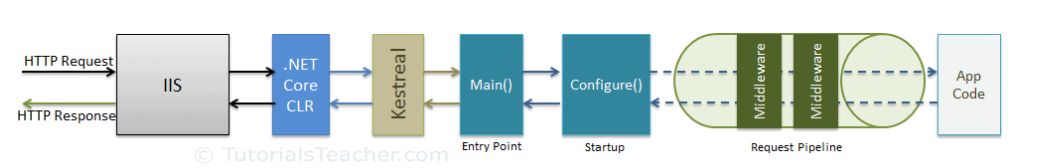
**ASP.NET CORE 6**

ASP.NET Core is a cross-platform, high-performance, open-source framework for building modern, cloud-based, Internet-connected applications.

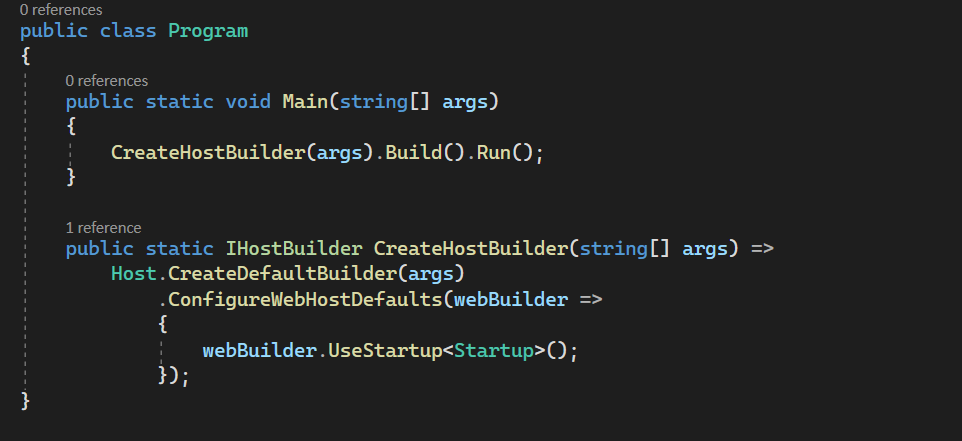
****

**ASP.NET Core:**

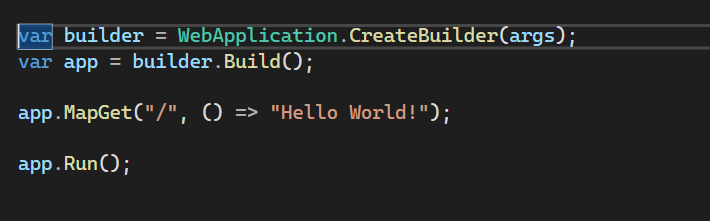
|  |  |
| --- | --- |
|  | - start up |
|  | - host |
|  | - servers |
|  | - configuration (options pattern) |
|  | - dependency injection (services) |
|  | - middleware |
|  | - environments => development, production and staging |
|  | - logging |
|  | - routing |
|  | - handling errors |
|  |  |

**Difference between .NET Core 3.1/5.0 and .NET CORE 6 in Program file**

.NET Core 3.1

****

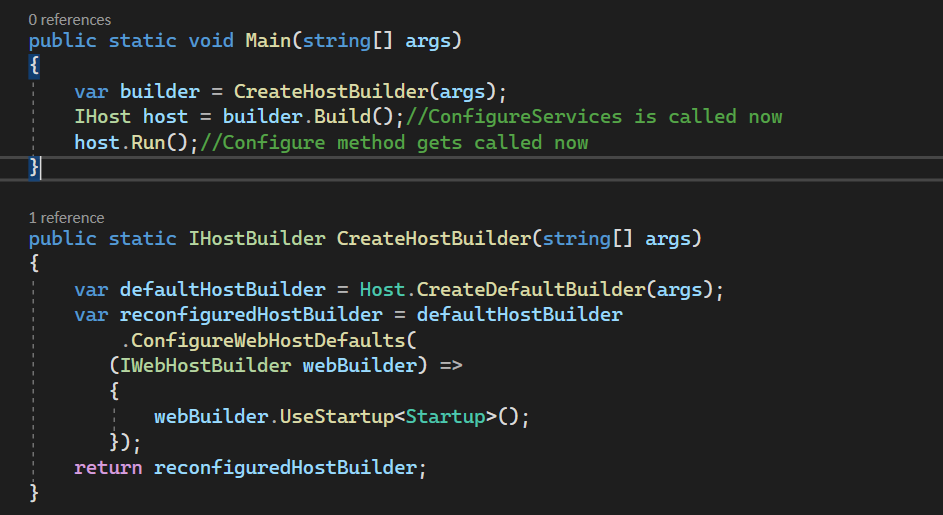
**.**NET CORE 6

****

**Hosting:**

ASP.NET Core apps configure and launch a host. The host is responsible for app startup and lifetime management. At a minimum, the host configures a server and a request processing pipeline. The host can also set up logging, dependency injection, and configuration.

<https://andrewlock.net/introducing-ihostlifetime-and-untangling-the-generic-host-startup-interactions/>

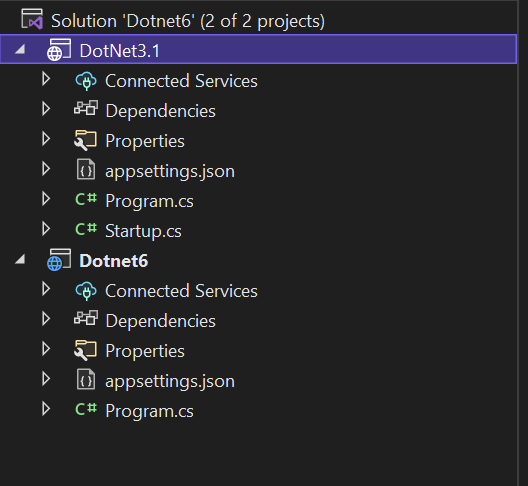
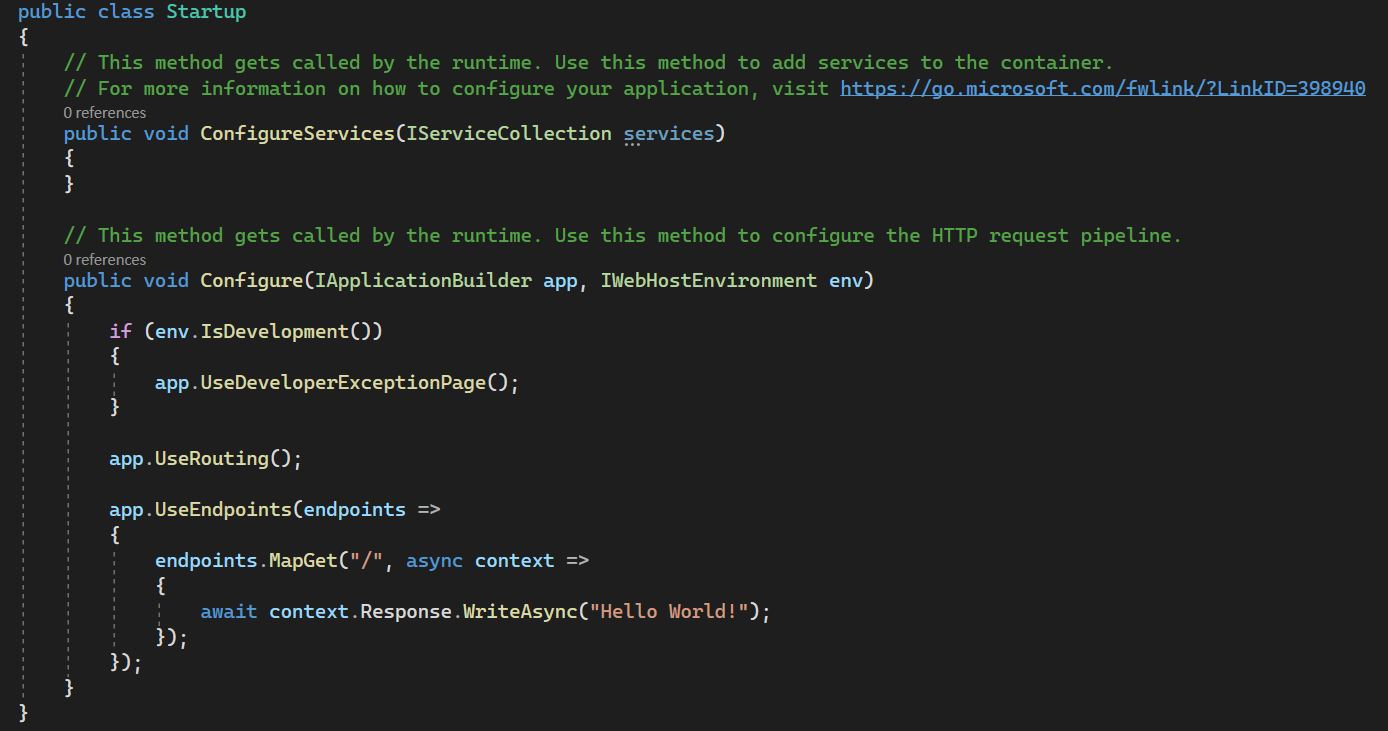
****

**Diagram

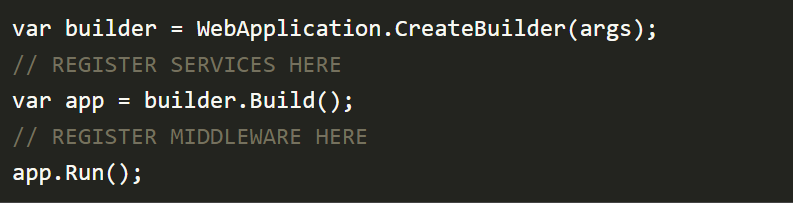
Description automatically generated**

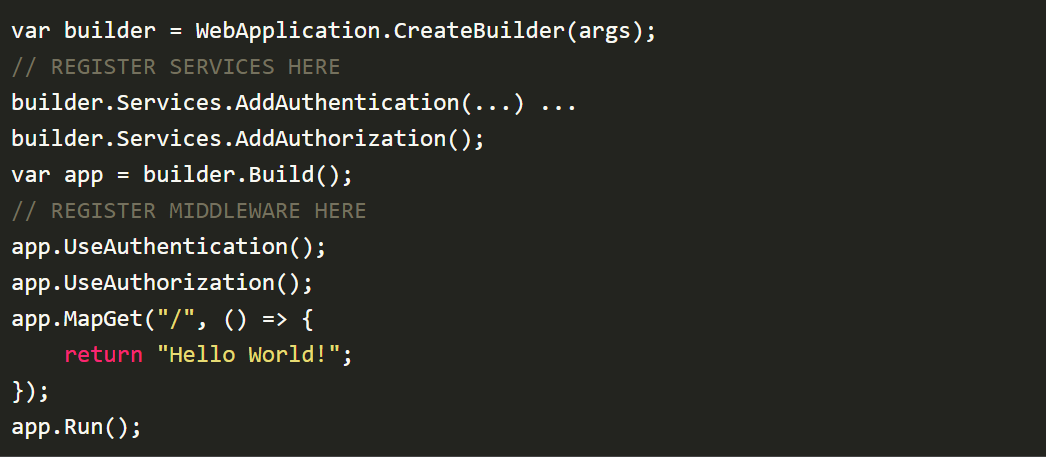
**Startup.cs**

In ConfigureServices() you register services for dependency injection. In Configure(), you outline your middleware pipeline order and structure

** **

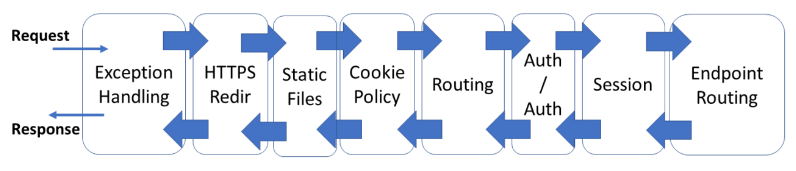
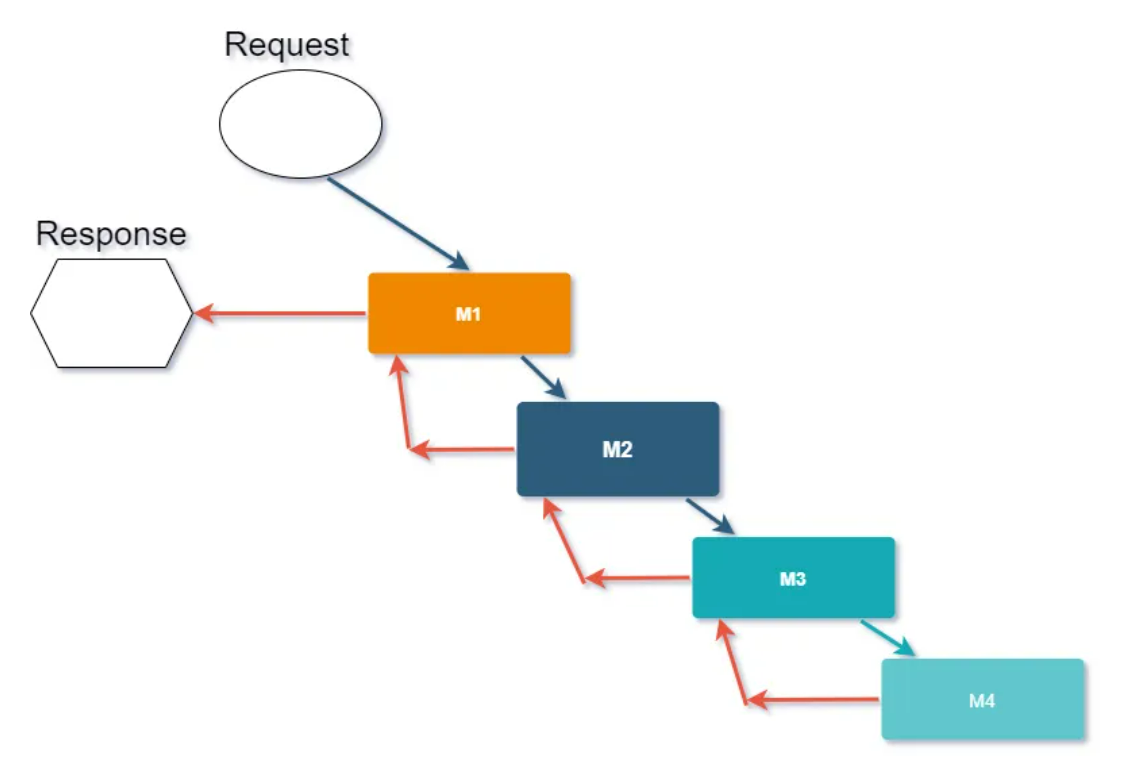
**Note**: In NET CORE 6 there is no startup.cs instead register like below in program.cs

****

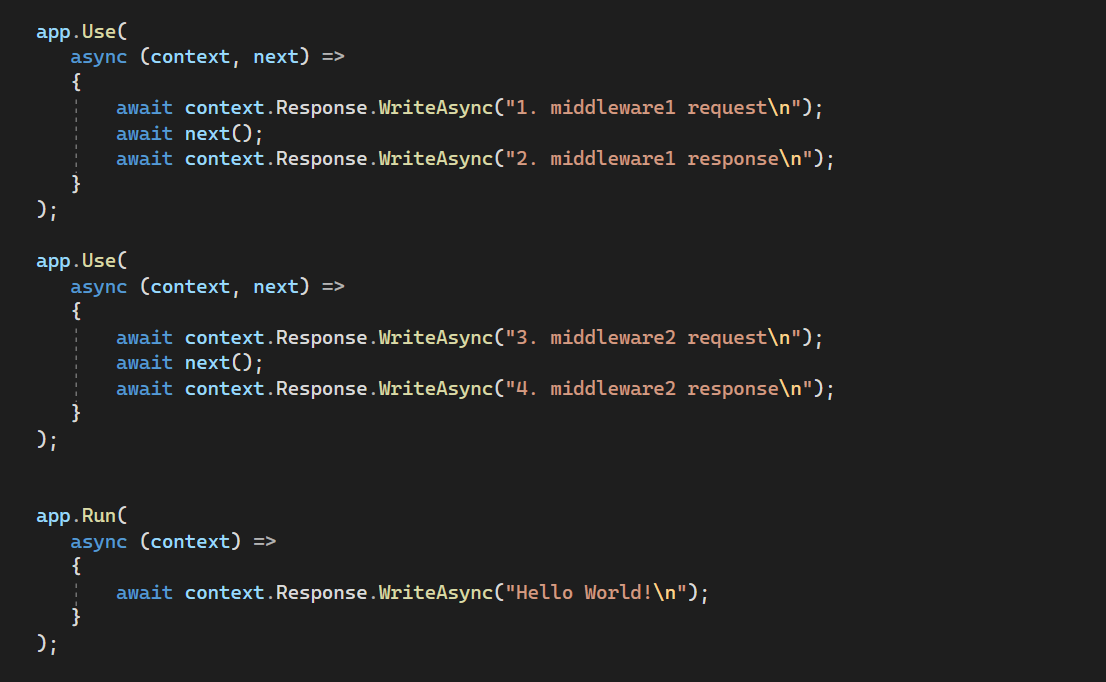
****

**Middleware:**

An ASP.NET Core Middleware is a component that sits on the HTTP Request Pipeline and handles requests and responses. There can be one or more Middleware on the HTTP pipeline in a line. RequestDelegate is used to build the request pipeline.

****

Here, next() method were used to call next middleware in this pipeline.

****

**Output:**

1.middleware1 request

3.middleware2 request

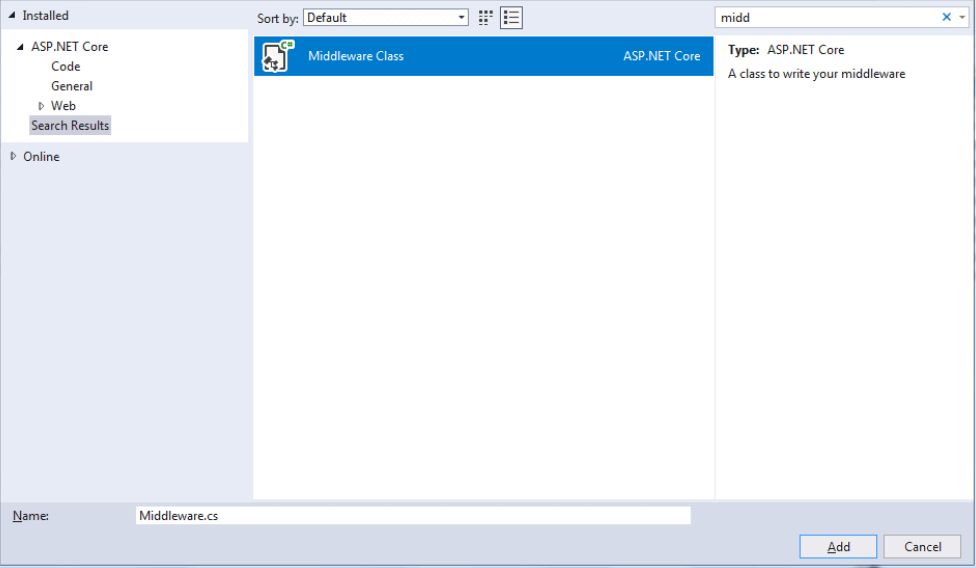
Hello World

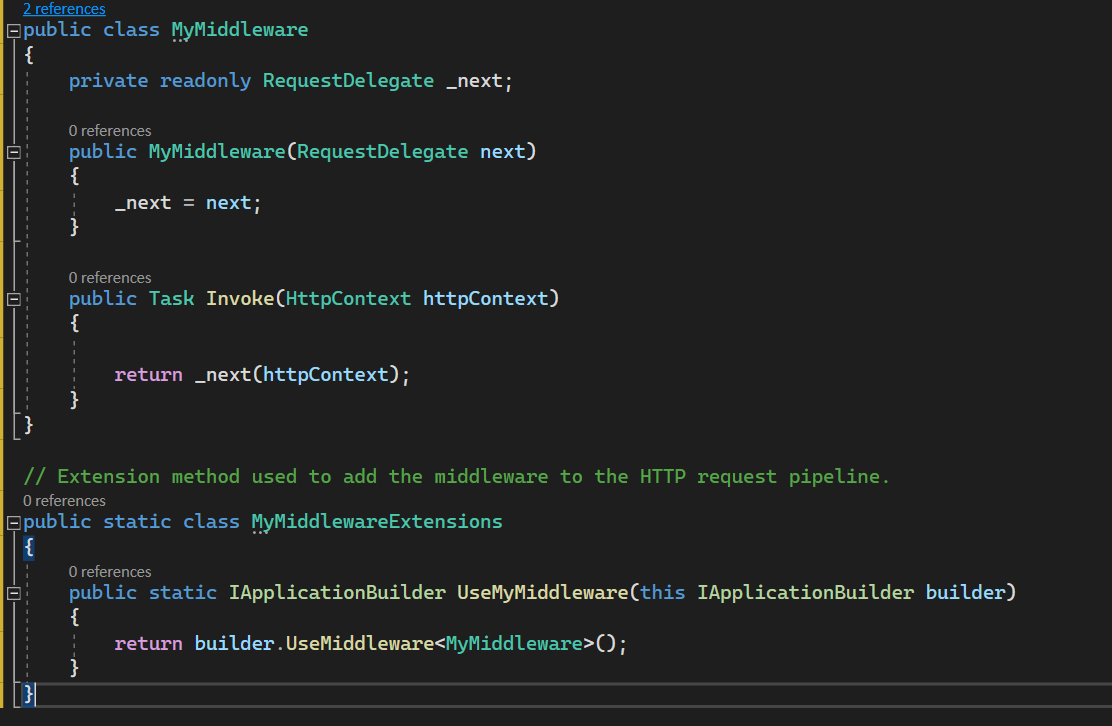
4.middleware2 response

2.middleware1 response

**Custom Middleware:**

The custom middleware component is like any other .NET class with Invoke() method. However, in order to execute next middleware in a sequence, it should have RequestDelegate type parameter in the constructor.

****

****

We can add middleware using app.UseMiddleware<MyMiddleware>() method of IApplicationBuilder also.

**Services:**

**a. AddControllers [web api]:**

Supports: controllers,model binding, API explorer, auth, cors, validation, format mapping

Does not support: Antiforgery, TempData, Views, Pages, TagHelpers, Memory cache etc.

**b. AddControllerWithViews [MVC]:**

Supports: controllers,model binding, API explorer, auth, cors, validation, format mapping, Antiforgery, TempData, Views

Does not support: Pages, TagHelpers, Memory cache etc.

**c. AddRazorPages [classic page based application]:**

Supports: controllers, model binding, API explorer, auth, validation, Antiforgery, TempData, Views, Pages, TagHelpers, Memory cache etc.

Does not support: Cors, format mapping.

**d. AddMvc:**

Supports: controllers,model binding, API explorer, auth, cors, validation, format mapping, Antiforgery, TempData, Views, Pages, TagHelpers, Memory cache etc

**e. AddMvcCore [basic MVC services]:**

**Custom Service:**

a. DbContext type class

AddDbContext<T>() [Microsoft.EntityFramrworkCore]

b. Configuration:

builder.Configuration.AddX()

c. custom services:

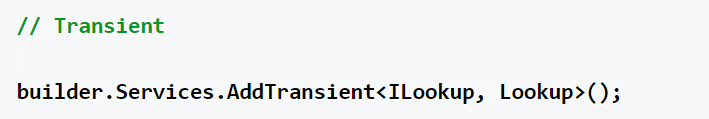
builder.Services.***AddTransient***<ITransientOperationService, OperationService>();

builder.Services.***AddScoped***<IScopedOperationService, OperationService>();

builder.Services.***AddSingleton***<ISingletionOperationService, OperationService>();

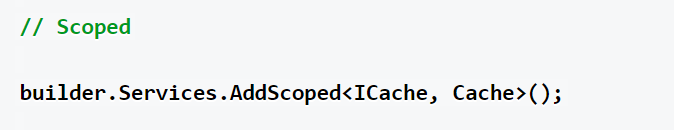
**AddTransient:**

With a transient service, a new instance is provided every time a service instance is requested whether it is in the scope of the same HTTP request or across different HTTP requests. Basically, for every request, a new service instance is provided. Transient creates a new instance for every service/controller as well as for every request and every user.



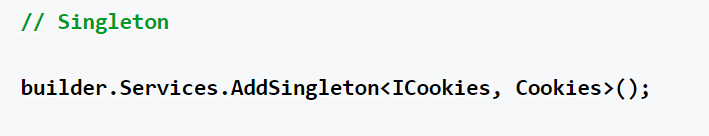
**AddScoped:**

In a scoped service, with every HTTP request, we get a new instance. However, within the same HTTP request, if the service is required in multiple places, like in the view and in the controller, then the same instance is provided for the entire scope of that HTTP request. But every new HTTP request will get a new instance of the service. Add scoped specifies that a single object is available per request.



**AddSingleton:**

AddSingleton() creates a single instance of the service when it is first requested and reuses that same instance in all the places where that service is needed. Basically, it remains the same for all instances of service, throughout the application, and for each subsequent request



**Server Hosting:**

1. ASP.NET core **InProcess** hosting model: IISExpress

2. ASP.NET Core **OutOfProcess** hosting model:

a. an internal web server (Kestrel)

b. an external web server (IIS, Apache, Nginx)

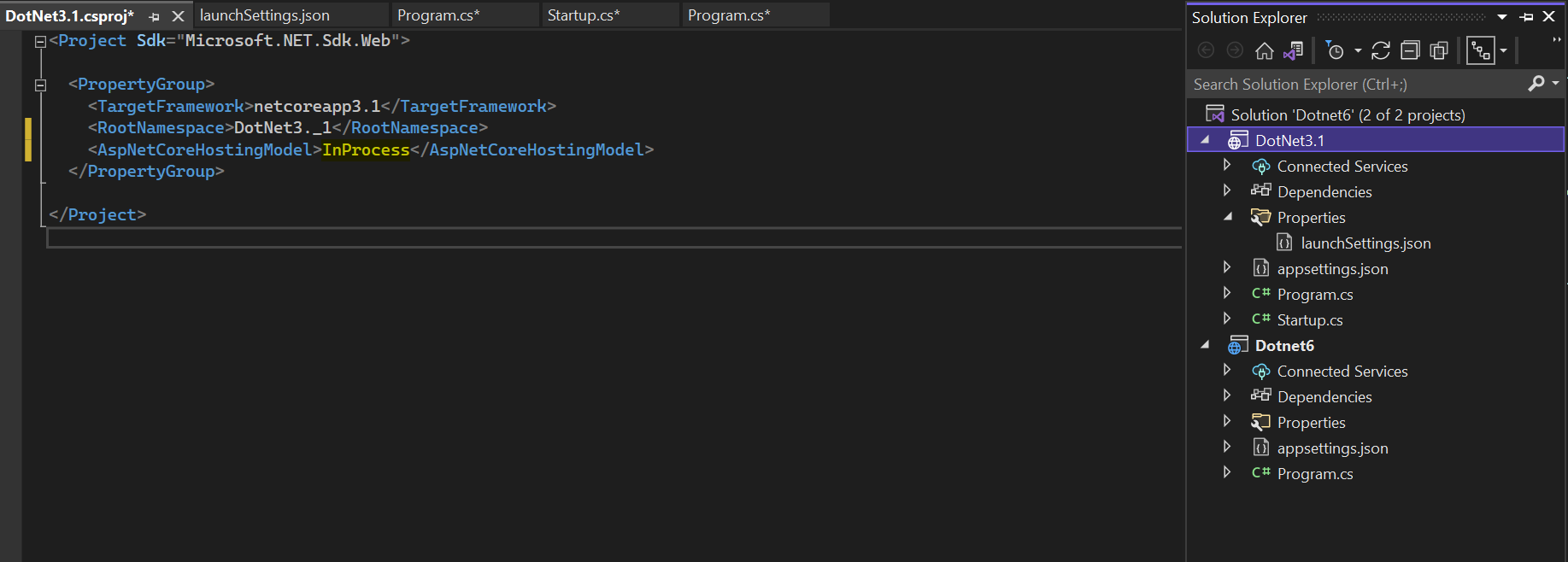
**Kestrel web server**: Internet facing web server, cross platform

**In-process hosting model:**

* Hosting an ASP.NET Core app inside of the IIS worker process i.e w3wp.exe, called the in-process hosting model.
* Also, we need to remember here that we cannot directly run the .NET Core application in IIS.
* We need to install a separate module i.e ASP.NET Core Module and this module allows ASP.NET Core apps to run behind IIS.

Diagram

Description automatically generated



**Out-of-process Hosting:**

* In Out-of-process hosting model, the ASP.NET Core application is hosted outside the IIS worker process by using a Kestrel server.
* Here we can either use the Kestrel server as an Internet-facing web server that can receive a request directly from the internet or we can use it as a reverse proxy server that runs behind a powerful Internet-facing web server such as IIS. Here the reverse proxy server receives HTTP requests from the
* **1: Kestrel server expose as Internet-facing web server (***So here Kestrel Server can receive request directly from the client and send response back to the client***)**
* **2: Kestrel server used with a reverse proxy server (***So here Kestrel Server can not receive request directly from the client , so the client sends the request to the reverse proxy server and then reverse proxy server send it to Kestrel server and similarly Kestrel server sends the response back to the proxy server and then proxy server send it back to the client***)**

Diagram

Description automatically generated

