

# Controller action return types in ASP.NET Core web API

02/03/2020 • 6 minutes to read • S R G S M +2

## In this article

[Specific type](#)

[IActionResult type](#)

[ActionResult<T> type](#)

[Additional resources](#)

By [Scott Addie](#)

[View or download sample code](#) ([how to download](#))


ASP.NET Core offers the following options for web API controller action return types:

- [Specific type](#)
- [IActionResult](#)
- [ActionResult<T>](#)

This document explains when it's most appropriate to use each return type.

## Specific type

The simplest action returns a primitive or complex data type (for example, `string` or a custom object type). Consider the following action, which returns a collection of custom `Product` objects:


C#	 Copy
<pre>[HttpGet] public List&lt;Product&gt; Get() =&gt;     _repository.GetProducts();</pre>	

Without known conditions to safeguard against during action execution, returning a specific type could suffice. The preceding action accepts no parameters, so parameter constraints validation isn't needed.


When multiple return types are possible, it's common to mix an [ActionResult](#) return type with the primitive or complex return type. Either [ActionResult](#) or [ActionResult<T>](#) are necessary to accommodate this type of action. Several samples of multiple return types are provided in this document.

## Return IEnumerable<T> or IQueryable<T>

In ASP.NET Core 2.2 and earlier, returning [IEnumerable<T>](#) from an action results in synchronous collection iteration by the serializer. The result is the blocking of calls and a potential for thread pool starvation. To illustrate, imagine that Entity Framework (EF) Core is being used for the web API's data access needs. The following action's return type is synchronously enumerated during serialization:

C#	 Copy
<pre>public IEnumerable&lt;Product&gt; GetOnSaleProducts() =&gt;     _context.Products.Where(p =&gt; p.IsOnSale);</pre>	


To avoid synchronous enumeration and blocking waits on the database in ASP.NET Core 2.2 and earlier, invoke `ToListAsync`:

C#	 Copy
<pre>public async Task&lt;IEnumerable&lt;Product&gt;&gt; GetOnSaleProducts() =&gt;     await _context.Products.Where(p =&gt; p.IsOnSale).ToListAsync();</pre>	

In ASP.NET Core 3.0 and later, returning [IAsyncEnumerable<T>](#) from an action:

- No longer results in synchronous iteration.
- Becomes as efficient as returning [IEnumerable<T>](#).

ASP.NET Core 3.0 and later buffers the result of the following action before providing it to the serializer:


C#	 Copy
<pre>public IEnumerable&lt;Product&gt; GetOnSaleProducts() =&gt;     _context.Products.Where(p =&gt; p.IsOnSale);</pre>	

Consider declaring the action signature's return type as [IAsyncEnumerable<T>](#) to guarantee the asynchronous iteration. Ultimately, the iteration mode is based on the


underlying concrete type being returned. MVC automatically buffers any concrete type that implements `IAsyncEnumerable<T>`.

Consider the following action, which returns sale-priced product records as

`IEnumerable<Product>`:

C#	 Copy
<pre>[HttpGet("syncsale")] public IEnumerable&lt;Product&gt; GetOnSaleProducts() {     var products = _repository.GetProducts();      foreach (var product in products)     {         if (product.IsOnSale)         {             yield return product;         }     } }</pre>	

The `IAsyncEnumerable<Product>` equivalent of the preceding action is:

C#	 Copy
<pre>[HttpGet("asyncsale")] public async IAsyncEnumerable&lt;Product&gt; GetOnSaleProductsAsync() {     var products = _repository.GetProductsAsync();      await foreach (var product in products)     {         if (product.IsOnSale)         {             yield return product;         }     } }</pre>	

Both of the preceding actions are non-blocking as of ASP.NET Core 3.0.

## IActionResult type


The [ActionResult](#) return type is appropriate when multiple `ActionResult` return types are possible in an action. The `ActionResult` types represent various HTTP status codes. Any non-abstract class deriving from `ActionResult` qualifies as a valid return type. Some common return types in this category are [BadRequestResult](#) (400), [NotFoundResult](#) (404), and [OkObjectResult](#) (200). Alternatively, convenience methods in the [ControllerBase](#) class can be used to return `ActionResult` types from an action. For example, `return BadRequest();` is a shorthand form of `return new BadRequestResult();`.

Because there are multiple return types and paths in this type of action, liberal use of the [\[ProducesResponseType\]](#) attribute is necessary. This attribute produces more descriptive response details for web API help pages generated by tools like [Swagger](#).

`[ProducesResponseType]` indicates the known types and HTTP status codes to be returned by the action.

## Synchronous action

Consider the following synchronous action in which there are two possible return types:

C#	 Copy
<pre>[HttpGet("{id}")] [ProducesResponseType(StatusCodes.Status200OK)] [ProducesResponseType(StatusCodes.Status404NotFound)] public IActionResult GetById(int id) {     if (!_repository.TryGetProduct(id, out var product))     {         return NotFound();     }      return Ok(product); }</pre>	

In the preceding action:

- A 404 status code is returned when the product represented by `id` doesn't exist in the underlying data store. The [NotFound](#) convenience method is invoked as shorthand for `return new NotFoundResult();`
- A 200 status code is returned with the `Product` object when the product does exist. The [Ok](#) convenience method is invoked as shorthand for `return new OkObjectResult(product);`

# Asynchronous action


Consider the following asynchronous action in which there are two possible return types:

C#	 Copy
<pre>[HttpPost] [Consumes(MediaTypeNames.Application.Json)] [ProducesResponseType(StatusCodes.Status201Created)] [ProducesResponseType(StatusCodes.Status400BadRequest)] public async Task&lt;IActionResult&gt; CreateAsync(Product product) {     if (product.Description.Contains("XYZ Widget"))     {         return BadRequest();     }      await _repository.AddProductAsync(product);      return CreatedAtAction(nameof(GetById), new { id = product.Id }, product); }</pre>	

In the preceding action:

- A 400 status code is returned when the product description contains "XYZ Widget". The `BadRequest` convenience method is invoked as shorthand for `return new BadRequestResult();`.
- A 201 status code is generated by the `CreatedAtAction` convenience method when a product is created. An alternative to calling `CreatedAtAction` is `return new CreatedAtActionResult(nameof(GetById), "Products", new { id = product.Id }, product);`. In this code path, the `Product` object is provided in the response body. A `Location` response header containing the newly created product's URL is provided.

For example, the following model indicates that requests must include the `Name` and `Description` properties. Failure to provide `Name` and `Description` in the request causes model validation to fail.

C#	 Copy
<pre>public class Product {     public int Id { get; set; }</pre>	

```

    [Required]
    public string Name { get; set; }

    [Required]
    public string Description { get; set; }
}

```

If the [\[ApiController\]](#) attribute in ASP.NET Core 2.1 or later is applied, model validation errors result in a 400 status code. For more information, see [Automatic HTTP 400 responses](#).

## ActionResult<T> type

ASP.NET Core 2.1 introduced the [ActionResult<T>](#) return type for web API controller actions. It enables you to return a type deriving from [ActionResult](#) or return a [specific type](#).

[ActionResult<T>](#) offers the following benefits over the [IActionResult](#) type:

- The [\[ProducesResponseType\]](#) attribute's `Type` property can be excluded. For example, `[ProducesResponseType(200, Type = typeof(Product))]` is simplified to `[ProducesResponseType(200)]`. The action's expected return type is instead inferred from the `T` in `ActionResult<T>`.
- [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;`

C# doesn't support implicit cast operators on interfaces. Consequently, conversion of the interface to a concrete type is necessary to use `ActionResult<T>`. For example, use of `IEnumerable` in the following example doesn't work:

C#

 Copy

```

[HttpGet]
public ActionResult<IEnumerable<Product>> Get() =>
    _repository.GetProducts();

```

One option to fix the preceding code is to return

```
_repository.GetProducts().ToList();
```

Most actions have a specific return type. Unexpected conditions can occur during action execution, in which case the specific type isn't returned. For example, an action's input

parameter may fail model validation. In such a case, it's common to return the appropriate `ActionResult` type instead of the specific type.

## Synchronous action

Consider a synchronous action in which there are two possible return types:

C#	 Copy
<pre>[HttpGet("{id}")] [ProducesResponseType(StatusCodes.Status200OK)] [ProducesResponseType(StatusCodes.Status404NotFound)] public ActionResult&lt;Product&gt; GetById(int id) {     if (!_repository.TryGetProduct(id, out var product))     {         return NotFound();     }      return product; }</pre>	

In the preceding action:

- A 404 status code is returned when the product doesn't exist in the database.
- A 200 status code is returned with the corresponding `Product` object when the product does exist. Before ASP.NET Core 2.1, the `return product;` line had to be `return Ok(product);`.

## Asynchronous action

Consider an asynchronous action in which there are two possible return types:

C#	 Copy
<pre>[HttpPost] [Consumes(MediaTypeNames.Application.Json)] [ProducesResponseType(StatusCodes.Status201Created)] [ProducesResponseType(StatusCodes.Status400BadRequest)] public async Task&lt;ActionResult&lt;Product&gt;&gt; CreateAsync(Product product) {     if (product.Description.Contains("XYZ Widget"))     {         return BadRequest();     } }</pre>	

```
}

    await _repository.AddProductAsync(product);

    return CreatedAtAction(nameof(GetById), new { id = product.Id },
product);
}
```

In the preceding action:

- A 400 status code ([BadRequest](#)) is returned by the ASP.NET Core runtime when:
  - The [\[ApiController\]](#) attribute has been applied and model validation fails.
  - The product description contains "XYZ Widget".
- A 201 status code is generated by the [CreatedAtAction](#) method when a product is created. In this code path, the `Product` object is provided in the response body. A `Location` response header containing the newly created product's URL is provided.

## Additional resources

- [Handle requests with controllers in ASP.NET Core MVC](#)
- [Model validation in ASP.NET Core MVC](#)
- [ASP.NET Core web API documentation with Swagger / OpenAPI](#)

---

Is this page helpful?

 Yes  No

---