## **Route Constraints in ASP.NET Core Web API**

1. **Type: int, double, bool, float, datetime, etc**
2. **Min: min(number)**
3. **Max: max(number)**
4. **Range: range(10. 15)**
5. **Alpha: alpha**
6. **MinLength: minlength(5)**
7. **MaxLength: maxlength(10)**
8. **Length: length(10)**
9. **Required: required**
10. **Regex: regex(expression)**

**ASP.NET Core Web API Attribute Routing with Route Constraints**

The Route Constraints in ASP.NET Core Web API Attribute Routing are nothing but a set of rules that can be applied to routing parameters to restrict how the parameters in the route template are matched. The syntax to use Route Constraints is: **{parameter:constraint}**

**Examples to Understand Route Constraints in ASP.NET Core Web API**

Let us understand ASP.NET Core Web API Attribute Routing Route Constraints with some examples. Please modify the Employee Controller class as shown below.

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

**namespace** *RoutingInASPNETCoreWebAPI.Controllers*

**{**

**[**ApiController**]**

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

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

**{**

**[**Route**(**"{EmployeeId}"**)]**

**public** string GetEmployeeDetails**(int** EmployeeId**)**

**{**

**return** $"Response from GetEmployeeDetails Method, EmployeeId : {EmployeeId}";

**}**

**[**Route**(**"{EmployeeName}"**)]**

**public** string GetEmployeeDetails**(**string EmployeeName**)**

**{**

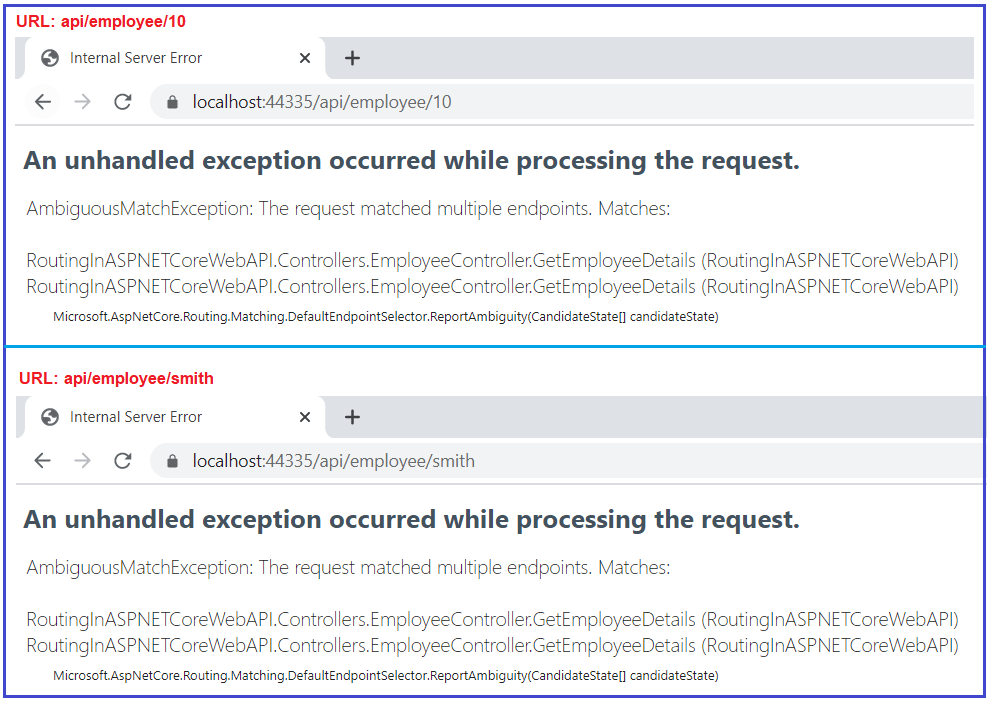
**return** $"Response from GetEmployeeDetails Method, EmployeeName : {EmployeeName}";

**}**

**}**

**}**

With the above changes in place, now the run application and navigate to the URL **api/employee/10** and **api/employee/smith**, and in both cases you will get the following error.



This is because, when the request comes, the application does not identify which version of the GetEmployeeDetails() method to use, and hence it gives an Ambiguous Match Exception as two end pints match the same request. This is the situation where the Route Constraints come into the picture in ASP.NET Core Web API.

**How to use Route Constraint in ASP.NET Core Web API?**

What we want to achieve is, if an integer is specified in the URL like **api/employee/10**, then we need to execute the **GetEmployeeDetails(int EmployeeId)** method which takes an integer parameter whereas if a string is specified in the URL like **api/employee/smith**, then we need to execute the **GetEmployeeDetails(string EmployeeName)** method of the Employee Controller which takes the string as a parameter.

This can be very easily achieved in ASP.NET Core Web API Application using Route Constraints. We need to use the following syntax to specify the route constraint,

**{parameter:constraint}**

**Type Route Constraint in ASP.NET Core Web API:**

We can use the type constraint to specify the parameter type. The different types of type constraints supported in ASP.NET Core Web Application are int, decimal, float, long, double, bool, etc. So, with the type constraint in place, if an integer is specified in the URL, then the GetEmployeeDetails(int EmployeeId) action method is invoked and if a string is specified in the URL then the GetEmployeeDetails(string EmployeeName) method is invoked.

**Example: int type constraint:**

If you want any parameter to accept only integer values then you need to specify the int type constraint. So, let us decorate the GetEmployeeDetails method which takes an integer parameter with the following Route Attribute.

**[Route(“{EmployeeId:int}”)]**

**Note:** we don’t need to specify anything for the string parameter as by default all the parameters in ASP.NET Core Web Application are string only.

**Let’s modify the Employee Controller to use the above-discussed int type Route Constraints as shown below.**

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

**namespace** *RoutingInASPNETCoreWebAPI.Controllers*

**{**

**[**ApiController**]**

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

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

**{**

**[**Route**(**"{EmployeeId:int}"**)]**

**public** string GetEmployeeDetails**(int** EmployeeId**)**

**{**

**return** $"Response from GetEmployeeDetails Method, EmployeeId : {EmployeeId}";

**}**

**[**Route**(**"{EmployeeName}"**)]**

**public** string GetEmployeeDetails**(**string EmployeeName**)**

**{**

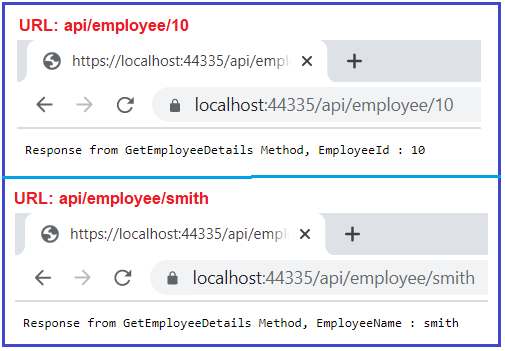
**return** $"Response from GetEmployeeDetails Method, EmployeeName : {EmployeeName}";

**}**

**}**

**}**

With the above changes in place, now run the application and navigate to **api/employee/10** and **api/employee/smith** URLs and you should get the following output.



**Min(number) constraint in ASP.NET Core:**

If you want to apply some minimum value constraint for any parameter then you can use the Min constraint. The min constraint takes one parameter i.e. the minimum value to be applied on the parameter. For example, if you want the GetEmployeeDetails(int EmployeeId) action method to be invoked only if the EmployeeId is a number greater than 1000, then you can use the min(1000) constraint in ASP.NET Core Web API as shown below.

**[**Route**(**"{EmployeeId:int:min(1000)}"**)]**

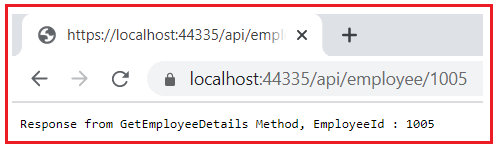
**public** string GetEmployeeDetails**(int** EmployeeId**)**

**{**

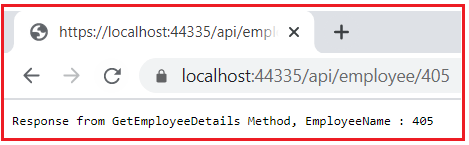
**return** $"Response from GetEmployeeDetails Method, EmployeeId : {EmployeeId}";

**}**

Now run the application and navigate to the URL by passing a value greater than 1000 and you should get the following output.



Now, if you pass a value less than 1000, then the other GetEmployeeDetails method which takes string parameter is executed as shown in the below image.



This is because the second GetEmployeeDetails method takes a string parameter and here it treats the value 405 as a string and executes that method.

**Alpha constraint in ASP.NET Core Web API:**

If you want any parameter to accept only alphabets (a to z characters) values then you need to specify the alpha constraint. So, let us decorate the GetEmployeeDetails method which takes string parameters with the following Route Attribute.

**[Route(“{EmployeeName:alpha}”)]**

Here, alpha stands for uppercase or lowercase alphabet characters. So, changes the GetEmployeeDetails method which takes the string parameter as shown below.

**[**Route**(**"{EmployeeName:alpha}"**)]**

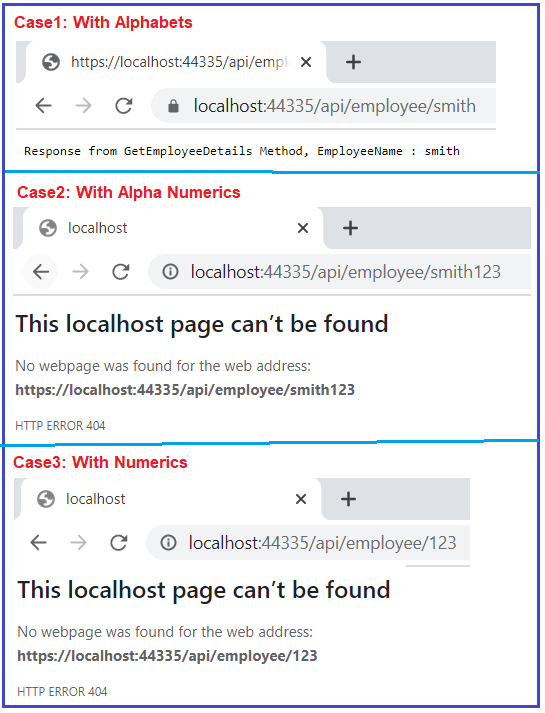
**public** string GetEmployeeDetails**(**string EmployeeName**)**

**{**

**return** $"Response from GetEmployeeDetails Method, EmployeeName : {EmployeeName}";

**}**

With the above changes in place, now run the application and test the below three cases. Case1: with only alphabets and it should work. Case2: with alphanumeric and should not work. Case3: with an integer value less than 1000 and it should not work. All these three cases are shown in the below image.



**Max(Number) constraint in ASP.NET Core Web API:**

Along with the min constraint, you can also specify the max constraint in ASP.NET Core Web API. The max constraint also takes one parameter which is used to specify the max value that can be applied to the parameter. For example, if you want the EmployeeId in the URL should not to be greater than 1000, then you can use the max constraint as shown below.

**[**Route**(**"{EmployeeId:int:max(1000)}"**)]**

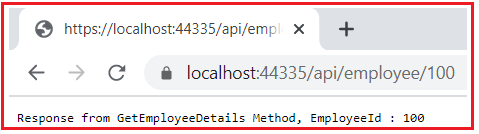
**public** string GetEmployeeDetails**(int** EmployeeId**)**

**{**

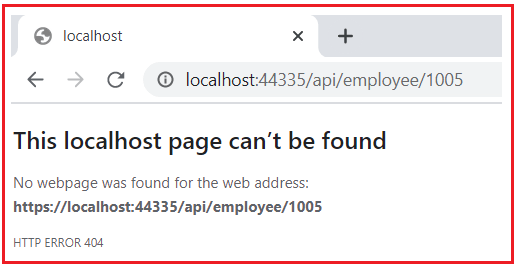
**return** $"Response from GetEmployeeDetails Method, EmployeeId : {EmployeeId}";

**}**

Now save the changes and run the application and navigate to the URL by passing a value less than 1000 and you should get the following output.



Now, if you pass a value greater than 1000, then you will get the following resource not found 404 error page.



It is also possible to use both min and max constraints for a single route. For example, if you want the EmployeeId value in the URL to be between 100 and 1000 inclusive, then we can specify both min and max constraints as shown below.

**[**Route**(**"{EmployeeId:int:min(100):max(1000)}"**)]**

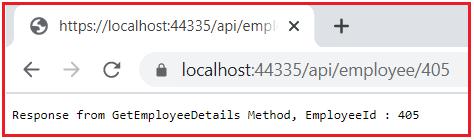
**public** string GetEmployeeDetails**(int** EmployeeId**)**

**{**

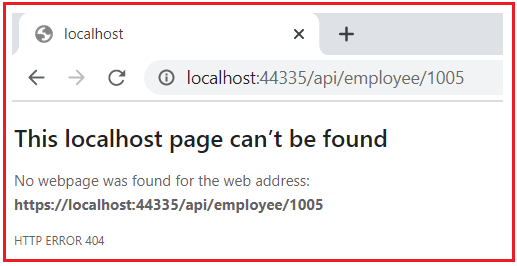
**return** $"Response from GetEmployeeDetails Method, EmployeeId : {EmployeeId}";

**}**

Now save the changes and run the application and navigate to the URL by passing a value between 100 and 1000 and you should get the response as expected as shown in the below image.



Now, if you pass a value greater than 1000 or less than 100, then you will get the following resource not found 404 error page.



**Range Constraint in ASP.NET Core:**

Instead of using the min and max constraint to specify the minimum and maximum value, we can also use the Range constraint. The Range method takes two parameters, the first parameter is the minimum value and the second parameter is the maximum value. Let us rewrite the previous example using the Range method bypassing 100 and 1000 as the two parameters as shown below.

**[**Route**(**"{EmployeeId:int:range(100,1000)}"**)]**

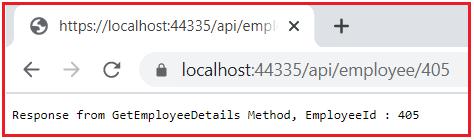
**public** string GetEmployeeDetails**(int** EmployeeId**)**

**{**

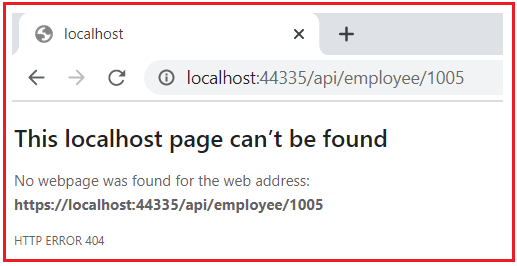
**return** $"Response from GetEmployeeDetails Method, EmployeeId : {EmployeeId}";

**}**

With the above changes in place, run the application and navigate to the URL by passing a value between 100 and 1000 and you should get the response as expected as shown in the below image.



Now, if you pass a value greater than 1000 or less than 100, then you will get the following resource not found 404 error page.



**MinLength Route Constraint in ASP.NET Core Web API:**

The MinLength constraint is used to specify the minimum length constraint on the string parameter. For example, the GetEmployeeDetails(string EmployeeName) method takes one string parameter and we want if the length of the input parameter is greater than 5 characters then only invoke this method. We can achieve this very easily by using the minlength constraint which takes one parameter to specify the minimum length. So, let modify the GetEmployeeDetails(string EmployeeName) action method as shown below.

**[**Route**(**"{EmployeeName:alpha:minlength(5)}"**)]**

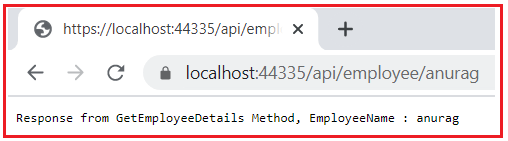
**public** string GetEmployeeDetails**(**string EmployeeName**)**

**{**

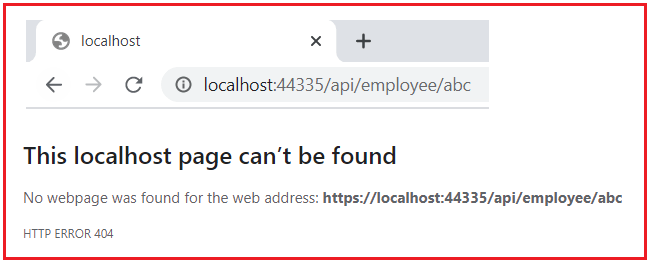
**return** $"Response from GetEmployeeDetails Method, EmployeeName : {EmployeeName}";

**}**

With the above changes in place, run the application and pass a string of more than 5 characters and you should see the following response.



Now, if you pass a string less than 5 characters in URL like **api/employee/abc**, then you will get the resource not found 404 error page as shown in the below image.



**MaxLength Route Constraint in ASP.NET Core Web API:**

Just like the minlength constraint, we can also apply the MinLength constraint to specify the maximum length on the string parameter. For example, the GetEmployeeDetails(string EmployeeName) method takes one string parameter, we want if the length of the input parameter is less than 10 characters then only invoke the method.

We can achieve the above very easily by using the maxlength constraint which takes one parameter to specify the maximum length. So, let modify the GetEmployeeDetails(string EmployeeName) action method as shown below.

**[**Route**(**"{EmployeeName:alpha:maxlength(10)}"**)]**

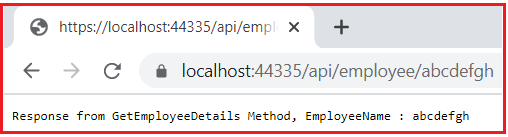
**public** string GetEmployeeDetails**(**string EmployeeName**)**

**{**

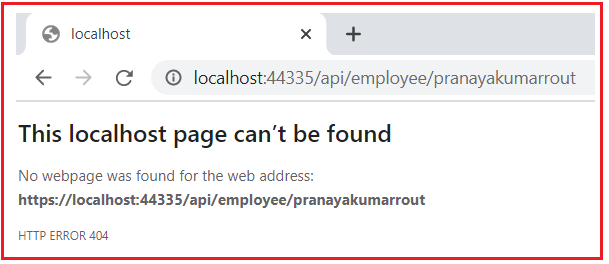
**return** $"Response from GetEmployeeDetails Method, EmployeeName : {EmployeeName}";

**}**

With the above changes in place, now run the application and pass a string of less than 10 characters and it should work and you should get the following response.



Now, if you pass a string greater than 10 characters in URL like **api/employee/pranayakumarrout**, then you will get the following resource not found 404 error page.



It is also possible to apply both minlength and maxlength constraints on a single parameter. For example, if we want to specify the EmployeeName parameter to a minimum of 5 characters and a maximum of 10 characters, then we can apply both minlength and maxlength constraints as shown below.

**[**Route**(**"{EmployeeName:alpha:minlength(5):maxlength(10)}"**)]**

**public** string GetEmployeeDetails**(**string EmployeeName**)**

**{**

**return** $"Response from GetEmployeeDetails Method, EmployeeName : {EmployeeName}";

**}**

Now run the application and test the same by yourself by passing a string between 5 and 10 characters as well as a string less than 5 and greater than 10 characters.

**Length Route Constraint in ASP.NET Core Web API:**

The Length Route Constraint is basically used to specify the exact length of a string. This constraint takes one parameter which specifies the length to be applied for the parameter. For example, we want the GetEmployeeDetails(string EmployeeName) method to be invoked only when the EmployeeName is five characters. Then we can achieve the same very easily by using the length route constraint. So, please modify the GetEmployeeDetails(string EmployeeName) method as shown below.

**[**Route**(**"{EmployeeName:alpha:length(5)}"**)]**

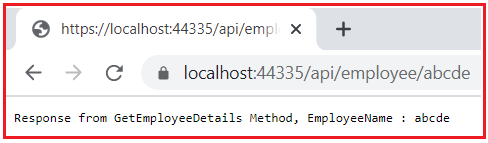
**public** string GetEmployeeDetails**(**string EmployeeName**)**

**{**

**return** $"Response from GetEmployeeDetails Method, EmployeeName : {EmployeeName}";

**}**

Save the above changes and run the application and pass a string with exactly 5 characters in the URL and you should get the following response.



Now, if you pass a string less than 5 like **api/employee/abcd** or a string greater than 5 like **api/employee/abcdef**, then you will get the resource not found 404 error page.

**Regex Route Constraint in ASP.NET Core Web API:**

You can also validate the value of a particular variable by using the regex route constraint. The regex takes one parameter and you can specify an expression or pattern to validate. Let us understand this with an example. Modify the **GetEmployeeDetails(string EmployeeName)** method as shown below. Here, we are adding a simple pattern that will validate if the string starts with the letter a and followed by b or c, then this method is going to be invoked.

**[**Route**(**"{EmployeeName:regex(a(b|c))}"**)]**

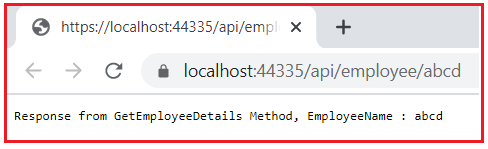
**public** string GetEmployeeDetails**(**string EmployeeName**)**

**{**

**return** $"Response from GetEmployeeDetails Method, EmployeeName : {EmployeeName}";

**}**

Save the above changes and run the application and pass abcd string in the URL and you should get the following response.



Now, if you pass a string that does not start with a or if it starts with a but not followed by b or c character, then you will get the resource not found 404 error page.