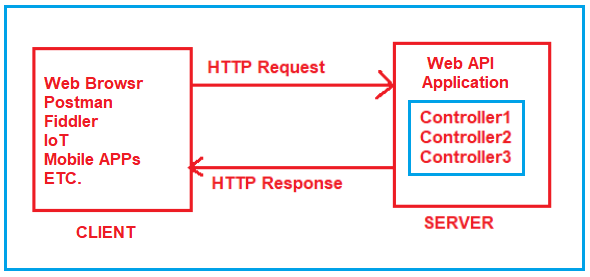
**What is Routing in ASP.NET Core?**

Let us understand Routing with an example. Please have a look at the following image. On the right-hand side, we have the server and within the server, we have deployed our ASP.NET Core Web API application. And inside the ASP.NET Core Web API Application, assume that we have three controllers and each controller contains some action method. On the left-hand side, we have the client. The client can be a Web Browser, Postman, Fiddler, Swagger, IoTs, Mobile APP, etc. Suppose we are sending a request from the client to the browser. As we already discussed, first, the request has to go through the request processing pipeline and if everything is fine, then the ASP.NET Core Framework navigates that to the controller action method, and based on the processing of that action method, the respective client will get the response.



Here, the important that we need to understand is, how the application will come to know which request will be mapped to which controller action method.

Basically, the mapping between the URL and resource is nothing but the concept or Routing.

**How does the Routing work in ASP.NET Core Web API?**

Routing in ASP.NET Core Web API application is the process of mapping the incoming HTTP Request (URL) to a particular resource i.e. controller action method.

For the Routing Concept in ASP.NET Core Web API, we generally set some URLs for each resource. When we run the application, then it will create the Route table and the Route table will contain the mapping information between the URL and the Resource. So, when we are sending a request from the client to the server, then the application will check the URL in the Route table and if it found an exact, then the application will forward the request to that particular resource else it will throw an error saying resource not found.

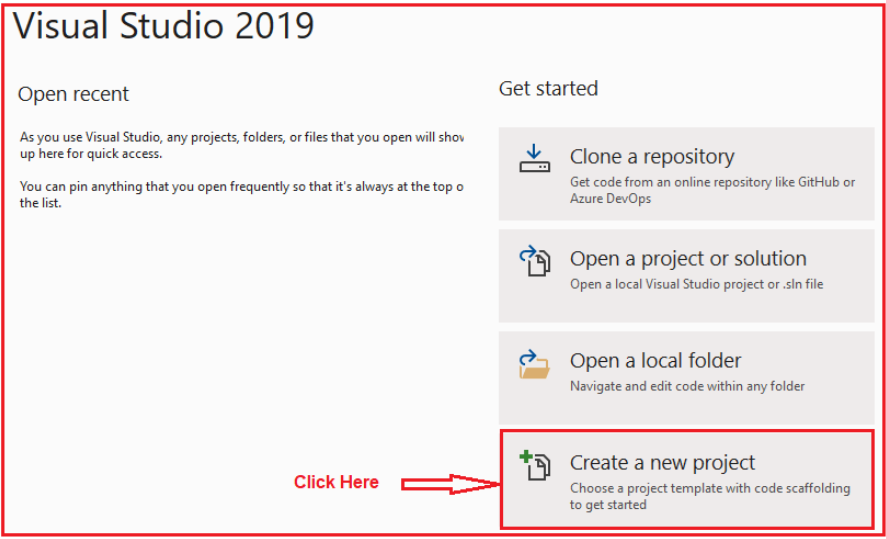
We can access any resource using a unique URL in ASP.NET Core Web API Application. It is also possible that a resource can have multiple unique URLs. But multiple resources can not have the same URL and if you do so, then the application gets confused to invoke which action method and as a result, you will get an ambiguity error.

So, the ASP.NET Core Framework maps the incoming HTTP Requests i.e. URLs to the action methods of Controllers based on the routes that are configured for your application. In ASP.NET Core, it is also possible to configure multiple routes, and also it is possible to set some specific configurations such as default values, constraints, message handlers, etc for each route. If this is not clear at the moment then don’t worry, we will discuss each and everything with examples.

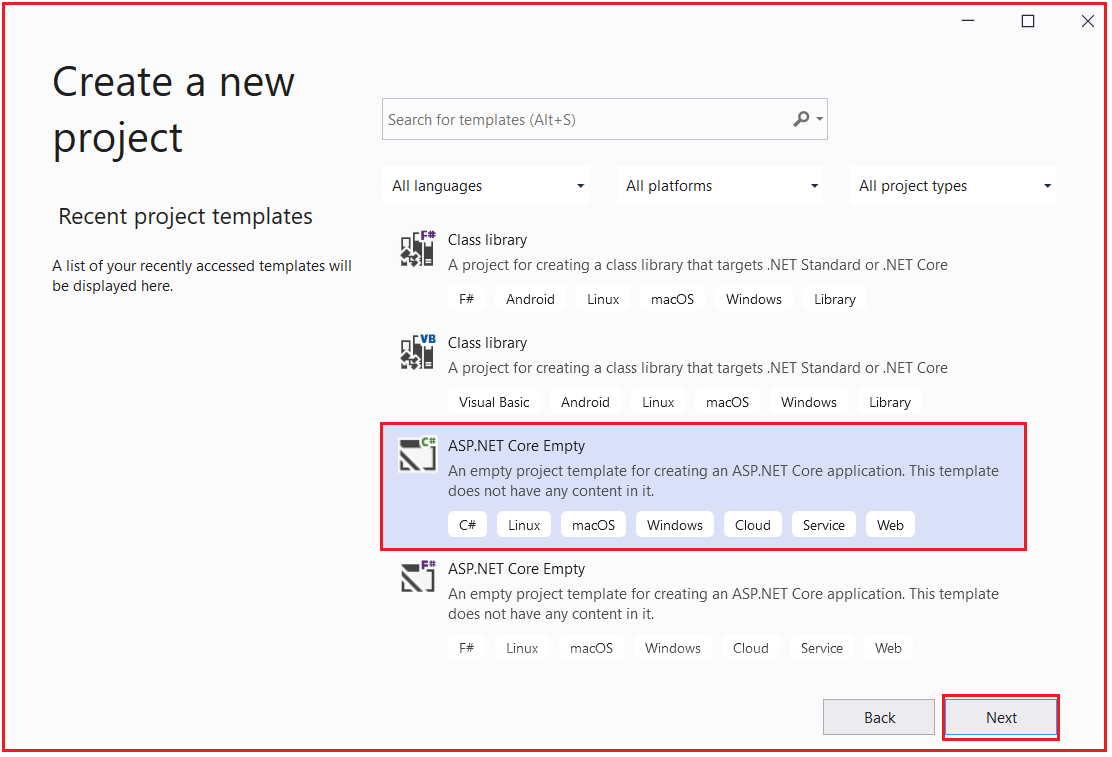
**How to enable Routing in ASP.NET Core Web API?**

Let us understand How to enable Routing in ASP.NET Core Web API Application with an example. Here, we are going to explain everything from scratch. So, create an empty ASP.NET Core Application by following the below steps.

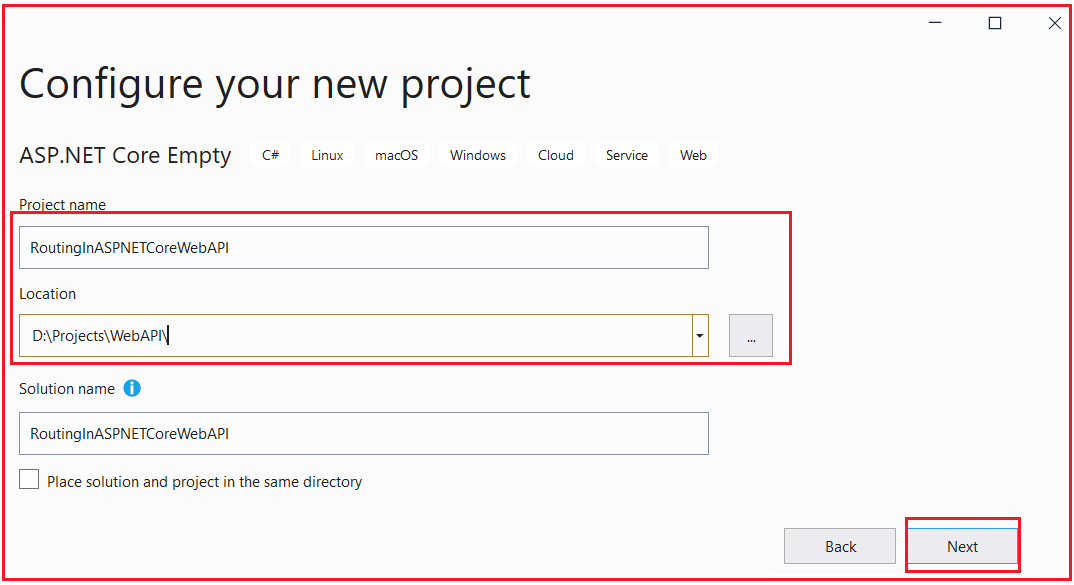
First, open Visual Studio 2019 and then click on the Create a new project option as shown in the below image.



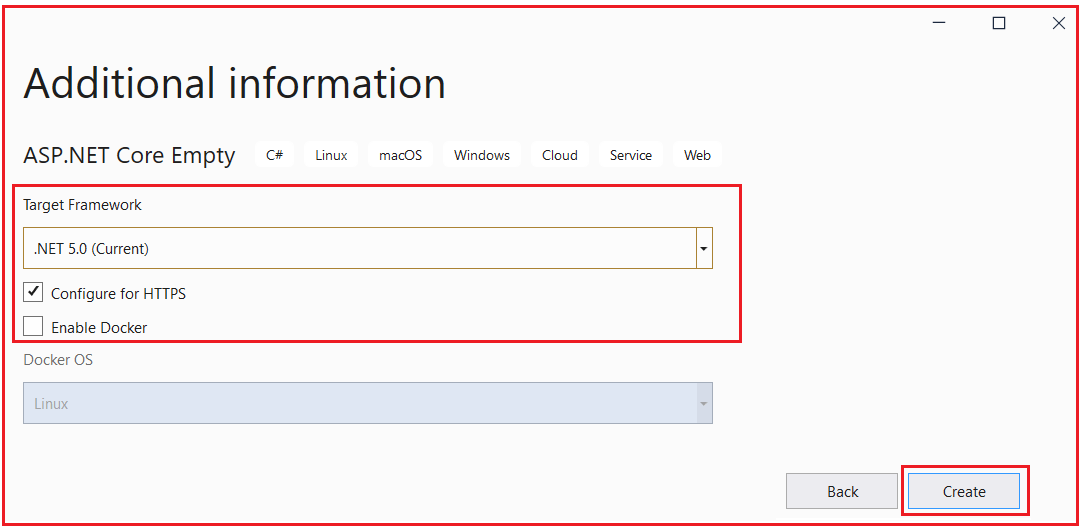
Once you click on the Create a new project option, it will open the Create a new project window. From this window select the ASP.NET Core Empty Project template which uses the programming language as C# and finally, click on the Next button as shown in the below image.



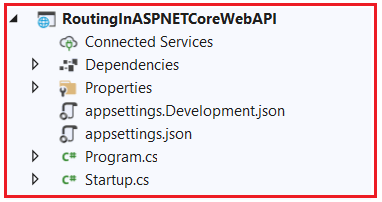
Once you click on the Next button, it will open configure your new project window. Here, you need to specify the Project name (RoutingInASPNETCoreWebAPI) and the location where you want to create the project. And then click on the Next button as shown in the below image.



Once you click on the Next button, it will open the Additional Information window. Here, I am going with the default configuration, and please make sure to select the Target Framework as .NET 5.0 and click on the Create button as shown in the below image.



Once you click on the Create button, it will create the ASP.NET Core Empty Project with the following file and folder structure.



**Adding ASP.NET Core Web API Service:**

Now, we have created an Empty ASP.NET Core Application. Let us add the ASP.NET Core Web API services so that our application will support the ASP.NET Core Web API Features. To do so we need to inject the ASP.NET Core Web API Service to the built-in IoC Container as **services.AddControllers();** You can do the same using the ConfigureService method of the Startup class as shown in the below code.

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

**{**

services.AddControllers**()**;

**}**

**Enabling Routing in ASP.NET Core Web API Application:**

In ASP.NET Core Web API Application, we can enable the Routing through Middleware. In order to enable Routing in ASP.NET Core, we need to add the following two middleware components to the HTTP Request processing Pipeline.

1. **UseRouting():** The UseRouting Middleware only enables the Routing for your application. This will not map any URL to any resource.
2. **UseEndpoints():** This middleware will map the URL to the resource. But the most important point that you need to remember is, the action methods are not only the resource that you can map. You can also map static file resources to a URL. But here we are only focusing on ASP.NET Core Web API and hence we are going to map the URL to action methods.

**Configuring the Routing Middlewares in ASP.NET Core:**

As we already discussed, if we want to configure any middleware then we need to configure the same inside the Configure method of the Startup class. So, let us modify the Configure method of the Startup class as shown below to configure the UseRouting and UseEndpoints Middlewares which will enable Routing as well as the mapping between the URL and Resource. As you can see in the below code within the UseEndpoints Middleware we write MapControllers which will actually set the mapping between URL and Controller action method. So, the Map controller will act as a bridge between the HTTP Request and the Controllers.

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

**{**

**if** **(**env.IsDevelopment**())**

**{**

app.UseDeveloperExceptionPage**()**;

**}**

app.UseRouting**()**;

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

**{**

endpoints.MapControllers**()**;

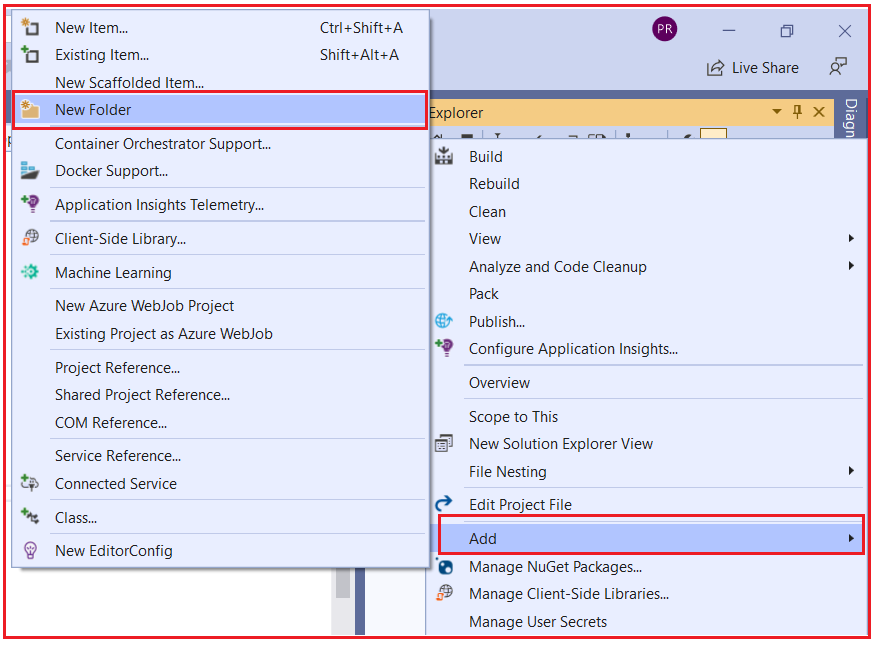
**})**;

**}**

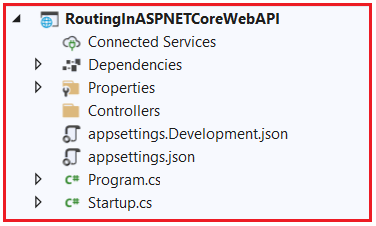
In ASP.NET Core Application, we have two ways to define the Routing i.e. Conventional Based and Attribute-Based Routing. And Attribute Routing is the most preferred way of defining routes in ASP.NET Core Web API Application and hence, we are only going to focus on Attribute Routing in this course.

**Adding Controller:**

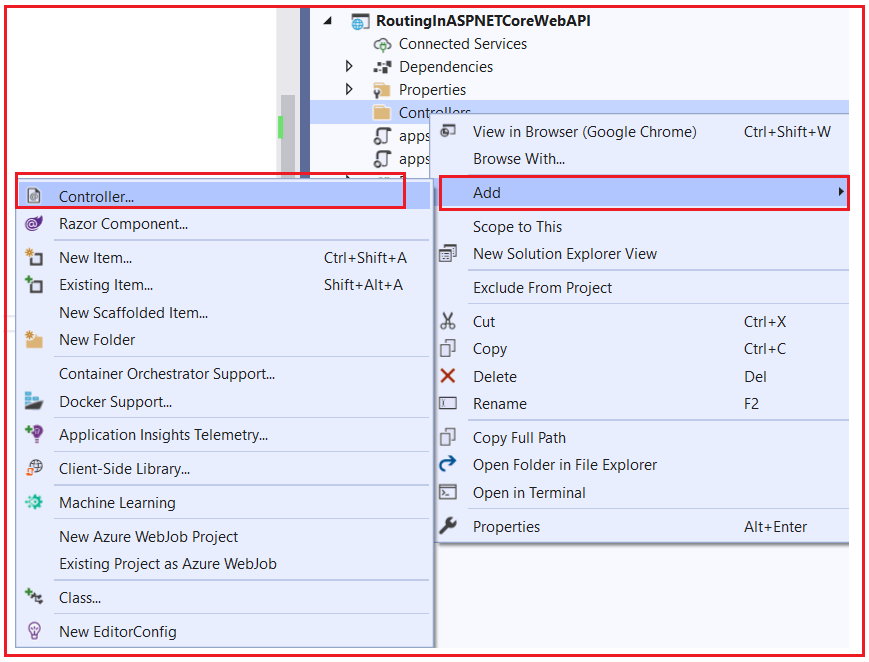
As of now, we have added the ASP.NET Core Web API Service and set the Routing for our application. Now let us add a controller i.e. HomeController to our application. The Controllers in the ASP.NET Core Application should be added inside the Controllers folder in your project. So, let us first add the Controllers folder to the project root directory. To do so, right-click on your project and then select **Add => New Folder** option from the context menu as shown in the below image.



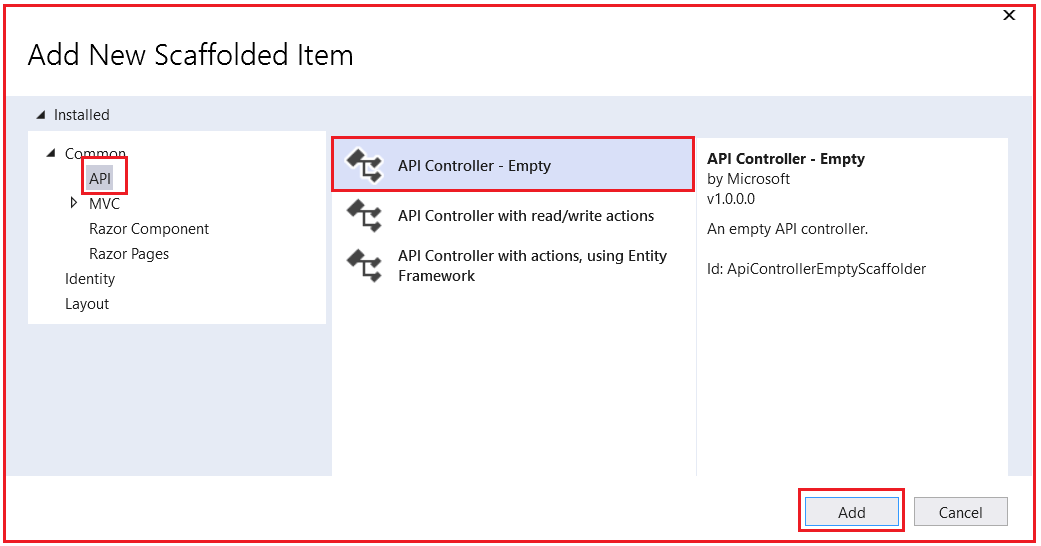
Then rename the folder name as Controllers. Once the controller folder is added your project structure should looks as shown in the below image.



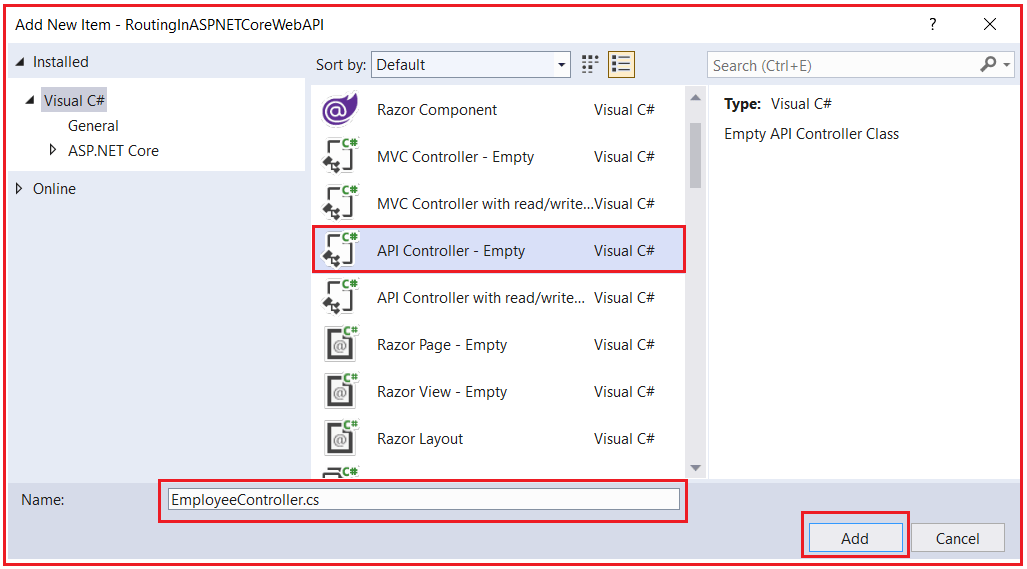
Let’s add the Home controller within the Controllers folder. To do so, right-click on the Controllers folder and then select Add => Controller from the context menu as shown in the below image.



It will open the following Add New Scaffolded item window. Here, from the left side, select API, and from the middle pane select API Controller – Empty and click on the Add button as shown in the below image. As we are going to do everything from scratch, so here we are selecting the Empty Controller template. In our upcoming articles, we are also going to discuss the other two templates.



From the next window, provide the controller name as EmployeeController and click on the Add button as shown in the below image.



Once you click on the Add button, it will add a class file with the name EmployeeController.cs within the Controllers folder. Now, you can find the following default code inside the EmployeeController.cs class file.

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

**using** *Microsoft.AspNetCore.Mvc;*

**using** *System;*

**using** *System.Collections.Generic;*

**using** *System.Linq;*

**using** *System.Threading.Tasks;*

**namespace** *RoutingInASPNETCoreWebAPI.Controllers*

**{**

**[**Route**(**"api/[controller]"**)]**

**[**ApiController**]**

**public** **class** EmployeeController : ControllerBase

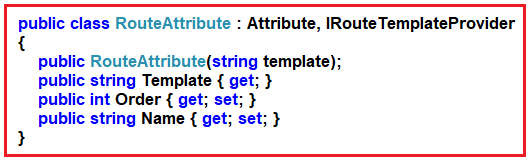
**{**

**}**

**}**

**Understanding Route Attribute in ASP.NET Core:**

Using the Route Attribute, we can define the Routes in ASP.NET Core Application either at the Controller level or at the action method level. Before using the Route Attribute lets us have a look at the signature of the Route Attribute. The following image shows the definition of the Route Attribute.



As you can see in the above image, Route Attribute is basically a class inherited from the Attribute class and IRouteTemplateProvider interface. The Constructor of the RouteAttribute class takes the template as an input parameter which is nothing but the URL that you are excepting from the client and it cannot be null.

**Adding Attribute Routing in ASP.NET Core Web Application:**

Now let us add two action methods within the EmployeeController class. Now, don’t concentrate on the return type and the data that we are returning from the action method, rather concentrate on the Routing concept. We want to invoke the GetAllEmployees method with the URL **/Emp/All** and GetEmployeeById method with the URL **/Emp/ById**. To achieve this, we can use the Route Attribute and decorate the action GetAllEmployees and GetEmployeeById method as [Route(“Emp/All”)] and [Route(“Emp/ById”)] respectively. So, modify the EmployeeController class as shown below.

**using** *Microsoft.AspNetCore.Mvc;*

**namespace** *RoutingInASPNETCoreWebAPI.Controllers*

**{**

**[**ApiController**]**

**public** **class** EmployeeController : ControllerBase

**{**

**[**Route**(**"Emp/All"**)]**

**public** string GetAllEmployees**()**

**{**

**return** "Response from GetAllEmployees Method";

**}**

**[**Route**(**"Emp/ById"**)]**

**public** string GetEmployeeById**()**

**{**

**return** "Response from GetEmployeeById Method";

**}**

**}**

**}**

Now everything is ready. So, run the application and access the action method using the URL we configured as shown in the below image.

