**Adding Controller Class in ASP.NET Core**

In this article, I am going to discuss **How to Add a Controller to ASP.NET Core Web API Application**. This is the last step of converting ASP.NET Core Console Application to ASP.NET Core Web API Application. So, before proceeding to this article, please read [**Step1**](https://dotnettutorials.net/lesson/build-asp-net-core-web-api-project/), [**Step2**](https://dotnettutorials.net/lesson/adding-web-host-builder/), and [**Step3**](https://dotnettutorials.net/lesson/configuring-startup-class/) articles.

**Adding Controller Class in ASP.NET Core Web API**

As of now, we have done the very basic configuration that is required to convert a console application to a Web API Application. Now we are going to discuss the most important concept of Web API i.e. Controllers.

**Points to Remember while working with Controller class in ASP.NET Core Web API**:

1. The Controller class in ASP.NET Core Web API must have a “Controller” suffix. For example, if you want to add a controller with the name Home, then the name of the controller should be HomeController. similarly, if you want to add a controller for Student, then the name should be StudentController.
2. The Controller class must be inherited from the ControllerBase class. If you are coming from an ASP.NET Core MVC background, then, in that case, the Controller class is inherited from the Controller class.
3. We need to decorate the controller class with the ApiController attribute.
4. We also need to use Attribute Routing to access the resource using URI.

**What is the difference between the Controller and ControllerBase class in ASP.NET Core?**

The Controller class that we used in ASP.NET Core MVC has support Views, ViewBag, ViewData, TempData, etc. But here in ASP.NET Core Web API, we don’t require such concepts. So, we skip controller class and use the ControllerBase class.

The point that you need to remember is the ControllerBase class is also serves as the base class for the Controller class. That means behind the scene the Controller class is inherited from the ControllerBase class.

So, if you are using the Controller class in your ASP.NET Core MVC application, it means you are also using ControllerBase class. But in Web API, we simply need to use the ControllerBase class.

**ControllerBase Class in ASP.NET Core Web API:**

The ControllerBase Class in ASP.NET Core provides many methods and properties to handle HTTP Requests and Responses. For example, if you want 200 Status codes from your action method, this provides one method. Similarly, if you want to return the 201 status code from your controller action method, it is provided with another method.

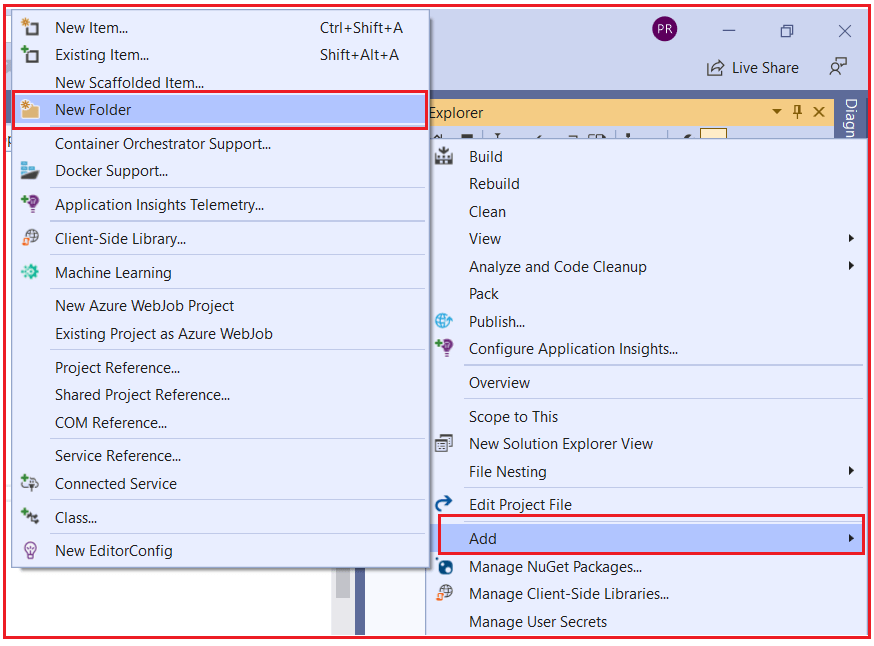
**ApiController Attribute in ASP.NET Core Web API:**

It Indicates that a type and all derived types are used to serve HTTP API responses. Controllers decorated with this attribute are configured with features and behavior targeted at improving the developer experience for building APIs. When decorated on an assembly, all controllers in the assembly will be treated as controllers with API behavior. Some of the features are as follows:

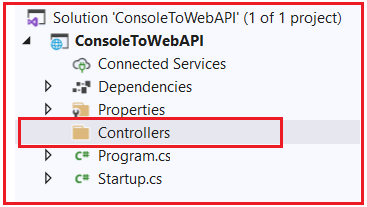
1. Attribute Routing Requirement
2. Handle the client error i.e. 400 HTTP status code.
3. Multipart/form-data request inference
4. Binding the incoming data with the parameters using the Model Binding concepts

**Adding Controller in ASP.NET Core Web API:**

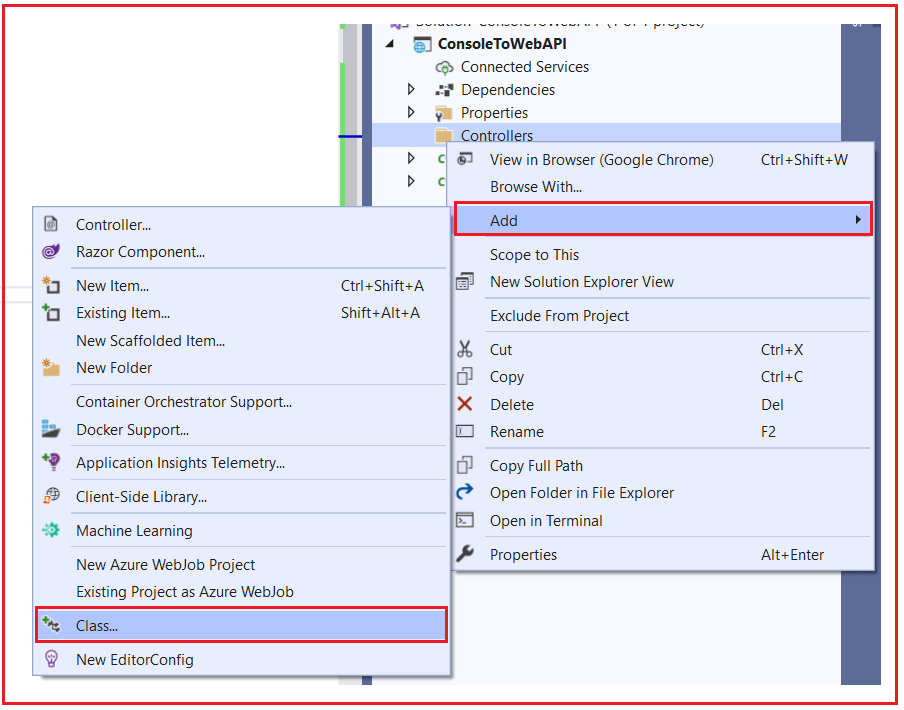
The Controller should be added inside the Controllers folder in your project. So, let us first add the Controllers folder to the project root directory. To do so, right-click on your project and then select Add => New Folder option from the context menu as shown in the below image.



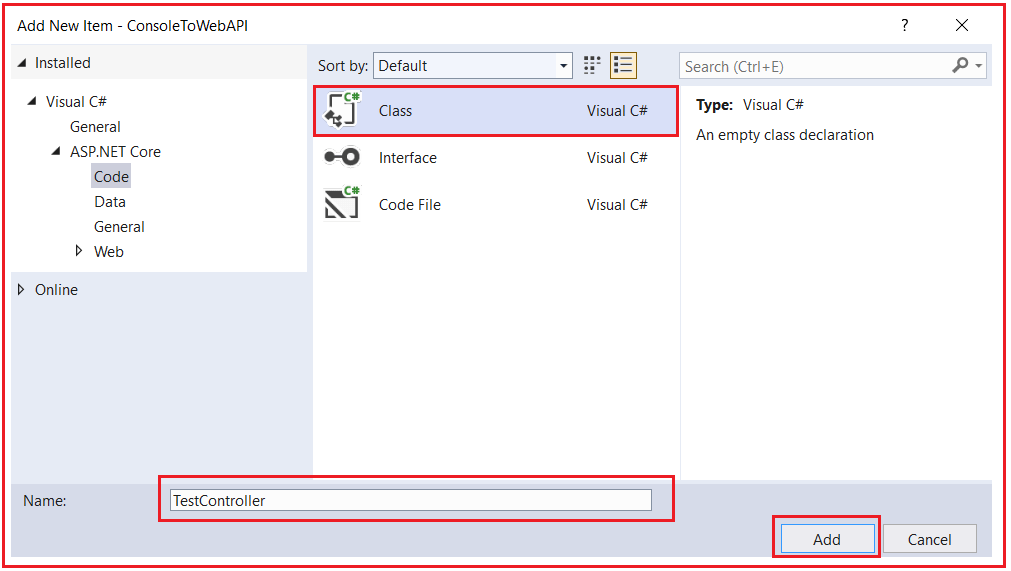
Then rename the folder as Controllers. Once the controller folder is added your project structure should looks as shown in the below image.



Let’s add a controller with the name Test. To do so, right-click on the Controllers folder and then select Add => Class from the context menu as shown in the below image.



From the next Add new item window, select class template and provide the class name as TestController and click on the Add button as shown in the below image which will add a simple class inside the Controllers folder.



**Modifying the TestController class:**

First, we need to inherit the TestController class from the ControllerBase class and ControllerBase class present in Microsoft.AspNetCore.Mvc namespace. Then decorate the Controller with ApiController and Route attribute. A Controller class can have multiple action methods. Let add one action method which simply returns a string. So, modify the TestController class as shown in the below code.

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

**namespace** *ConsoleToWebAPI.Controllers*

**{**

**[**ApiController**]**

**[**Route**(**"test"**)]**

**public** **class** TestController : ControllerBase

**{**

**public** string Get**()**

**{**

**return** "Returning from TestController Get Method";

**}**

**}**

**}**

**Mapping HTTP Request to Web API Controller:**

Now we need to tell the ASP.NET Core Framework to map the incoming HTTP Request to our Web API Controller. To do so, modify the Configure method of the Startup class as shown in the below code. Here, we are using the built-in MapControllers extension method to map the incoming HTTP Request to our Controllers.

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

**{**

app.UseRouting**()**;

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

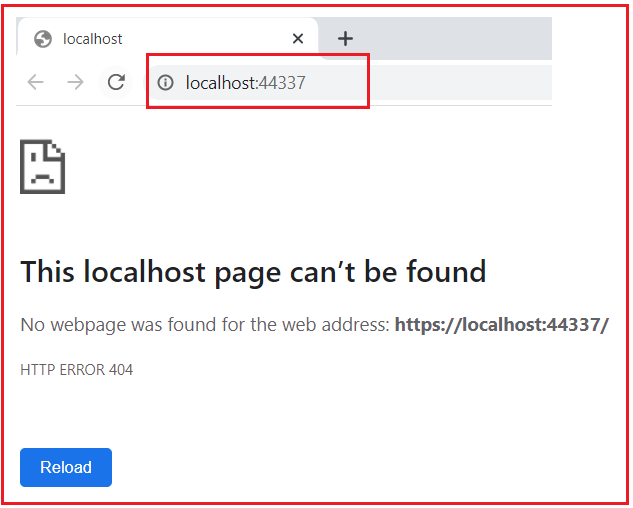
**{**

endpoints.MapControllers**()**;

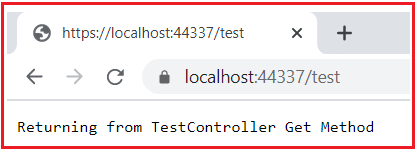
**})**;

**}**

With the above changes in place, now run the application and you will get the following 404 error page. This is because at this point there is no resource for this request



Let’s append /test at the end of the URL and you should get the data as expected as shown in the below image.



**What happens if we add one more resource into our Controller class?**

Here resource means method. Let’s add one method into the Test Controller class as shown in the below code.

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

**namespace** *ConsoleToWebAPI.Controllers*

**{**

**[**ApiController**]**

**[**Route**(**"test"**)]**

**public** **class** TestController : ControllerBase

**{**

**public** string Get**()**

**{**

**return** "Returning from TestController Get Method";

**}**

**public** string Get2**()**

**{**

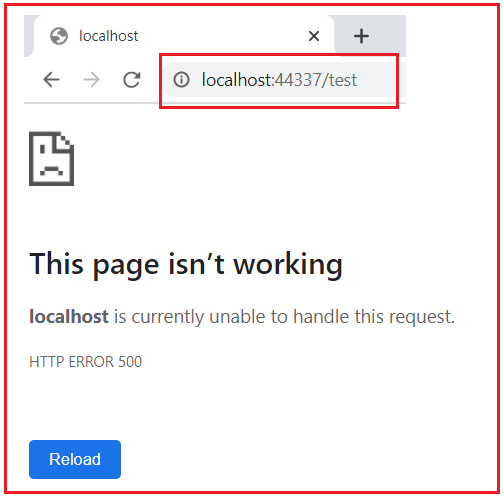
**return** "Returning from TestController Get2 Method";

**}**

**}**

**}**

At this point, if you run the application, then you will get the following.



As you can see in the above image, the actual exception details are not visible to you and the reason for this is we have not added the Developer Exception Page Middleware into the application request processing pipeline. Let us add the Developer Exception Page middleware into our request processing pipeline. So, modify the Configure method of the Startup class as shown in the below code. Here, we checking if the environment is Development, then only show the exception details. Please also include Microsoft.Extensions.Hosting namespace

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

**{**

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

**{**

app.UseDeveloperExceptionPage**()**;

**}**

app.UseRouting**()**;

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

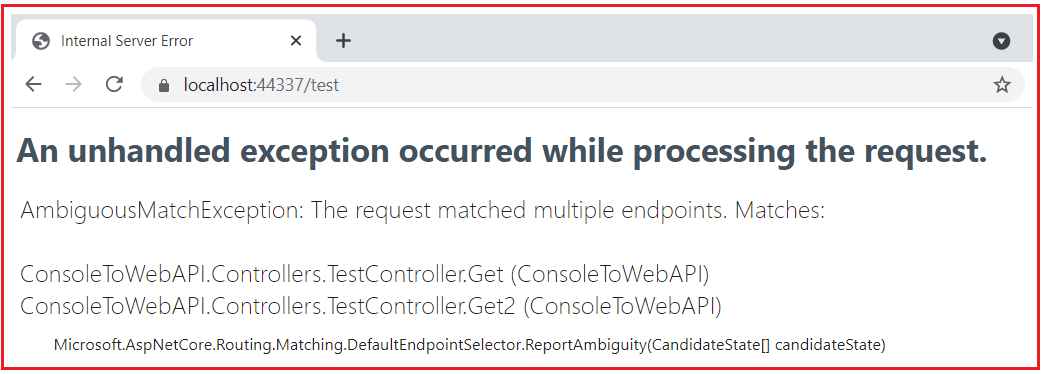
**{**

endpoints.MapControllers**()**;

**})**;

**}**

With the above changes in place, now run the application and navigates to /Test in the URL and you should see the exception details as shown in the below image.



As you can see in the above image, it clearly said that

**AmbiguousMatchException: The request matched multiple endpoints. Matches:**  
**ConsoleToWebAPI.Controllers.TestController.Get (ConsoleToWebAPI)**  
**ConsoleToWebAPI.Controllers.TestController.Get2 (ConsoleToWebAPI)**

So, the application finds two endpoints for the incoming request and gets confused about who is going to serve the request, and hence it gives the above exception.

As per Web API standard, each resource should have a Unique Identifier. Let us do some changes in our Route Attribute so that each request will have a unique URI. In the Route attribute, we specified the action as part of the URI. In our upcoming articles, we will discuss Routing in detail. As of now, just modify the TestController class as shown below.

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

**namespace** *ConsoleToWebAPI.Controllers*

**{**

**[**ApiController**]**

**[**Route**(**"test/{action}"**)]**

**public** **class** TestController : ControllerBase

**{**

**public** string Get**()**

**{**

**return** "Returning from TestController Get Method";

**}**

**public** string Get2**()**

**{**

**return** "Returning from TestController Get2 Method";

**}**

**}**

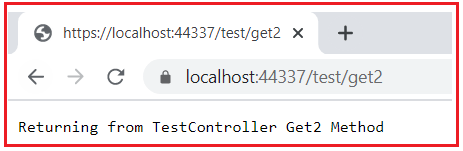
**}**

With the above changes in place, now run the application. Now we have to pass the method name as part of the URL as shown in the below image.

**Test/get**



**Test/Get2**



That’s it. We have done all the steps. Now we have successfully converted an ASP.NET Core Console Application to an ASP.NET Core Web API Application.