

We now integrate with Microsoft Teams, helping you to connect your internal knowledge base with your chat. [Learn more.](#)

ASP.NET Core return JSON with status code

Asked 2 years, 8 months ago Active 1 month ago Viewed 198k times



I'm looking for the correct way to return JSON with a HTTP status code in my .NET Core Web API controller. I use to use it like this:

114

```
public IActionResult GetResourceData()
{
    return this.Content(HttpStatusCode.OK, new { response = "Hello"});
}
```

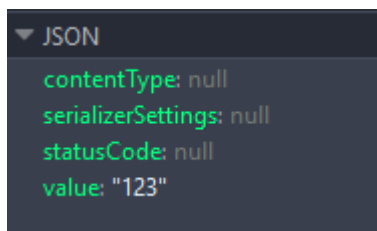


51

This was in a 4.6 MVC application but now with .NET Core I don't seem to have this `IActionResult` I have `ActionResult` and using like this:

```
public ActionResult IsAuthenticated()
{
    return Ok(Json("123"));
}
```

But the response from the server is weird, as in the image below:



I just want the Web API controller to return JSON with a HTTP status code like I did in Web API 2.

c#

json

asp.net-core

asp.net-core-webapi

edited May 17 '18 at 10:05
Vadim Ovchinnikov

asked Feb 21 '17 at 6:26
Rossco



8,363

4

33

58



1,383

2

15

30

- 1 The "ok" methods return 200 as status code. The predefined methods cover all the common cases. To Return 201 (+header with new resource location), you use `CreatedAtRoute` method etc. – [Tseng](#) Feb 21 '17 at 6:57

9 Answers



The most basic version responding with a `JsonResult` is:

153



```
// GET: api/authors
[HttpGet]
public JsonResult Get()
{
    return Json(_authorRepository.List());
}
```

However, this isn't going to help with your issue because you can't explicitly deal with your own response code.

The way to get control over the status results, is you need to return a `ActionResult` which is where you can then take advantage of the `StatusCodeResult` type.

for example:

```
// GET: api/authors/search?nameLike=foo
[HttpGet("Search")]
public IActionResult Search(string nameLike)
{
    var result = _authorRepository.GetByNameSubstring(nameLike);
    if (!result.Any())
    {
        return NotFound(nameLike);
    }
    return Ok(result);
}
```

Note both of these above examples came from a great guide available from Microsoft Documentation: [Formatting Response Data](#)

Extra Stuff

The issue I come across quite often is that I wanted more granular control over my WebAPI rather than just go with the defaults configuration from the "New Project" template in VS.

Let's make sure you have some of the basics down...

Step 1: Configure your Service

In order to get your ASP.NET Core WebAPI to respond with a JSON Serialized Object along full control of the status code, you should start off by making sure that you have included the `AddMvc()` service in your `ConfigureServices` method usually found in `Startup.cs`.

It's important to note that `AddMvc()` will automatically include the Input/Output Formatter for JSON along with responding to other request types.

If your project requires **full control** and you want to strictly define your services, such as how your WebAPI will behave to various request types including `application/json` and not respond to other request types (such as a standard browser request), you can define it manually with the following code:

```
public void ConfigureServices(IServiceCollection services)
{
    // Build a customized MVC implementation, without using the default AddMvc(),
    // instead use AddMvcCore().
    // https://github.com/aspnet/Mvc/blob/dev/src/Microsoft.AspNetCore.Mvc/MvcServiceCollectionEx

    services
        .AddMvcCore(options =>
        {
            options.RequireHttpsPermanent = true; // does not affect api requests
            options.RespectBrowserAcceptHeader = true; // false by default
            //options.OutputFormatters.RemoveType<HttpNoContentOutputFormatter>();

            //remove these two below, but added so you know where to place them...
            options.OutputFormatters.Add(new YourCustomOutputFormatter());
            options.InputFormatters.Add(new YourCustomInputFormatter());
        })
        //.AddApiExplorer()
        //.AddAuthorization()
        .AddFormatterMappings()
        //.AddCacheTagHelper()
        //.AddDataAnnotations()
```

```
        //.AddCors()  
        .AddJsonFormatters(); // JSON, or you can build your own custom one (above)  
    }
```

You will notice that I have also included a way for you to add your own custom Input/Output formatters, in the event you may want to respond to another serialization format (protobuf, thrift, etc).

The chunk of code above is mostly a duplicate of the `AddMvc()` method. However, we are implementing each "default" service on our own by defining each and every service instead of going with the pre-shipped one with the template. I have added the repository link in the code block, or you can check out `AddMvc()` [from the GitHub repository](#).

Note that there are some guides that will try to solve this by "undoing" the defaults, rather than just not implementing it in the first place... If you factor in that we're now working with Open Source, this is redundant work, bad code and frankly an old habit that will disappear soon.

Step 2: Create a Controller

I'm going to show you a really straight-forward one just to get your question sorted.

```
public class FooController  
{  
    [HttpPost]  
    public async Task<IActionResult> Create([FromBody] Object item)  
    {  
        if (item == null) return BadRequest();  
  
        var newItem = new Object(); // create the object to return  
        if (newItem != null) return Ok(newItem);  
  
        else return NotFound();  
    }  
}
```

Step 3: Check your Content-Type and Accept

You need to make sure that your `Content-Type` and `Accept` headers in your **request** are set properly. In your case (JSON), you will want to set it up to be `application/json`.

If you want your WebAPI to respond as JSON as default, regardless of what the request header is specifying you can do that in a **couple ways**.

Way 1 As shown in the article I recommended earlier ([Formatting Response Data](#)) you could force a particular format at the Controller/Action level. I personally don't like this approach... but here it is for completeness:

Forcing a Particular Format If you would like to restrict the response formats for a specific action you can, you can apply the [Produces] filter. The [Produces] filter specifies the response formats for a specific action (or controller). Like most Filters, this can be applied at the action, controller, or global scope.

```
[Produces("application/json")]  
public class AuthorsController
```

The [Produces] filter will force all actions within the `AuthorsController` to return JSON-formatted responses, even if other formatters were configured for the application and the client provided an `Accept` header requesting a different, available format.

Way 2 My preferred method is for the WebAPI to respond to all requests with the format requested. However, in the event that it doesn't accept the requested format, then **fall-back** to a default (ie. JSON)

First, you'll need to register that in your options (we need to rework the default behavior, as noted earlier)

```
options.RespectBrowserAcceptHeader = true; // false by default
```

Finally, by simply re-ordering the list of the formatters that were defined in the services builder, the web host will default to the formatter you position at the top of the list (ie position 0).

More information can be found in this [.NET Web Development and Tools Blog entry](#).

edited Apr 7 '17 at 11:31

answered Feb 21 '17 at 6:43



Svek

7,657

3 25 51

Thanx so much for the effort you put in. Your answer inspired me to implement `ActionResult` with the `return Ok(new {response = "123"})`;
Cheers! – [Rossco](#) Feb 21 '17 at 7:13

1 @Rossco No problem. Hopefully the rest of the code will help guide you as your project develops. – [Svek](#) Feb 21 '17 at 7:17

1 To extend this topic, I created an additional and more complete guide to implementing the WebAPI here: stackoverflow.com/q/42365275/3645638 – [Svek](#) Apr 29 '17 at 3:40

On setting: `RespectBrowserAcceptHeader = true`; You are not explaining why you are doing it, and it is typically unnecessary and wrong to do so. Browsers ask for html, and hence they shouldn't affect formatter selection in anyway (that chrome unfortunately does by asking for XML). In short it's something I would keep off, and the fallback you are specifying is already the default behavior – [Yishai Galatzer](#) Jun 22 '17 at 23:21

@YishaiGalatzer The main theme of that part of my answer was to highlight how to unburden the default middleware between the client and the API logic. In my opinion, `RespectBrowserAcceptHeader` is critical when implementing use of an alternative serializer or more commonly, when you want to make sure that your clients are not sending malformed requests. Hence, I emphasized *"If your project requires **full control** and you want to strictly define your service"* and note the highlighted block quote above that statement too. – [Svek](#) Jun 24 '17 at 3:51

48

You have predefined methods for most common status codes.

- `Ok(result)` returns `200` with response
- `CreatedAtRoute` returns `201` + new resource URL
- `NotFound` returns `404`
- `BadRequest` returns `400` etc.

See [BaseController.cs](#) and [Controller.cs](#) for a list of all methods.

But if you really insist you can use `StatusCode` to set a custom code, but you really shouldn't as it makes code less readable and you'll have to repeat code to set headers (like for `CreatedAtRoute`).

```
public ActionResult IsAuthenticated()
{
    return StatusCode(200, "123");
}
```

edited Sep 29 at 13:36

answered Feb 21 '17 at 7:04



[Tseng](#)

40.3k 5 119 148

this gave me insight to my response below. Thank you – [Oge Nwike](#) May 21 at 15:56

This code isn't correct for ASP.NET Core 2.2. I just have tried it and it serializes into JSON the `ActionResult` created by the `Json()` method. It doesn't include the "123" string directly. – [amedina](#) Sep 29 at 13:33

1 @amedina: My bad, just remove the `Json(...)` and pass the string to `StatusCode` – [Tseng](#) Sep 29 at 13:35

▲
34
▼

With **ASP.NET Core 2.0**, the ideal way to return object from Web API (which is unified with MVC and uses same base class `Controller`) is

```
public IActionResult Get()
{
    return new OkObjectResult(new Item { Id = 123, Name = "Hero" });
}
```

Notice that

1. It returns with `200 OK` status code (it's an `Ok` type of `ObjectResult`)
2. It does content negotiation, i.e. it'll return based on `Accept` header in request. If `Accept: application/xml` is sent in request, it'll return as `XML`. If nothing is sent, `JSON` is default.

If it needs to send **with specific status code**, use `ObjectResult` or `StatusCode` instead. Both does the same thing, and supports content negotiation.

```
return new ObjectResult(new Item { Id = 123, Name = "Hero" }) { StatusCode = 200 };
return StatusCode( 200, new Item { Id = 123, Name = "Hero" });
```

If you specifically want to **return as JSON**, there are couple of ways

```
//GET http://example.com/api/test/asjson
[HttpGet("AsJson")]
public JsonResult GetAsJson()
{
    return Json(new Item { Id = 123, Name = "Hero" });
}
```

```
//GET http://example.com/api/test/withproduces
[HttpGet("WithProduces")]
[Produces("application/json")]
public Item GetWithProduces()
{
    return new Item { Id = 123, Name = "Hero" };
}
```

Notice that

1. Both enforces `JSON` in two different ways.

2. Both ignores content negotiation.
3. First method enforces JSON with specific serializer `Json(object)` .
4. Second method does the same by using `Produces()` attribute (which is a `ResultFilter`) with `contentType = application/json`

Read more about them in [the official docs](#). Learn about [filters here](#).

The simple model class that is used in the samples

```
public class Item
{
    public int Id { get; set; }
    public string Name { get; set; }
}
```

edited Feb 2 '18 at 8:10

answered Feb 2 '18 at 7:59



Arghya C

7,068 1 29 47

8 This is a good answer because it focuses on the question and explains some practicalities in brief. – [netfed](#) Apr 14 '18 at 3:28

▲ The easiest way I came up with is :

21

▼

```
var result = new Item { Id = 123, Name = "Hero" };

return new JsonResult(result)
{
    StatusCode = StatusCodes.Status201Created // Status code here
};
```

edited Jul 5 at 9:09


answered Dec 8 '17 at 14:58



Gerald Hughes

2,406 8 43 90

2 I think this is better than the answer from [@tseng](#) because his solution includes duplicated fields for Status Codes etc. – [Christian Sauer](#) Dec 18 '17 at 10:29

- 2 One improvement you can make is to use the StatusCodes defined in Microsoft.AspNetCore.Http like this: `return new JsonResult(new { }) { StatusCode = StatusCodes.Status404NotFound };` – [Bryan Bedard](#) May 24 '18 at 18:55 
- 1 This should be the accepted answer. Although there are ways to universally setup the json, sometimes we have to work with legacy endpoints and the settings can be different. Until we can stop supporting some legacy endpoints, this is the ultimate way to have full control – [pqsk](#) Feb 25 at 22:51

▲ This is my easiest solution:

9

▼

```
public IActionResult InfoTag()
{
    return Ok(new {name = "Fabio", age = 42, gender = "M"});
}
```

or

```
public IActionResult InfoTag()
{
    return Json(new {name = "Fabio", age = 42, gender = "M"});
}
```

edited Jun 15 '18 at 13:05

answered Jun 15 '18 at 10:15



Fabio

91 1 3

▲ Instead of using 404/201 status codes using enum

3

▼

```
public async Task<IActionResult> Login(string email, string password)
{
    if (string.IsNullOrWhiteSpace(email) || string.IsNullOrWhiteSpace(password))
    {
        return StatusCode((int)HttpStatusCode.BadRequest, Json("email or password is
null"));
    }

    var user = await _userManager.FindByEmailAsync(email);
    if (user == null)
    {
        return StatusCode((int)HttpStatusCode.BadRequest, Json("Invalid Login and/or
```

```
password"));

    }
    var passwordSignInResult = await _signInManager.PasswordSignInAsync(user,
password, isPersistent: true, lockoutOnFailure: false);
    if (!passwordSignInResult.Succeeded)
    {
        return StatusCode((int)HttpStatusCode.BadRequest, Json("Invalid Login and/or
password"));
    }
    return StatusCode((int)HttpStatusCode.OK, Json("Sucess !!!"));
}
```

answered Feb 3 at 3:22



Enum is a great idea !. – [Bhimbim](#) Apr 29 at 5:29

Awesome answers I found here and I also tried this return statement see `StatusCode(whatever code you wish)` and it worked!!!

0

```
return Ok(new {
    Token = new JwtSecurityTokenHandler().WriteToken(token),
    Expiration = token.ValidTo,
    username = user.FullName,
    StatusCode = StatusCode(200)
});
```

answered May 21 at 15:53



What I do in my Asp Net Core Api applications it is to create a class that extends from `ObjectResult` and provide many constructors to customize the content and the status code. Then all my Controller actions use one of the constructors as appropriate. You can take a look at my implementation at: <https://github.com/melardev/AspNetCoreApiPaginatedCrud>

and

<https://github.com/melardev/ApiAspCoreEcommerce>

here is how the class looks like(go to my repo for full code):

```
public class StatusCodeAndDtoWrapper : ObjectResult
{

    public StatusCodeAndDtoWrapper(AppResponse dto, int statusCode = 200) : base(dto)
    {
        StatusCode = statusCode;
    }

    private StatusCodeAndDtoWrapper(AppResponse dto, int statusCode, string message) :
    base(dto)
    {
        StatusCode = statusCode;
        if (dto.FullMessages == null)
            dto.FullMessages = new List<string>(1);
        dto.FullMessages.Add(message);
    }

    private StatusCodeAndDtoWrapper(AppResponse dto, int statusCode, ICollection<string>
    messages) : base(dto)
    {
        StatusCode = statusCode;
        dto.FullMessages = messages;
    }
}
```

Notice the base(dto) you replace dto by your object and you should be good to go.

answered Mar 9 at 19:41



Melardev

439 3 12

Please refer below code, You can manage multiple status code with different type JSON

-1

```
public async Task<HttpResponseMessage> GetAsync()
{
    try
```

```
{
    using (var entities = new DbEntities())
    {
        var resourceModelList = entities.Resources.Select(r=> new
ResourceModel{Build Your Resource Model}).ToList();

        if (resourceModelList.Count == 0)
        {
            return this.Request.CreateResponse<string>(HttpStatusCode.NotFound, "No
resources found.");
        }

        return this.Request.CreateResponse<List<ResourceModel>>(HttpStatusCode.OK,
resourceModelList, "application/json");
    }
}
catch (Exception ex)
{
    return this.Request.CreateResponse<string>(HttpStatusCode.InternalServerError,
"Something went wrong.");
}
}
```

answered Feb 21 '17 at 7:18



Suyog

151 6

8 No. This is bad. – Phillip Copley Oct 3 '17 at 17:43
