**Configuring the Startup Class in ASP.NET Core**

The Startup class is a very important class for any type of ASP.NET Core Web Application. This class provides two important methods are as follows:

1. **ConfigureService**
2. **Configure**

**ConfigureService Method:**

The ConfigureService method is used to configure all the services that you want to use in this application. The ConfigureService method takes IServiceCollection as an input parameter. The IServiceCollection interface belongs to Microsoft.Extensions.DependencyInjection namespace. So, whenever we inject any kind of method to this IServiceCollection object that means we are injecting the service to the built-in dependency injection.

**Configure Method:**

The Configure method is used to configure the HTTP request processing pipeline of the application. In other words, we can say that it will configure all the middleware that you want to use in your application. The Configure method takes two parameters i.e. IApplicationBuilder and IWebHostEnvironment instance. The IApplicationBuilder interface belongs to Microsoft.AspNetCore.Builder namespace and IWebHostEnvironment interface belong to Microsoft.AspNetCore.Hosting namespace.

The IApplicationBuilder Defines a class that provides the mechanisms to configure an application’s request pipeline. On the other hand, the IWebHostEnvironment Provides information about the web hosting environment an application is running in.

**Modifying Startup class:**

Please modify the Startup class as shown below. As you can see here, we just define the two methods ConfigureService and Configure and also include the respective namespaces.

**using** *Microsoft.AspNetCore.Builder;*

**using** *Microsoft.AspNetCore.Hosting;*

**using** *Microsoft.Extensions.DependencyInjection;*

**namespace** *ConsoleToWebAPI*

**{**

**public** **class** Startup

**{**

**public** **void** ConfigureServices**(**IServiceCollection services**)**

**{**

**}**

**public** **void** Configure**(**IApplicationBuilder app, IWebHostEnvironment env**)**

**{**

**}**

**}**

**}**

**Add Default Route:**

Once we define the ConfigureService and Configure methods, then we need to add the Route. Routing is basically a mechanism to access the resources by using a URI (Unique Resource Identifier). Each resource is uniquely identified. So, whenever you type the URL and press the enter button in the URL, then that request is mapped to a particular resource. In ASP.NET Core, if we want to add any kind of functionalities then we need to use Middleware. And as discussed, we need to configure the Middleware inside the Configure method of the Startup class.

So, we need to add a Middleware for Routing. By calling the UseRouting() method on the IApplicationBuilder object (i.e. app) we are configuring the Routing middleware in our application. By calling app.UseRouting() method we are just enabling the routing for our application. But we are not telling any kind of mapping i.e. when the request comes who is going to handle that request. Or in other words, we can say which resource is going to be executed when the request comes from the client.

So, we need to tell the mapping between a URL and a resource. And we can do the same by using the UseEndpoints middleware. Inside the UseEndpoints method, we need to call the MapGet method. And inside the MapGet method, we need to pass the URL. Along with we also need to set the content that we need to return. So, modify the Configure method as shown below.

**public** **void** Configure**(**IApplicationBuilder app, IWebHostEnvironment env**)**

**{**

app.UseRouting**()**;

app.UseEndpoints**(**endpoints =**>**

**{**

endpoints.MapGet**(**"/", **async** context =**>** **{**

**await** context.Response.WriteAsync**(**"Hello From ASP.NET Core Web API"**)**;

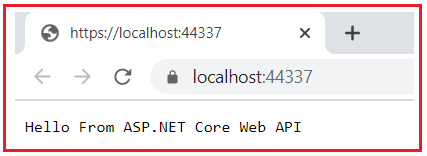
**})**;

**})**;

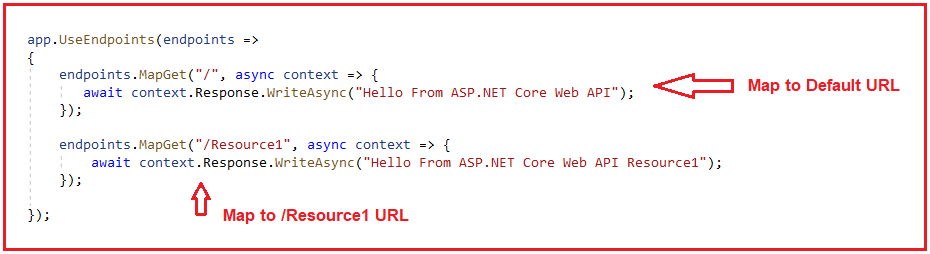
**}**

**Note:**Also include the using Microsoft.AspNetCore.Http namespace as we are using the WriteAsync method.

With the above changes in place, now run the application and you should the message on the browser as shown in the below image.



Let us add another endpoint to our application. Here, we are specifying the URL as /Resource1 as shown in the below image. With this, we have configured two URLs.



The Complete code of the Startup.cs class is given below.

**using** *Microsoft.AspNetCore.Builder;*

**using** *Microsoft.AspNetCore.Hosting;*

**using** *Microsoft.AspNetCore.Http;*

**using** *Microsoft.Extensions.DependencyInjection;*

**namespace** *ConsoleToWebAPI*

**{**

**public** **class** Startup

**{**

**public** **void** ConfigureServices**(**IServiceCollection services**)**

**{**

**}**

**public** **void** Configure**(**IApplicationBuilder app, IWebHostEnvironment env**)**

**{**

app.UseRouting**()**;

app.UseEndpoints**(**endpoints =**>**

**{**

endpoints.MapGet**(**"/", **async** context =**>** **{**

**await** context.Response.WriteAsync**(**"Hello From ASP.NET Core Web API"**)**;

**})**;

endpoints.MapGet**(**"/Resource1", **async** context =**>** **{**

**await** context.Response.WriteAsync**(**"Hello From ASP.NET Core Web API Resource1"**)**;

**})**;

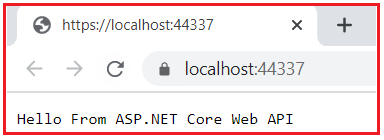
**})**;

**}**

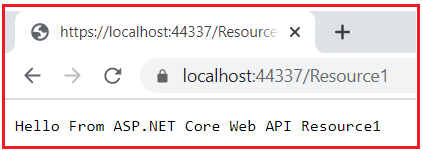
**}**

**}**

Now run the application and see the output. With the default URL, you will get the below output.



When we add /Resource1 to the URL, you will get the below output as expected.



This is what you will get when you create an empty ASP.NET Core Web Application. Again we need to add some features in order to make this application an ASP.NET Web API application

**How to use Web API Service?**

In ASP.NET Core, we can create different types of Web Applications such as MVC, Razor Pages, and Web API Applications. And for each application type, ASP.NET Core provided some set of services that should be included in the application.

So, the service that you want in your application should be injected in the ConfigureService method of the Startup class. We have different service like AddMvc(), AddControllers(), AddRazorPages(), and AddControllersWithViews() that we can inject into the dependency injection container. Now the question is which one we need to use or what are the differences between AddMvc, AddControllers, AddRazorPages, and AddControllersWithViews.

**Differences between AddMvc, AddControllers, AddRazorPages, and AddControllersWithViews:**

1. **AddControllers**: The AddControllers method is used only for Web APIs. That means if you want to create a Web API Application where there are no views, then you need to use the AddController extension method.
2. **AddControllersWithViews**: The AddControllersWithViews method is used to support MVC Based Applications. That means If you want to develop a Model View Controller (i.e. MVC) application then you need to use AddControllersWithViews() extension method.
3. **AddRazorPages**: The AddRazorPages method is used for the Razor Pages application. That means if you want to work with the Razor Page application, then you need to use the AddRazorPages() extension method
4. **AddMvc:**The AddMvc method adds all the services that are required for developing any type of application. That means we can say if we are using the AddMvc method, then we can develop Web API, MVC, and Razor Pages applications.

**Note:** Adding AddMvc() method will add extra features to your application even though they are not required which might impact the performance of the application.

**Adding Web API Service:**

As we are going to deal with ASP.NET Core Web API, so, we need to configure the **AddControllers** extension method to the built-in dependency injection (services) in the ConfigureService method of the Startup class. So, please modify the ConfigureService method of the Startup class as shown in the below code.

**public** **void** ConfigureServices**(**IServiceCollection services**)**

**{**

services.AddControllers**()**;

**}**

With the above changes in place, we configure the services required for ASP.NET Core Web API Application. But that’s also not enough.