This method gets called by the runtime. Use this method to configure tl
        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            app.Run(async (context) =>
            {
                await context.Response.WriteAsync("Hello World1");
            });
            app.Run(async (context) =>
            {
                await context.Response.WriteAsync("Hello World2");
            });
        }
```

Output

```
finished in
71.2455ms
200
info:
  Microsoft.
  AspNetCore
  .Hosting.I
  nternal.We
  bHost[1]

Request
starting
HTTP/1.1 GET
 http://
localhost:44
387/
favicon.ico

info:
  Microsoft.
  AspNetCore
  .Hosting.I
  nternal.We
  bHost[2]

Request
finished in
0.3983ms 200
```

Output:- Hello World1 HelloWorld2
use method not terminate the request pipeline.



```
            //app.Run(async (context) =>
            //{
            //     await context.Response.WriteAsync("Hello World2");
            //});
            //Ex2
            app.Use(async (context,next) =>
            {
                await context.Response.WriteAsync("Hello World1");
                await next();
            });
            app.Run(async (context) =>
            {
                await context.Response.WriteAsync("Hello World2");
            });

        }
    }
}
```

```
Configuration: Debug Any CPU ------
1>Ex2_RequestPipeline -> C:\Users\Lenovo\Desktop\Asp.net_Core\Examples\Ex2_RequestPipeline\bin\Debug\netcoreapp2.1\Ex2_RequestPipeline.dll
==========
Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped
==========
```

Build succeeded

- For short circuit Use run method
- For switch to another middleware then use, Use() Method

Understanding the Use extension method:

- The Use extension method adds a middleware delegate defined in-line to the application's request pipeline. Following is the definition of the Use extension method:

```
public static class UseExtensions
{
    public static IApplicationBuilder Use(
                this IApplicationBuilder app,
                Func<HttpContext, Func<Task>, Task> middleware
        );
}
```

- This method is also implemented as an extension method on the IApplicationBuilder interface. This is the reason why we are able to invoke this method using the IApplicationBuilder instance. As you can see from the above definition, this method takes two input parameters.
  - The first parameter is the HttpContext context object through which it can access both the HTTP request and response.
  - The second parameter is the Func type i.e. it is a generic delegate that can handle the request or call the next middleware component in the request pipeline.

Thanks