**Controller Action Return Types in ASP.NET Core Web API**

**Controller Action Return Types in ASP.NET Core Web API**

In this article, I am going to discuss the different **Controller Action Method Return Types in ASP.NET Core Web API** Application with Examples. Please read our previous article, where we discussed [**Routing in ASP.NET Core Web API**](https://dotnettutorials.net/lesson/routing-in-asp-net-core-web-api/). At the end of this article, you will understand what are the different ways to return data from the ASP.NET Core Controller action method.

**Controller Action Return Types in ASP.NET Core Web API**

In ASP.NET Core Web API, in three different ways, we can return data from the controller action method. They are as follows:

1. **Specific type**
2. **IActionResult**
3. **ActionResult<T>**

Let us discuss each of them in detail. Before understanding this, let us first create a new ASP.NET Core Web API project with the name ReturnTypeAndStatusCodes.

**Adding Employee Model:**

Once you created the Project, then add a folder with the name **Models** to the project root directory. Then add a class file with the name **Employee.cs** with the Models folder. Then open the **Employee.cs** class and copy-paste the following code in it. This is a very simple Employee class that contains Id, Name, Gender, City, Age, and Department properties.

**namespace** *ReturnTypeAndStatusCodes.Models*

**{**

**public** **class** Employee

**{**

**public** **int** Id **{** **get**; **set**; **}**

**public** string Name **{** **get**; **set**; **}**

**public** string Gender **{** **get**; **set**; **}**

**public** string City **{** **get**; **set**; **}**

**public** **int** Age **{** **get**; **set**; **}**

**public** string Department **{** **get**; **set**; **}**

**}**

**}**

**Adding Employee Controller:**

Then add an empty ASP.NET Core Web API Controller with the name EmployeeController within the Controllers folder. Once you created the controller, it should be created with the following code.

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

**using** *System;*

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

**using** *System.Linq;*

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

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

**namespace** *ReturnTypeAndStatusCodes.Controllers*

**{**

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

**[**ApiController**]**

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

**{**

**}**

**}**

**Specific Type as the Return type in ASP.NET Core Web API**

We can return any type of primitive data like string, integer, Boolean, etc., or complex data like Employee, Product, etc, directly from the controller action method.

**Returning String from ASP.NET Core Web API Controller Action Method:**

Please add the following action method within the Employee Controller. The following GetName Method simply returning a string.

**[**Route**(**"Name"**)]**

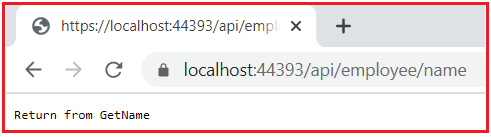
**public** string GetName**()**

**{**

**return** "Return from GetName";

**}**

Now save the changes, run the application and navigate to **api/employee/name** and you should get the following response.



In this case, you can return any primitive data from the ASP.NET Core Web API Controller action method.

**Returning Complex Type in ASP.NET Core Web API:**

Now let us see how to return complex data from the controller action method. As we have created the Employee model, let us return an employee object from the controller action method. So, add the following GetEmployeeDetails action method within the Employee Controller. Also, add the namespace to which the Employee model belongs to.

**[**Route**(**"Details"**)]**

**public** Employee GetEmployeeDetails**()**

**{**

**return** new Employee**()**

**{**

Id = 1001,

Name = "Anurag",

Age = 28,

City = "Mumbai",

Gender = "Male",

Department = "IT"

**}**;

**}**

With the above action method in place, now run the application and navigate to **api/employee/details** and you should get the following response. Here, you are getting the response in JSON format which will return the employee data in key-value pair.

**Returning List<T> from ASP.NET Core Web API Action Method**

Now let us see how to return a complex type of collection data from the controller action method. Let us return the List of employees from the controller action method. So, add the following GetAllEmployee action method within the Employee Controller.

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

**public** List**<**Employee**>** GetAllEmployee**()**

**{**

**return** new List**<**Employee**>()**

**{**

new Employee**(){** Id = 1001, Name = "Anurag", Age = 28, City = "Mumbai", Gender = "Male", Department = "IT" **}**,

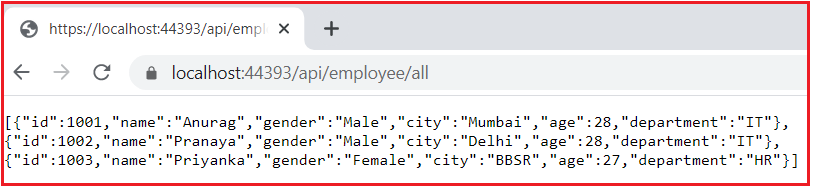
new Employee**(){** Id = 1002, Name = "Pranaya", Age = 28, City = "Delhi", Gender = "Male", Department = "IT" **}**,

new Employee**(){** Id = 1003, Name = "Priyanka", Age = 27, City = "BBSR", Gender = "Female", Department = "HR"**}**,

**}**;

**}**

With the above changes in place, run the application and navigate to **api/employee/all** and you should get the following response. Here, you are getting the list of employees as an array of JSON.



**Returning IEnumerable<T> from ASP.NET Core Web API Controller Actions**

Instead of using List<Employee> as return type, you can also use IEnumerable<Employee>. Please modify the GetAllEmployee action method as shown below.

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

**public** IEnumerable**<**Employee**>** GetAllEmployee**()**

**{**

**return** new List**<**Employee**>()**

**{**

new Employee**(){** Id = 1001, Name = "Anurag", Age = 28, City = "Mumbai", Gender = "Male", Department = "IT" **}**,

new Employee**(){** Id = 1002, Name = "Pranaya", Age = 28, City = "Delhi", Gender = "Male", Department = "IT" **}**,

new Employee**(){** Id = 1003, Name = "Priyanka", Age = 27, City = "BBSR", Gender = "Female", Department = "HR"**}**,

**}**;

**}**

Now run the application and navigate to **api/employee/all**and you should get the same response as the previous example.

**Returning IAsyncEnumerable<T> From ASP.NET Core Web API Action Methods:**

In ASP.NET Core 3.0 and later, you can return IAsyncEnumerable<T> from an action method that provides the following benefits:

1. It no longer results in synchronous iteration.
2. Becomes as efficient as returning IEnumerable<T>.

So, ASP.NET Core 3.0 and later buffers the result of the action method before providing it to the serializer. So, to understand this concept, please add the following GetAllEmployeeAsync action method within the Employee Controller.

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

**public** **async** IAsyncEnumerable**<**Employee**>** GetAllEmployeeAsync**()**

**{**

var listEmployees = new List**<**Employee**>()**

**{**

new Employee**(){** Id = 1001, Name = "Anurag", Age = 28, City = "Mumbai", Gender = "Male", Department = "IT" **}**,

new Employee**(){** Id = 1002, Name = "Pranaya", Age = 28, City = "Delhi", Gender = "Male", Department = "IT" **}**,

new Employee**(){** Id = 1003, Name = "Priyanka", Age = 27, City = "BBSR", Gender = "Female", Department = "HR"**}**,

**}**;

**foreach** **(**var item in listEmployees**)**

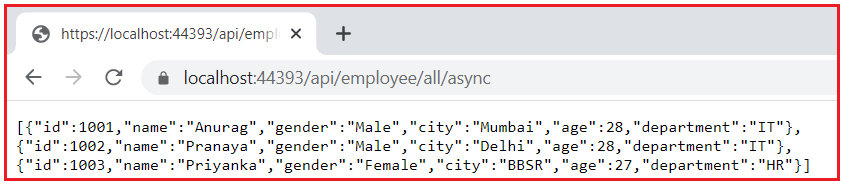
**{**

**yield** **return** item;

**}**

**}**

Now save the changes and run the application and navigate to **api/employee/all/async** and you should get the following JSON array as a response.



So, the Specific return type that we can return from an ASP.NET Core Web API Controller action method are as follows:

1. Any primitive data types from action methods such as int, string, bool, etc.
2. Any complex data object such as Employee, Student, Product, etc.
3. Collection of objects (like List<T> etc)
4. IEnumerable<T>
5. IAsyncEnumerable<T>, etc.

**Benefits of using Specific Return Type in ASP.NET Core Web API:**

While using the swagger or similar type of application, there is no need to define ProducesResponseType because we have defined the return type explicitly.

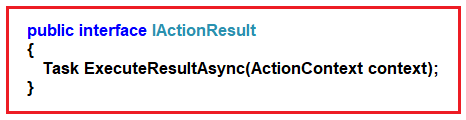
**The drawback of using Specific Return Type in ASP.NET Core Web API:**

You cannot return multiple types of data, let’s say NotFound, OK, Redirect, etc. which are very common in ASP.NET Core Web API to indicate the status of the request.

**IActionResult Return Type in ASP.NET Core Web API:**

The IActionResult return type is appropriate when multiple ActionResult return types are possible in an action. The ActionResult types represent various HTTP status codes.

The IActionResult is an interface and it is used to return multiple types of data. For example, if you want to return NotFound, OK, Redirect, etc. data from your action method then you need to use IActionResult as the return type from your action method. The following is the signature of the IActionResult interface.



The IActionResult interface defines a contract that represents the result of an action method. This interface has the ExecuteResultAsync method. The ExecuteResultAsync method Executes the result operation of the action method asynchronously. This method is called MVC to process the result of an action method and returns a task that represents the asynchronous execute operation. It accepts the context parameter in which the result is going to be executed. The context information includes information about the action that was executed and request information.

**Example to understand IActionResult in ASP.NET Core Web API:**

We want to return a list of employees from our action method. If the number of employees is greater than 0, then we need to return the status OK with the list of employees else we need to return the status Not Found. We can achieve this very easily by using the IActionResult result type. So, modify the Employee Controller as shown below. Here, we have created one action method i.e. GetAllEmployees with the return type as IActionResult.

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

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

**using** *ReturnTypeAndStatusCodes.Models;*

**namespace** *ReturnTypeAndStatusCodes.Controllers*

**{**

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

**[**ApiController**]**

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

**{**

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

**public** IActionResult GetAllEmployee**()**

**{**

var listEmployees = new List**<**Employee**>()**

**{**

new Employee**(){** Id = 1001, Name = "Anurag", Age = 28, City = "Mumbai", Gender = "Male", Department = "IT" **}**,

new Employee**(){** Id = 1002, Name = "Pranaya", Age = 28, City = "Delhi", Gender = "Male", Department = "IT" **}**,

new Employee**(){** Id = 1003, Name = "Priyanka", Age = 27, City = "BBSR", Gender = "Female", Department = "HR"**}**,

**}**;

**if(**listEmployees.Count **>** 0**)**

**{**

**return** Ok**(**listEmployees**)**;

**}**

**else**

**{**

**return** NotFound**()**;

**}**

**}**

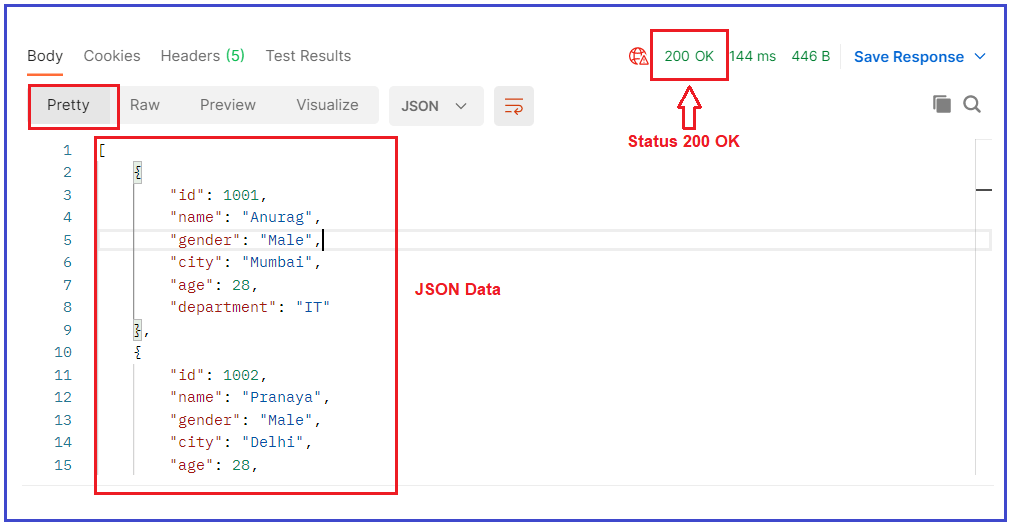
**}**

**}**

First, run the application and find the port number on which your application is running. Now in order to check the returned data with status, we are going to use the Postman client tool. So, open postman and then make a GET to Request as shown in the below image.



As you can see in the above image, first select the HTTP verb as GET and then provide the URL and finally click on the Send button which will make a GET request to the URL you specified. Once you hit the send button, you will get the following response. Notice along with the employee data in JSON format, here, you are also getting status code as 200 OK as shown in the below image.



**Returning Not Found Data using IActionResult in ASP.NET Core:**

Now add the following GetEmployeeDetail Action method within the Employee Controller. It accepts the Employee Id as an input parameter and then returns that employee data. If the employee id is existing then it will return the employee data else it will return Not Found data.

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

**public** IActionResult GetEmployeeDetails**(int** Id**)**

**{**

var listEmployees = new List**<**Employee**>()**

**{**

new Employee**(){** Id = 1001, Name = "Anurag", Age = 28, City = "Mumbai", Gender = "Male", Department = "IT" **}**,

new Employee**(){** Id = 1002, Name = "Pranaya", Age = 28, City = "Delhi", Gender = "Male", Department = "IT" **}**,

new Employee**(){** Id = 1003, Name = "Priyanka", Age = 27, City = "BBSR", Gender = "Female", Department = "HR"**}**,

**}**;

var employee = listEmployees.FirstOrDefault**(**emp =**>** emp.Id == Id**)**;

**if** **(**employee != **null)**

**{**

**return** Ok**(**employee**)**;

**}**

**else**

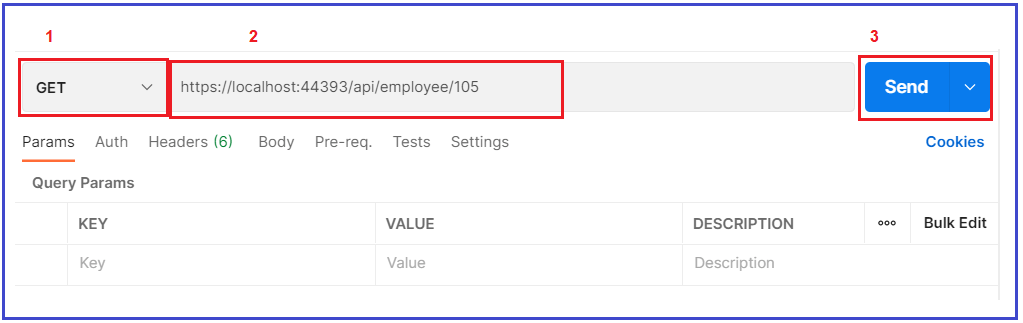
**{**

**return** NotFound**()**;

**}**

**}**

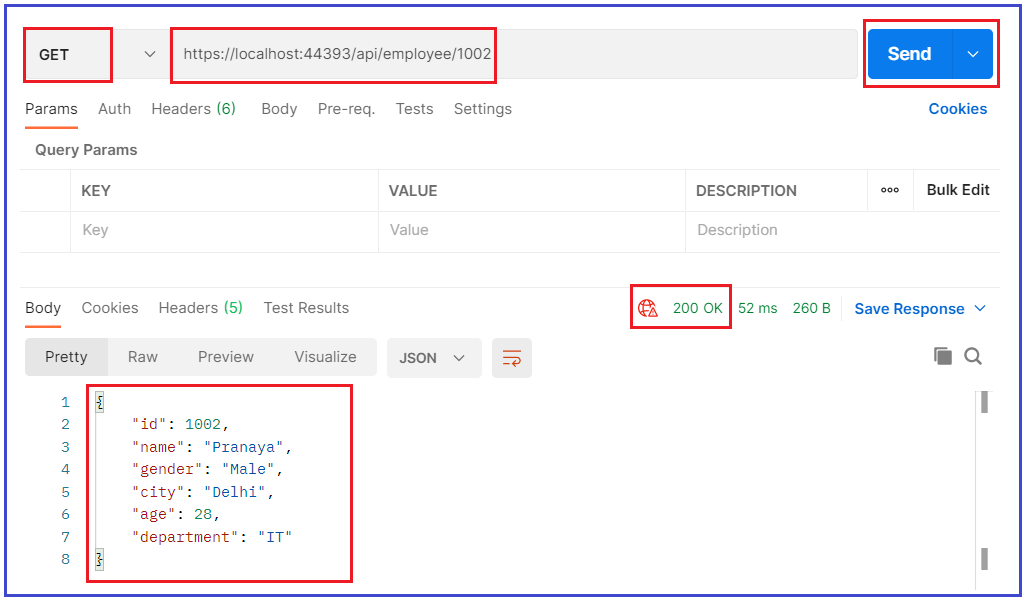
Save the changes and run the application and issue the following request from the postman. As you can see, here, we passing the employee id as 105 which does not exist.



As you can see in the above image, we select the HTTP verb as GET and then provide the URL and finally click on the Send button which will make a GET request to the URL you specified. Once you hit the send button, you will get the following response. Notice, now we are getting 404 Not Found response as the employee id which we are sending does not exist.



Now if you issue a request with an existing employee Id then you will get a 200 OK status code with the employee data in JSON format as shown in the below image.



So, as you can see in the above example, for the same action method GetEmployeeDetails we returned two different types of data. The IActionResult is an Interface and allows us to return multiple types. You can return the data using some built-in methods are as follows.

1. OK()
2. NotFound()
3. Content()
4. File()
5. Redirect, Etc.

**Benefits of using IActionResult type in ASP.NET Core Web API**

It allows us to return multiple types of data along with the status code, this is very important for RESTful APIs

**The drawback of using IActionResult type in ASP.NET Core Web API**

As it returns multiple types of data, the swagger would not be able to identify the output, so we need to use the ProducesResponseType explicitly as shown below.

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

**[**ProducesResponseType**(**StatusCodes.Status200OK, Type = typeof**(**Employee**))]**

**[**ProducesResponseType**(**StatusCodes.Status404NotFound**)]**

**public** IActionResult GetEmployeeDetails**(int** Id**)**

**{**

var listEmployees = new List**<**Employee**>()**

**{**

new Employee**(){** Id = 1001, Name = "Anurag", Age = 28, City = "Mumbai", Gender = "Male", Department = "IT" **}**,

new Employee**(){** Id = 1002, Name = "Pranaya", Age = 28, City = "Delhi", Gender = "Male", Department = "IT" **}**,

new Employee**(){** Id = 1003, Name = "Priyanka", Age = 27, City = "BBSR", Gender = "Female", Department = "HR"**}**,

**}**;

var employee = listEmployees.FirstOrDefault**(**emp =**>** emp.Id == Id**)**;

**if** **(**employee != **null)**

**{**

**return** Ok**(**employee**)**;

**}**

**else**

**{**

**return** NotFound**()**;

**}**

**}**

**ActionResult<T> Return Type in ASP.NET Core Web API:**

It is the combination of ActionResult and Specific type. The ASP.NET Core 2.1 introduced the **ActionResult<T>** return type for the Web API controller action methods. It enables us to return a type deriving either from ActionResult or return a specific type.

Let us understand this with an example. Please modify the Employee Controller class as shown below. The GetEmployeeDetails method takes one parameter and if the parameter value is 0, then it returns NotFound else it returns the employee object.

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

**using** *ReturnTypeAndStatusCodes.Models;*

**namespace** *ReturnTypeAndStatusCodes.Controllers*

**{**

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

**[**ApiController**]**

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

**{**

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

**public** ActionResult**<**Employee**>** GetEmployeeDetails**(int** Id**)**

**{**

**if** **(**Id == 0**)**

**{**

**return** NotFound**()**;

**}**

**else**

**{**

**return** new Employee**()** **{** Id = 1001, Name = "Anurag", Age = 28, City = "Mumbai", Gender = "Male", Department = "IT" **}**;

**}**

**}**

**}**

**}**

**Advantages of ActionResult<T> over ActionResult in ASP.NET Core Web API:**

ActionResult<T> offers the following benefits over the IActionResult type:

1. The [ProducesResponseType] attribute’s Type property can be excluded. For example, [ProducesResponseType(200, Type = typeof(Employee))] is simplified to [ProducesResponseType(200)]. The action’s expected return type is instead inferred from the T in ActionResult<T>.
2. Implicit cast operators support the conversion of both T and ActionResult to ActionResult<T>. T converts to ObjectResult, which means return new ObjectResult(T); is simplified to return T.

**HTTP Status Codes in ASP.NET Core Web API**

**HTTP Status Code in ASP.NET Core Web API**

In this article, I am going to discuss **HTTP Status Code in ASP.NET Core Web API**. Returning the response with a proper status code is the backbone of any restful Web APIs. Now, it is time to learn how can we format the response with the proper response code as per our business requirement.

**HTTP Status Codes:**

The HyperText Transport Protocol status code is one of the important components of HTTP Response. The Status code is issued from the server and they give information about the response. Whenever we get any response from the server, in that Response, we must have one HTTP Status code. All the HTTP Status codes are divided into five categories. They are as follows. Here, XX will represent the actual number.

1. **1XX**: Informational Response (Example: 100, 101, 102, etc.)
2. **2XX**: Successful, whenever you get 2XX as the response code, it means the request is successful. For example, we get 200 HTTP Status Code for the success of a GET request, 201 if a new resource has been successfully created. 204 status code is also for success but in return, it does not return anything just like if the client has performed a delete operation and in return doesn’t really expect something back.
3. **3XX**: 3XX HTTP status codes are basically used for redirection. Whenever you get 3XX as the response code, it means it is re-directional. for example, to tell a client that the requested resource like page, the image has been moved to another location.
4. **4XX**: 4XX HTTP status codes are meant to state errors or Client Error. Whenever you get 4XX as the response code, it means there is some problem with your request. For example, status code 400 means Bad Request, 401 is Unauthorized that is invalid authentication credentials or details have been provided by the client, 403 HTTP Status code means that authentication is a success, but the user is not authorized. 404 HTTP Status code means the requested resource is not available.
5. **5XX**: 5XX HTTP status codes are meant for Server Error. Whenever you get 5XX as the response code, it means there is some problem in the server. Internal Server Error exception is very common, which contains code 500. This error means that there is some unexpected error on the server and the client cannot do anything about it.

**Frequently used HTTP Status Codes in ASP.NET Core Web API:**

The following are some of the frequently used Status codes.

1. **100**: 100 means Continue. The HTTP 100 Continue informational status response code indicates that everything so far is OK and that the client should continue with the request or ignore it if it is already finished.
2. **200**: 200 means OK. The HTTP 200 OK success status response code indicates that the request has succeeded. If you are searching for some data and you got the data properly. That means the request is successful and, in that case, you will get 200 OK as the HTTP status code.
3. **201**: 201 means a new resource created. The HTTP 201 Created success status response code indicates that the request has succeeded and has led to the creation of a resource. The new resource is effectively created before this response is sent back and the new resource is returned in the body of the message, its location being either the URL of the request or the content of the Location header. If you are adding successfully a new resource by using the HTTP Post method, then in that case you will get 201 as the Status code.
4. **204**: 204 means No Content. The HTTP 204 No Content success status response code indicates that a request has succeeded, but that the client doesn’t need to navigate away from its current page. If the server processed the request successfully and it is not returning any content, then in that case you will get a 204-response status code.
5. **301**: 301 means Moved Permanently. If you are getting 301 as a status code from the server, it means the resource you are looking for is moved permanently to the URL given by the Location headers.
6. **302**: 302 means Found. If you are getting 302 as a status code from the server, it means the resource you are looking for is moved temporarily to the URL given by the Location headers.
7. **400**: 400 means Bad Request. If you are getting 400 as the status code from the server, then the issue is with the client request. If the request contains some wrong data such as malformed request syntax, invalid request message framing, or deceptive request routing, then we will get this 400 Bad Request status code.
8. **401**: 401 means Unauthorized. If you are trying to access the resource for which you don’t have access (Invalid authentication credentials), then you will get a 401 unauthorized status code from the server.
9. **404**: 404 means Not Found. If you are looking for a resource that does not exist, then you will get this 404 Not Found status code from the server. Links that lead to a 404 page are often called broken or dead links.
10. **405**: 405 means Method Not Allowed. The 405 Method Not Allowed response status code indicates that the request method is known by the server but is not supported by the target resource. For example, we have one method which is a POST method in the server and we trying to access that method from the client using GET Verb, then, in that case, you will get a 405-status code.
11. **500**: 500 means Internal Server Error. If there is some error in the server, then you will get a 500 Internal Server Error status code.
12. **503**: 503 means Service Unavailable. The 503 Service Unavailable server error response code indicates that the server is not ready to handle the request. If the server is down for maintenance or the server is overloaded then in that case, you will get the 503 Service Unavailable Status code.
13. **504**: 504 means Gateway Timeout. The 504 Gateway Timeout server error response code indicates that the server while acting as a gateway or proxy, did not get a response in time from the upstream server that is needed in order to complete the request.

**Why HTTP Status Codes are Important?**

If we want to consume any Restful API, then we will send an HTTP Request and in return, we will get the response and the response include data as well as an HTTP Status code. The HTTP Status codes are important because they tell the client (client means who initiate the request, for example, Web, Android, iOS, Postman, IoT, Fiddler, etc) about what exactly happened to the request. If you send a wrong HTTP Status code, then that will confuse the client i.e. the consumer of the API.

The client should know that its request has been taken care of or not, and if the response is not as expected, then the Status Code should tell the client where the problem is? Whether the problem is at the Client level or at the API level.

Suppose there is a situation where the client gets the response with the HTTP status code as 200, but at the API level, there is some problem or issue. In that case, as the client gets 200 HTTP Status code, so the client will get a false assumption of everything being fine, whereas that won’t be the case.

So, if there is something wrong at the API level or there are some errors that occurred on the server, the HTTP status code 500 should be sent to the client so that the client knows there is something wrong with the request being sent. This is the reason why sending proper HTTP Response code from Restful APIs is important.

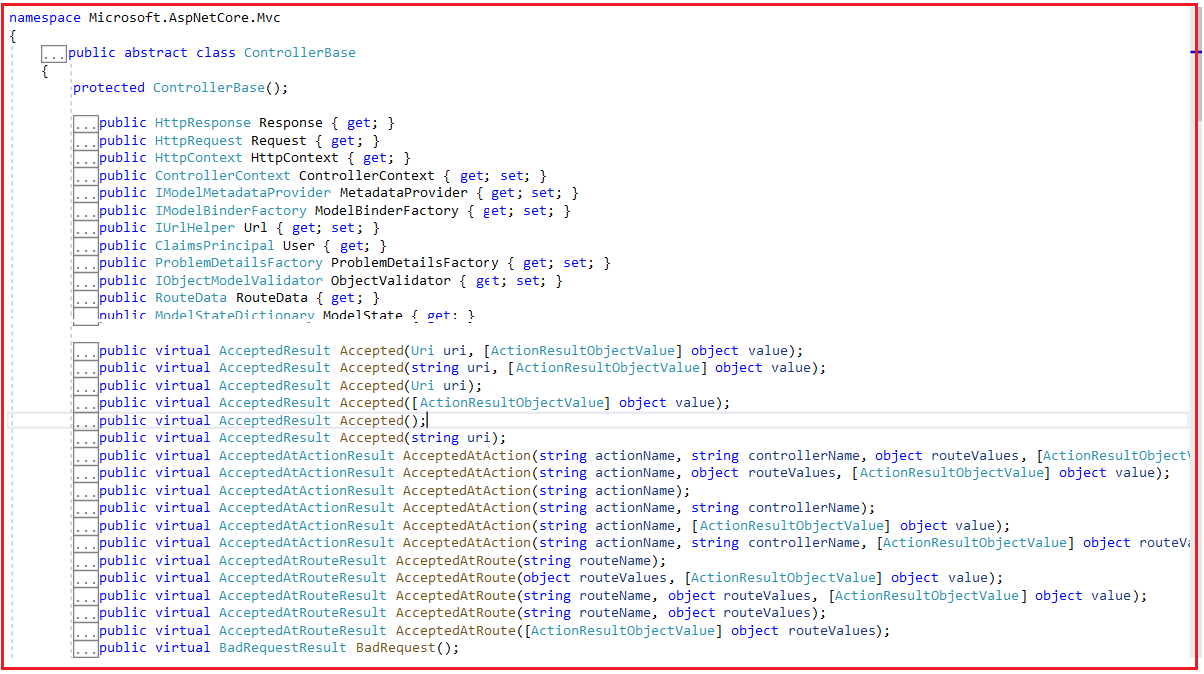
**200 HTTP Status Code in ASP.NET Core Web API**

**200 HTTP Status Code in ASP.NET Core Web API**

In this article, I am going to discuss **how to return 200 HTTP Status Code from the ASP.NET Core Web API** Controller Action method with Examples. Please read our previous article where we give an overview of the [**HTTP status code**](https://dotnettutorials.net/lesson/http-status-codes-in-asp-net-core-web-api/). We are going to work with the same example, that we created in our Controller Action Return Types in the ASP.NET Core Web API article.

**How to return Proper HTTP Status code?**

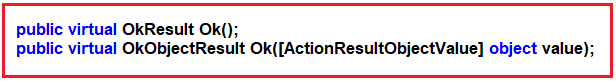
In order to return the proper status code, the ASP.NET Core Framework provided some built-in methods, and using those methods you can format your response data. And the best thing is that you don’t need to remember all these methods. What you need to do is, simply right-click on the ControllerBase class and choose to go to definition. And here, you can see this is an abstract class with lots of properties and methods as shown in the below image.



**How to return 200 HTTP Status Code?**

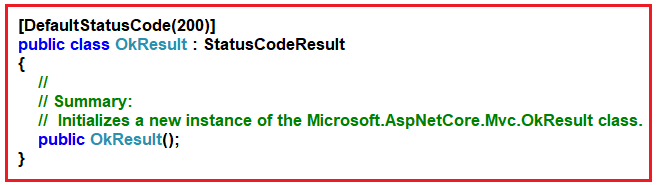
This is the most common HTTP Status Code that is going to be returned from the ASP.NET Core Web API controller method. As we already discussed in our previous article, the 200 HTTP Status Code belongs to the Successful category. That means the HTTP 200 Status code indicates that the request is successful.

ASP.NET Core provided the OK method to return HTTP 200 Status Code. If you check the ControllerBase class then you will find two overloaded versions of the OK method as shown below.

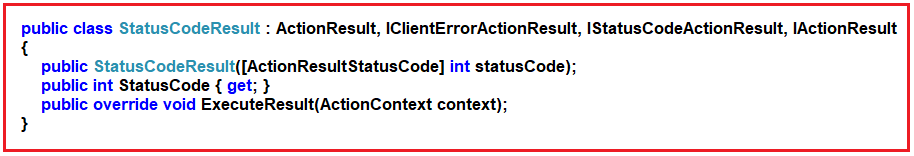


The first overloaded version which does not take any input parameter will create an object that produces an empty HTTP 200 OK Status code as a response. On the other hand, the second overloaded version takes an object (any type of value) as input and creates an object that produces HTTP 200 OK Status code as a response. The input data it takes will be formatted in the response body.

Further, if you notice the first overloaded version of the OK method returns OKResult. Now, right-click on the OkResult and choose to go to definition and you will see the following definition.

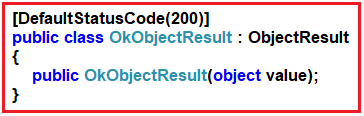


As you can see in the above image, this method is inherited from the StatusCodeResult class, and further if you right-click on the StatusCodeResult class and choose to go to definition, then you will find the following definition.

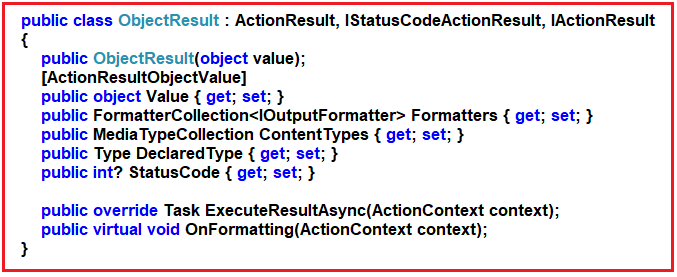


The StatusCodeResult class is inherited from ActionResult and IActionResult. So, you can use IActionResult as the return type for the OK method. The StatusCodeResult class has the StatusCode property and using this property you can set the proper status code.

Further, if you notice the second overloaded version of the OK method returns OkObjectResult. Now, right-click on the OkObjectResult class and choose to go to definition and you will see the following definition.



Here, as you can see, the OkObjectResult class is inherited from the ObjectResult class. Now, again, right-click in the ObjectResult class and choose to go to definition and you will find the following definition of ObjectResult class.



If you notice here, the ObjectResult class is inherited from ActionResult and IActionResult. So, you can use IActionResult as the return type for the second overloaded version of the OK method. Further, if you notice the ObjectResult class has the StatusCode property and using this property you can set the proper status code. Let us use the above two overloaded versions in our example.

**Returning 200 HTTP Status code without data:**

In this case, we have to use the first overloaded version of the OK method which does not take any parameter. So, please modify the Employee Controller as shown below.

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

**namespace** *ReturnTypeAndStatusCodes.Controllers*

**{**

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

**[**ApiController**]**

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

**{**

**[**Route**(**"GetEmployees"**)]**

**public** IActionResult GetEmployees**()**

**{**

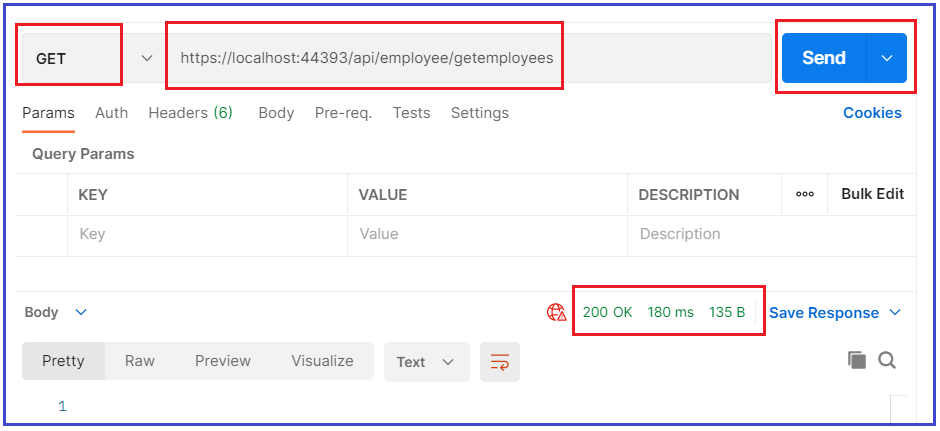
**return** Ok**()**;

**}**

**}**

**}**

Now, run the application and issue a GET request to the URL **api/employee/getemployees** using postman as shown in the below image. As you can see, once you hit the Send button, you will 200 OK Status code without any data in the response body.



**Returning 200 HTTP Status code with data:**

In this case, we have to use the second overloaded version of the OK method which takes an object as an input parameter. So, please modify the Employee Controller class as shown below. Here, we are returning the list of employees from the action method. As the OK method takes object type, so, we can pass any data. Here, I am passing the list of employees to the OK method.

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

**using** *ReturnTypeAndStatusCodes.Models;*

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

**namespace** *ReturnTypeAndStatusCodes.Controllers*

**{**

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

**[**ApiController**]**

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

**{**

**[**Route**(**"GetEmployees"**)]**

**public** IActionResult GetEmployees**()**

**{**

var listEmployees = new List**<**Employee**>()**

**{**

new Employee**(){** Id = 1001, Name = "Anurag", Age = 28, City = "Mumbai", Gender = "Male", Department = "IT" **}**,

new Employee**(){** Id = 1002, Name = "Pranaya", Age = 28, City = "Delhi", Gender = "Male", Department = "IT" **}**,

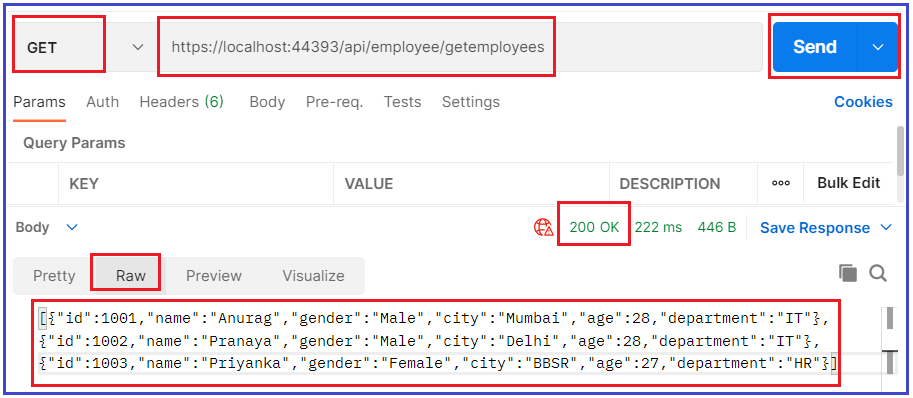
new Employee**(){** Id = 1003, Name = "Priyanka", Age = 27, City = "BBSR", Gender = "Female", Department = "HR"**}**,

**}**;

**return** Ok**(**listEmployees**)**;

**} } }**

Now save the changes and run the application and issue a GET request to the same URL **api/employee/getemployees** using postman as shown in the below image. Now, you can see, once you hit the Send button, you will get 200 OK Status code with the employee data in the response body in JSON format.



The OK Method is very helpful for GET type requests. For example, Get all employees data, Get single employee data, Get employee data based on some search parameters, etc.