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

In this article, I will discuss the different **Controller Action Method Return Types in ASP.NET Core Web API** Applications with Examples. Please read our previous article discussing **Routing in ASP.NET Core Web API**. At the end of this article, you will understand 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, controller actions can return various types of results, ultimately affecting the HTTP response sent to the client. The choice of return type depends on what you need to communicate to the client about the result of the operation performed by the action method. In ASP.NET Core Web API, we can return data from the controller action method in three different ways. They are as follows:

* **Primitive or Complex Types:** Actions can return primitive types (like int, string, etc.) or complex types (such as custom objects). When returning these directly, ASP.NET Core serializes the object into JSON (by default) and sets the content type of the response to application/json.
* **IActionResult:** The IActionResult return type can return various HTTP responses, including status codes, files, and error messages. This return type is useful when an action needs to return different types of responses, such as NotFound(), OK(), BadRequest(), etc.
* **ActionResult<T>:** The ActionResult<T> return type allows for returning either a response that can be serialized into JSON (or another format) or an action result (like IActionResult). This provides more flexibility in return types, allowing for both the return of data (as type T) and action results like NotFound() or Ok().
* **Specific Result Types:** ASP.NET Core provides several specific result types that implement the IActionResult interface and are used to return specific types of responses. Examples include Ok, BadRequest, NotFound, and File. These result types offer a clearer intention of the HTTP response returned from the API.
* **Void:** Returning void from an action method results in an empty 204 No Content response. This is typically used when an action performs an operation that does not need to return data to the client.
* **Task<IActionResult> or Task<ActionResult<T>>:** For asynchronous operations, actions can return Task<IActionResult> or Task<ActionResult<T>>. This enables the action methods to perform asynchronous operations, like database calls or web service requests, without blocking the calling thread. The framework handles the task completion and writes the result to the response.

**Examples to Understand Action Return Types in ASP.NET Core Web API**

Let us create a new ASP.NET Core Web API project called ReturnTypeAndStatusCodes.

**Adding Employee Model:**

Once you created the Project, add a Models folder to the root directory. Then, add a class file named **Employee.cs** with the Models folder. Then open the **Employee.cs** class and copy-paste the following code into it. This class 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 named EmployeeController within the Controllers folder. Once you create the controller, it should be created with the following code.

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

**namespace** *ReturnTypeAndStatusCodes.Controllers*

**{**

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

**[**ApiController**]**

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

**{**

**}**

**}**

**Primitive or Complex Types**

You can return primitive or complex types directly from your actions. When you do this, ASP.NET Core automatically wraps your response in a 200 OK response, assuming the operation was successful. We can return any primitive data, like strings, integers, Booleans, etc., or complex data, like employees, products, 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 returns a string. You can return any primitive data from the ASP.NET Core Web API Controller action method.

**[**HttpGet**(**"Name"**)]**

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

**{**

**return** "Return from GetName";

**}**

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

**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.

**[**HttpGet**(**"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** URL, and you should get the following response. Here, you are getting the response in JSON format, which will return the employee data in a key-value pair.

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

Now, let us see how to return a complex data collection 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.

**[**HttpGet**(**"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 URL,** and you should get the following response. Here, you get the list of employees as an array of JSON.

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

**[**HttpGet**(**"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**URL; you should get the same response as the previous example.

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

The IActionResult return type is used when the action method will return different types of data. The IActionResult return type is capable of representing various HTTP status codes. It’s the most flexible return type because it can return different types of responses, such as OK, BadRequest, NotFound, and many more from your action method.

This is useful when your action method needs to return different HTTP responses based on the outcome of the operation. 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.

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

We want to return a list of employees from our action method. If at least one Employee is present, we need to return the status OK with the list of employees. If no Employee is found, then we need to return the status code Not Found. We can achieve this very easily by using the IActionResult result type. So, modify the Employee Controller as shown below. Here, we created one action method, i.e., GetAllEmployees, with the return type IActionResult.

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

**using** *ReturnTypeAndStatusCodes.Models;*

**namespace** *ReturnTypeAndStatusCodes.Controllers*

**{**

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

**[**ApiController**]**

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

**{**

**[**HttpGet**(**"All"**)]**

//As we are going to return Ok and NotFound Result from this action method

//So, we are using IActionResult as the return type of this method

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

**{**

//In Real-Time, you will get the data from the database

//Here, we have hardcoded the data

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 at least of Emplpyee is Present return OK status code and the list of employees

**if** **(**listEmployees.Any**())**

**{**

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

**}**

**else**

**{**

//If no Employee is Present return Not Found Status Code

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

**}**

**}**

**}**

**}**

First, run the application and find the port number on which your application is running. We will use the Postman client tool to check the returned data with the HTTP Status Code. So, open Postman and then make a GET HTTP Request to **/api/employee/all** endpoints, as shown in the image below.

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

**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 exists, then it will return the employee data. Otherwise, it will return Not Found data.

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

//As the following method is going to return Ok and NotFound Result

//So, we are using IActionResult as the return type of this method

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

**{**

//In Real-Time, you will get the data from the database

//Here, we have hardcoded the data

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"**}**,

**}**;

//Fetch the Employee Data based on the Employee Id

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

//If Employee Exists Return OK with the Employee Data

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

**{**

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

**}**

**else**

**{**

//If Employee Does Not Exists Return NotFound

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

**}**

**}**

Save the changes, run the application, and issue the following HTTP request **(/api/employee/105**) from the postman. As you can see, we are passing the employee ID as 105, which does not exist.

Here, we select the HTTP verb as **GET**, provide the URL **(/api/employee/105**), and finally click the **Send** button to make a GET HTTP request to the provided URL. Once you hit the Send button, you will get the following response. Notice, now we are getting a 404 Not Found response as the employee ID that we are sending does not exist.

If you issue a request with an existing employee ID, you will get a 200 OK status code with the employee data in JSON format, as shown in the image below.

As you can see in the above example, we returned two different types of data from the same action method, GetEmployeeDetails. The IActionResult is an Interface and allows us to return multiple types. You can return the data using some built-in methods as follows, which we will discuss in our upcoming articles.

* OK()
* NotFound()
* Content()
* File()
* Redirect, Etc.

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

The IActionResult type in ASP.NET Core Web API provides several benefits that enhance web application development, maintenance, and scalability. It is a flexible and unified way to return different types of responses from controller actions. Here are some of the key benefits:

* **Flexibility in Response Types:** IActionResult allows an action method to return various response types, such as OK, BadRequest, NotFound, or File. This flexibility lets developers easily handle responses to the needs of different API calls without having to change the method signature.
* **Improved Readability and Maintainability:** By using IActionResult, the intent of the code becomes clearer. For instance, returning Ok(result) or NotFound() directly from an action method makes it easy to understand the possible outcomes of the API call, improving the maintainability of the code.
* **Support for Asynchronous Operations:** ASP.NET Core supports asynchronous action methods, allowing for non-blocking I/O operations. IActionResult can be used with asynchronous patterns, such as Task<IActionResult>, to improve the scalability and performance of web applications by efficiently handling web requests.
* **Enhanced Client-Side Interaction:** By using standardized responses provided by IActionResult, such as status codes and content results, client-side applications can more easily interpret and react to the API responses. This standardization helps in building robust client-server communication.

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

Using IActionResult also comes with certain drawbacks, which are important to consider in the design and implementation of your API:

* **Lack of Strong Typing:** When a controller action returns IActionResult, the specific type of response is not explicitly defined in the method signature. This lack of strong typing can make it harder for developers to understand the types of responses to expect. It can complicate the process of generating client code using tools like Swagger/OpenAPI.
* **Reduced Clarity and Predictability:** Since IActionResult can represent a wide range of HTTP responses, it might not be immediately clear to consumers of your API what the possible responses are without consulting documentation or inspecting the implementation. This can reduce the clarity and predictability of the API, making it more difficult to use.
* **Performance Overhead:** While often minimal, a slight performance overhead is associated with the abstraction layer IActionResult introduces. Each action result needs to be executed to generate the actual HTTP response. In high-performance scenarios, this overhead, however small, could become significant.

As it returns multiple data types, the swagger could not identify the output shown in the image below.

As you can see in the above image, the kind of data the API will return is not displayed. To overcome this problem, we need to use the ProducesResponseType explicitly, as shown below.

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

**using** *ReturnTypeAndStatusCodes.Models;*

**namespace** *ReturnTypeAndStatusCodes.Controllers*

**{**

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

**[**ApiController**]**

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

**{**

**[**HttpGet**(**"All"**)]**

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

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

**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.Any**())**

**{**

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

**}**

**else**

**{**

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

**}**

**}**

**[**HttpGet**(**"{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**()**;

**}**

**}**

**}**

**}**

Now, if you view the swagger, it must display the different responses based on the ProducesResponseType attribute, as shown in the image below.

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

ActionResult<T> extends IActionResult by allowing you to return either a response type that implements IActionResult or return a specific type T. So, it is the combination of ActionResult and Specific type. It enables us to return a type deriving either from ActionResult or 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, it returns NotFound (an ActionResult type); otherwise, it returns the employee object (a complex type).

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

**using** *ReturnTypeAndStatusCodes.Models;*

**namespace** *ReturnTypeAndStatusCodes.Controllers*

**{**

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

**[**ApiController**]**

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

**{**

**[**HttpGet**(**"{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" **}**;

**}**

**}**

**}**

**}**

**Specific Result Types in ASP.NET Core Web API**

You can directly return specific result types such as OkObjectResult, BadRequestResult, NotFoundResult, etc. This approach is more explicit but less flexible than returning IActionResult or ActionResult<T>.

For a better understanding, please have a look at the following example. Here, we marked the method return type as OkObjectResult, which means we can only return the OK status code from this action method.

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

**using** *ReturnTypeAndStatusCodes.Models;*

**namespace** *ReturnTypeAndStatusCodes.Controllers*

**{**

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

**[**ApiController**]**

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

**{**

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

**public** OkObjectResult GetEmployeeDetails**(int** Id**)**

**{**

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

**return** Ok**(**item**)**; // Directly returns an OkObjectResult

**}**

**}**

**}**

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

Returning void from an action results in a 204 No Content response. This is suitable for actions that perform operations but don’t need to return any data.

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

**public** **void** DeleteEmployee**(int** id**)**

**{**

// Returns a 204 No Content response

**}**

**Task<IActionResult> or Task<ActionResult<T>> Return Type in ASP.NET Core:**

For asynchronous operations, you can return Task<IActionResult> or Task<ActionResult<T>>. This allows your action methods to perform asynchronous operations and return various responses.

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

For a better understanding, please have a look at the following example, where we have used the Task<IActionResult> return type.

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

**using** *ReturnTypeAndStatusCodes.Models;*

**namespace** *ReturnTypeAndStatusCodes.Controllers*

**{**

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

**[**ApiController**]**

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

**{**

**[**HttpGet**(**"All"**)]**

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

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

**public** **async** Task**<**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.Any**())**

**{**

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

**}**

**else**

**{**

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

**}**

**}**

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

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

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

**public** **async** Task**<**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**()**;

**}**

**}**

**}**

**}**

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

For a better understanding, please have a look at the following example, where we have used the Task<ActionResult<T>> return type.

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

**using** *ReturnTypeAndStatusCodes.Models;*

**namespace** *ReturnTypeAndStatusCodes.Controllers*

**{**

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

**[**ApiController**]**

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

**{**

**[**HttpGet**(**"All"**)]**

**public** **async** Task**<**ActionResult**<**List**<**Employee**>>>** GetAllEmployee**()**

**{**

//Use async operation to get the data from the database

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.Any**())**

**{**

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

**}**

**else**

**{**

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

**}**

**}**

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

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

**{**

//Use async operation to get the data from the database

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**()**;

**}**

**}**

**}**

**}**

**When should you use which return type in ASP.NET Core?**

Choosing the appropriate return type depends on the specific requirements of the action method being implemented:

* Use IActionResult or ActionResult<T> for actions that might return different types of responses.
* Return specific result types when the response is known and fixed.
* Use primitive or complex types for simple data-returning actions.
* Return void or Task for actions that do not need to return data.
* Use Task<IActionResult> or Task<ActionResult<T>> for asynchronous operations needing flexibility in response types.

In the next and a few upcoming articles, we will discuss the most useful [**Status Code Methods in ASP.NET Core Web API**](https://dotnettutorials.net/lesson/http-status-codes-in-asp-net-core-web-api/) Applications. In this article, I try to explain **Controller Action Method Return Types in ASP.NET Core Web API** Application with examples. I hope you enjoy how to return different return types in the ASP.NET Core Web API Controller action methods article.