Web API Best Practices

STEVE SMITH

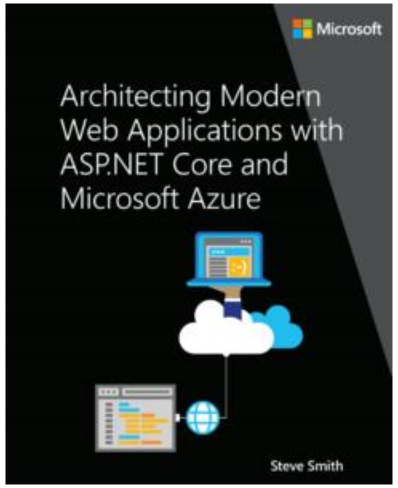
ARDALIS.COM | @ARDALIS | STEVE@DEVIQ.COM

DEVIQ.COM

Learn More After Today

1) DevIQ
- ASP.NET Core Quick Start — http://aspnetcorequickstart.com **DEVINTFALL17** 20% OFF!

2) Microsoft FREE eBook/Sample App
- eShopOnWeb eCommerce Sample — https://ardalis.com/architecture-ebook


Microsoft
Architecting Modern Web Applications with ASP.NET Core and Microsoft Azure
Steve Smith

3) Weekly Dev Tips Podcast / Newsletter
- http://ardalis.com/tips

4) Contact me for mentoring/training for your company/team
- http://ardalis.com

Web API Design

Representational State Transfer (REST)

"An architectural style for building distributed systems based on hypermedia"

Open standards-based

Technology-agnostic

Client issues a request to a URI that represents a resource;

- Request verb that indicates the operation to perform on the resource.
- Request body includes the data required for the operation.

REST-based APIs are stateless; each request may be handled by a different server-side resource

https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design

URI Design Considerations

URI values should correspond to nouns

- E.g. /customers, /authors, /orders

URI values should typically be plural (when referring to collections)

- Again, /customers, /authors, /orders

Requests for individual resources should append an identifier:

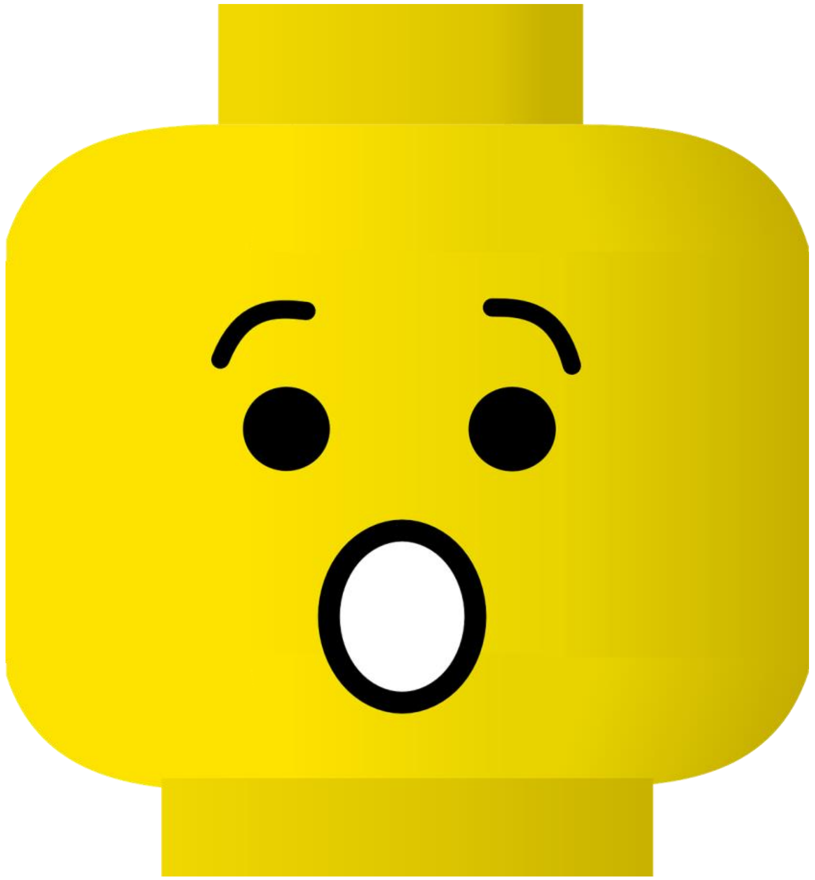
- Example: /customers/1, /orders/00234

Principle of Least **Astonishment**

Try not to surprise your client with how your API works!

Keep it **SIMPLE**

Keep it **CONSISTENT**



Don’t Expose Business/Data Model

Avoid coupling your Web API directly to your data model

API design, including URIs, may not may 1:1 to domain objects or database tables.

Example:

`POST /orders`

May map to a NewOrderRequest on the server that triggers processing payment, checking inventory, sending notifications, etc.

Or it could just insert a record in the Orders table.

It should be able to do either without the API changing.

Use a standard URI structure for subcollections

For performance or other reasons, might not return full object tree with root-level request:

```
GET /customers/1
{
  “id”:”1”,
  “name”: “Steve Smith”
}
```

To get the customer’s orders:

```
GET /customers/1/orders
[{“id”:”123”,”customerId”:”1”, …}, {“id”:”234”,”customerId”:”1”, …}]
```

Avoid Deeply Nested URI Structures

OK

```
/customers

/customers/1

/customers/1/orders
```

TOO MUCH

```
/customers/1/orders/123                      (instead: /orders/123)

/customers/1/orders/123/items/1/products/2   (instead: /products/2)
```

Hypertext as the Engine of Application State (HATEOAS)

Less commonly implemented aspect of REST approach

Currently no standards or specifications defining implementation

Basic idea: Each response includes links defining available requests on a given resource

Example:

```
GET /customers/1
```

Response includes customer data, as well as links to:

Update the customer	**Delete the customer**	**List customer orders**
List customer addresses	**Add an address**	**Add an order**

Standard Verbs and Behaviors

GET	Fetch a resource (or collection of resources)
PUT	Update a resource.
POST	Create a new resource.
DELETE	Delete a resource.

Safe and Idempotent API Requests

Safe requests are requests that do not change resources, and which can be made repeatedly without impact. Think of safe requests as read-only operations.

An idempotent HTTP method can be called multiple times without changing the expected response.

Are these the same?

HTTP Verb Idempotency/Safety

VERB	Idempotent?	Safe?
GET	Yes	Yes
PUT	Yes	No
POST	No	No
DELETE*	Yes	No

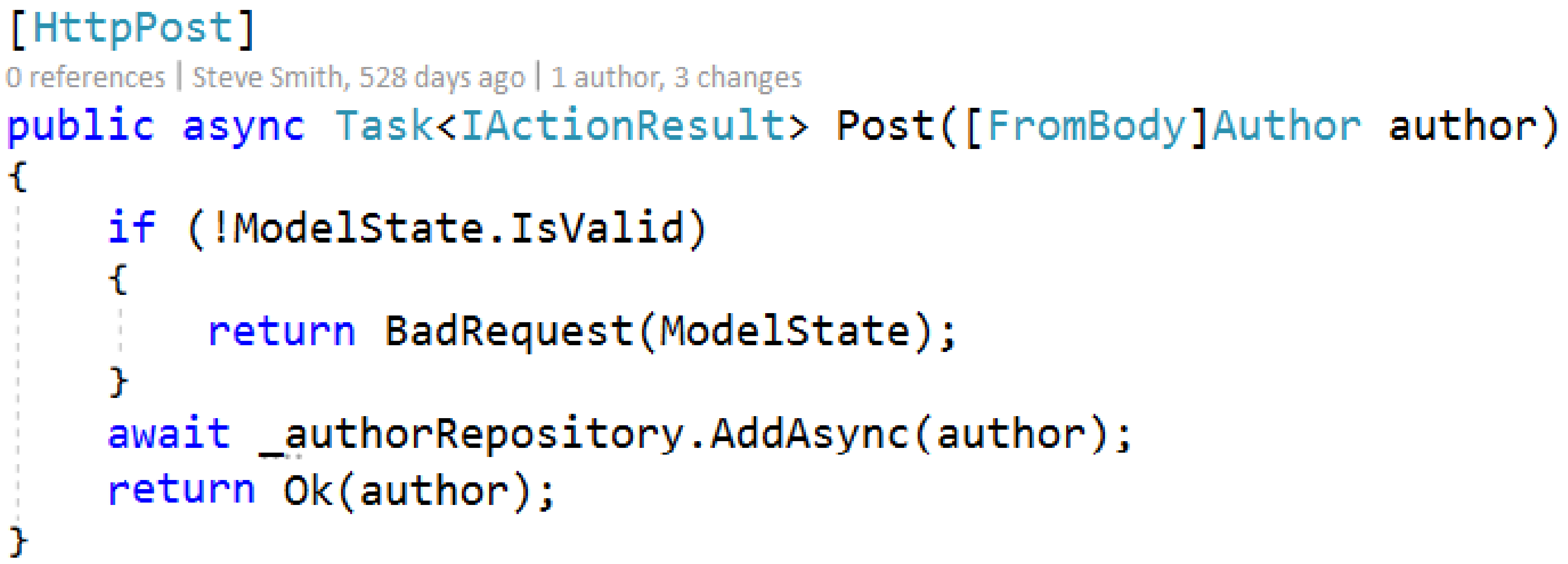
*Decide if a DELETE for a missing id should return a 404 or not. If so, then it won't be Idempotent.

http://restcookbook.com/HTTP%20Methods/idempotency/

Web API Implementation

Use Model Validation

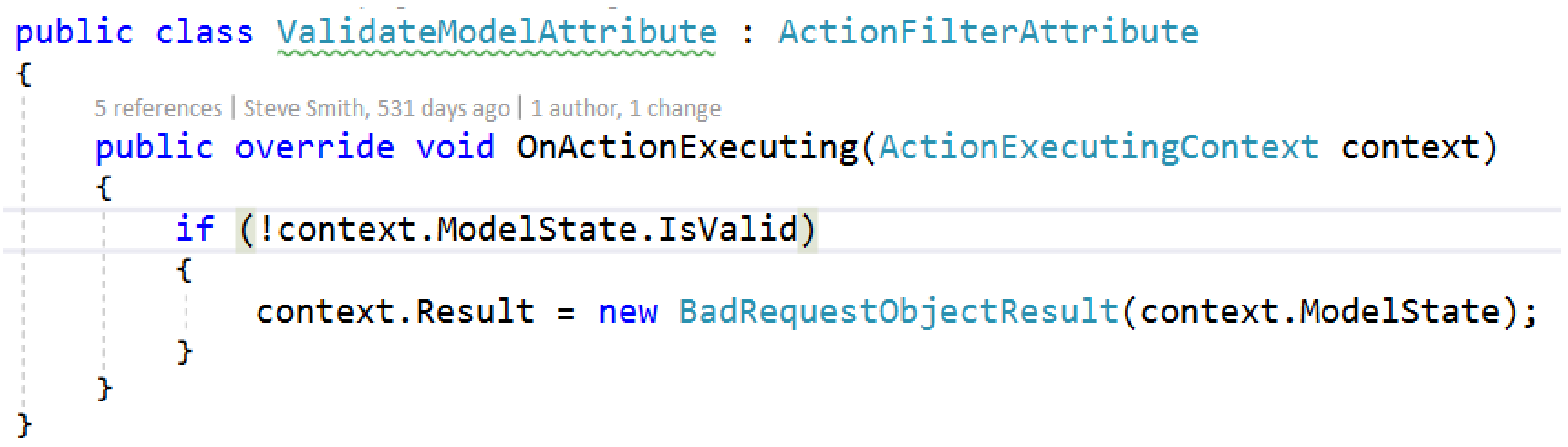
Always check if `Model.IsValid` before performing unsafe operations



```
[HttpPost]
0 references | Steve Smith, 528 days ago | 1 author, 3 changes
public async Task<IActionResult> Post([FromBody]Author author)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }
    await _authorRepository.AddAsync(author);
    return Ok(author);
}
```

Use Filters To Represent Policies

Validate Model State using a filter (globally, per-controller, or per-action)

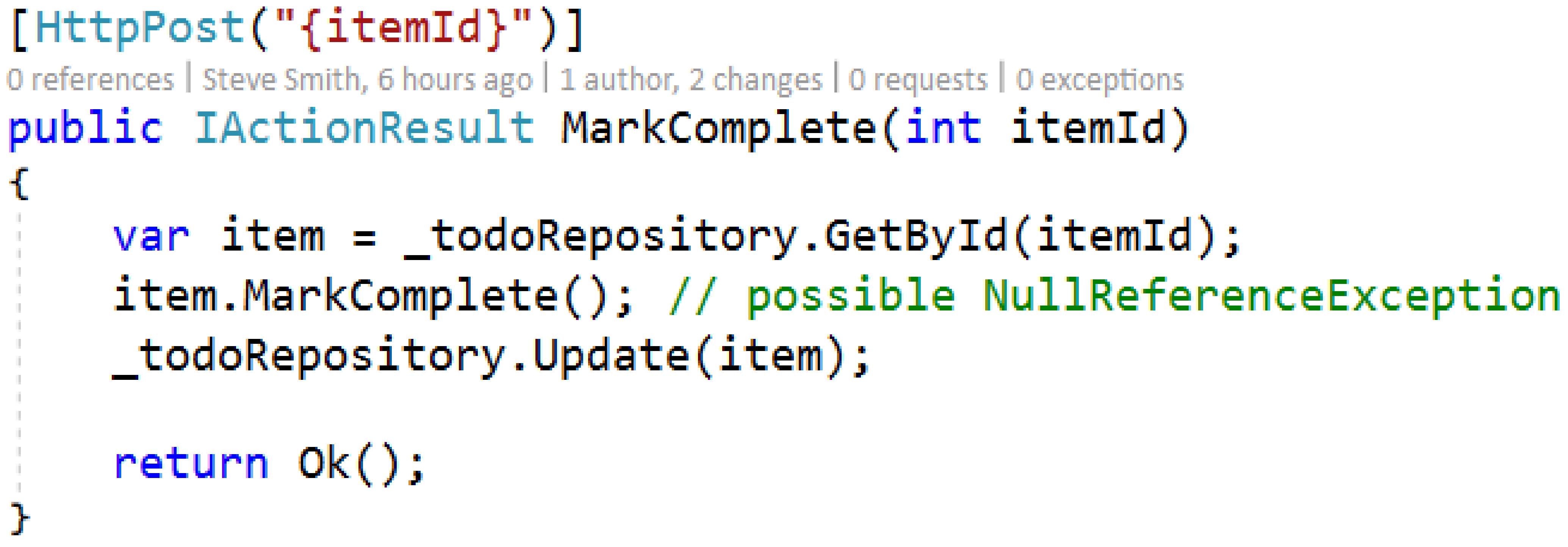


```
public class ValidateModelAttribute : ActionFilterAttribute
{
    5 references | Steve Smith, 531 days ago | 1 author, 1 change
    public override void OnActionExecuting(ActionExecutingContext context)
    {
        if (!context.ModelState.IsValid)
        {
            context.Result = new BadRequestObjectResult(context.ModelState);
        }
    }
}
```

Use Proper HTTP Status Codes as Results

200 OK	Request was successful; body has response.
201 OK	POST or PUT was successful; body has latest representation.
204 OK	DELETE was successful; resource was deleted.
400 BAD REQUEST	The request was invalid or cannot otherwise be served.
401 UNAUTHORIZED	Authorization failed or authentication details not supplied.
404 NOT FOUND	The URI requested or the resource requested doesn’t exist.
500 Internal Server Error	Something very bad happened. Unhandled exceptions lead to this.

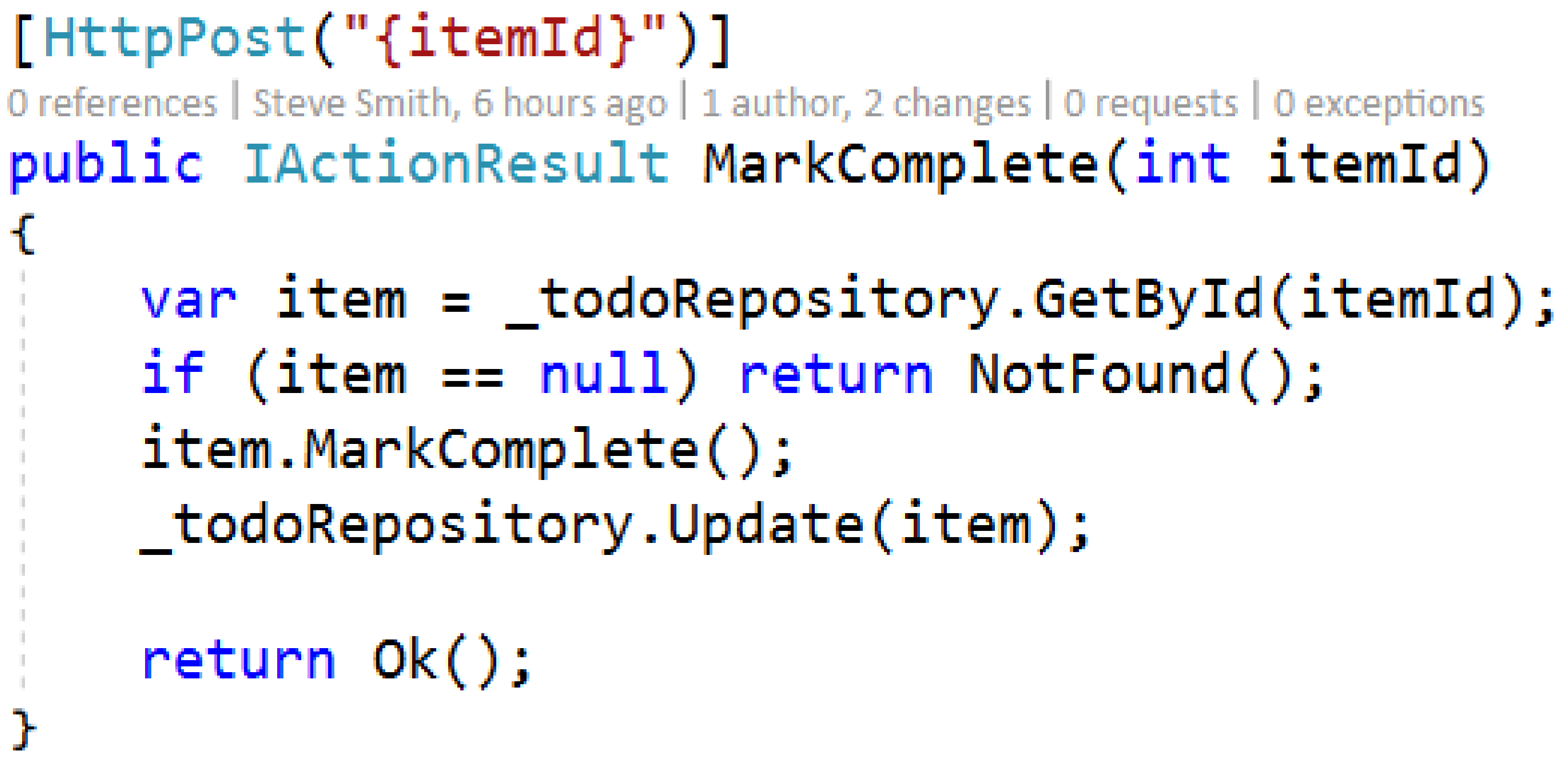
Prefer NotFound to NullReferenceException



```
[HttpPost("{itemId}")]
0 references | Steve Smith, 6 hours ago | 1 author, 2 changes | 0 requests | 0 exceptions
public IActionResult MarkComplete(int itemId)
{
    var item = _todoRepository.GetById(itemId);
    item.MarkComplete(); // possible NullReferenceException
    _todoRepository.Update(item);

    return Ok();
}
```

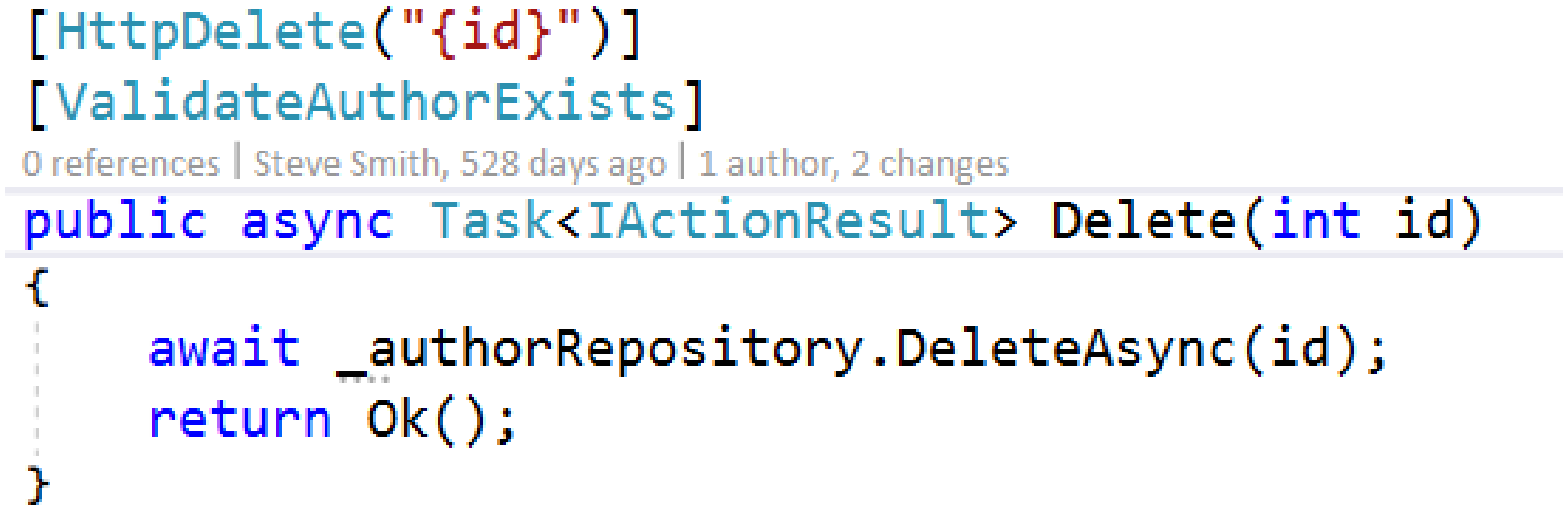
Prefer NotFound to NullReferenceException



```
[HttpPost("{itemId}")]
0 references | Steve Smith, 6 hours ago | 1 author, 2 changes | 0 requests | 0 exceptions
public IActionResult MarkComplete(int itemId)
{
    var item = _todoRepository.GetById(itemId);
    if (item == null) return NotFound();
    item.MarkComplete();
    _todoRepository.Update(item);

    return Ok();
}
```

Use a filter to confirm existence

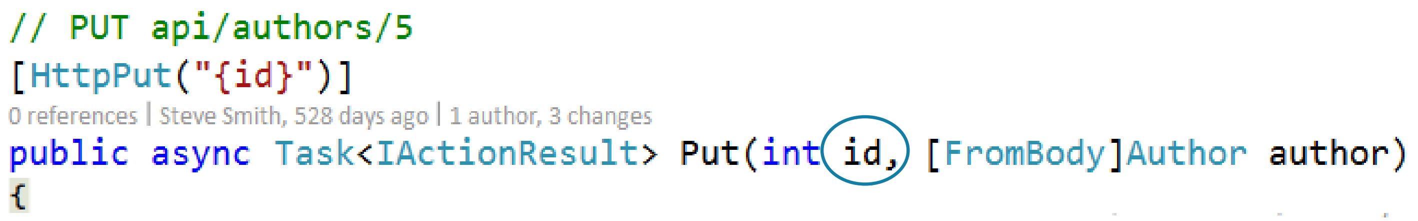


```
[HttpDelete("{id}")]
[ValidateAuthorExists]
0 references | Steve Smith, 528 days ago | 1 author, 2 changes
public async Task<IActionResult> Delete(int id)
{
    await _authorRepository.DeleteAsync(id);
    return Ok();
}
```

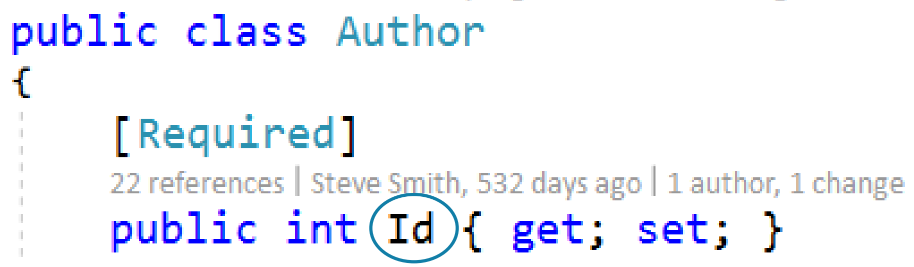
Avoid Duplicating Data within Requests

Don't ask for an ID in the route and also in the BindingModel

- Unless you're going to allow updates to a resource's ID!



```
// PUT api/authors/5
[HttpPut("{id}")]
0 references | Steve Smith, 528 days ago | 1 author, 3 changes
public async Task<IActionResult> Put(int id, [FromBody]Author author)
{
```



```
public class Author
{
    [Required]
    22 references | Steve Smith, 532 days ago | 1 author, 1 change
    public int Id { get; set; }
```

Which value should you use? How do you decide?

- Best to use a model type that doesn't include the ID if it's redundant

Use DTOs Appropriately

Avoid using domain objects or data entities as your API inputs or outputs.

- Doing so exposes your app's internal state and can be a security risk

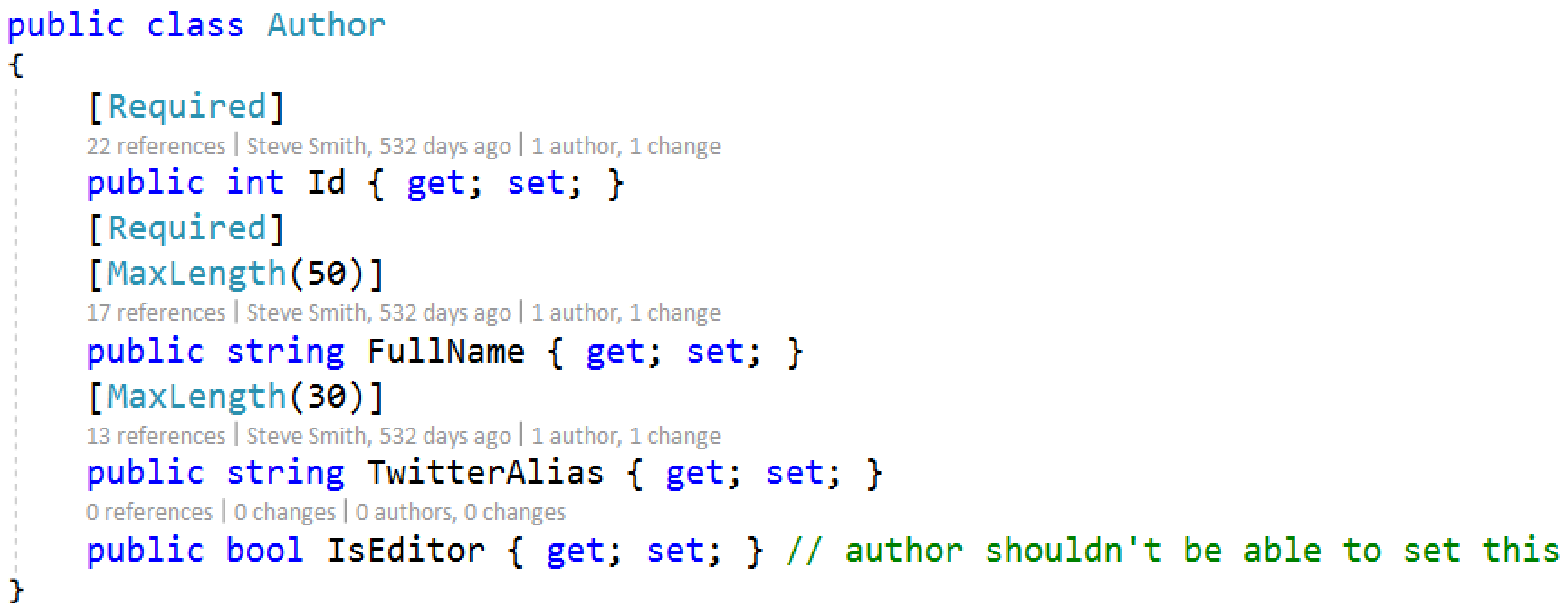
Be careful to avoid creating DTO types that inadvertently reference non-DTO types.

- Look for using statements in your DTO files that shouldn't be there

If specifying ID on DTOs, may not make sense to use for new object requests (POSTs)

- Consider having separate NewResourceDTO and ResourceDTO types
- ResourceDTO can inherit from NewResourceDTO and simply add the Id property

Non-DTOs May Expose Sensitive Data



```
public class Author
{
    [Required]
    22 references | Steve Smith, 532 days ago | 1 author, 1 change
    public int Id { get; set; }
    [Required]
    [MaxLength(50)]
    17 references | Steve Smith, 532 days ago | 1 author, 1 change
    public string FullName { get; set; }
    [MaxLength(30)]
    13 references | Steve Smith, 532 days ago | 1 author, 1 change
    public string TwitterAlias { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public bool IsEditor { get; set; } // author shouldn't be able to set this
}
```

Post-Redirect-Get (PRG) Pattern

Overview

- Client POSTs to Server
- Server performs requested operation and returns a Redirect (302) to new URI
- Client GETs new URI

This pattern is most appropriate to MVC non-API apps.

- One of its primary benefits is that it eliminates browser refreshes from reissuing POST commands.
- Not generally an issue with Web APIs

REST services should (typically) return the resource in the body of POST commands

What to Return?

Object

- Author, Customer, or void
- Automatically wrapped in a result (or

Encoding-Specific

- return Json(model); // `JsonResult`

`IActionResult`

- return Ok(model);
- return NotFound();
- return BadRequest();

Prefer IActionResult; Support Content Negotiation

Requests can include Accept header specifying content they want/support

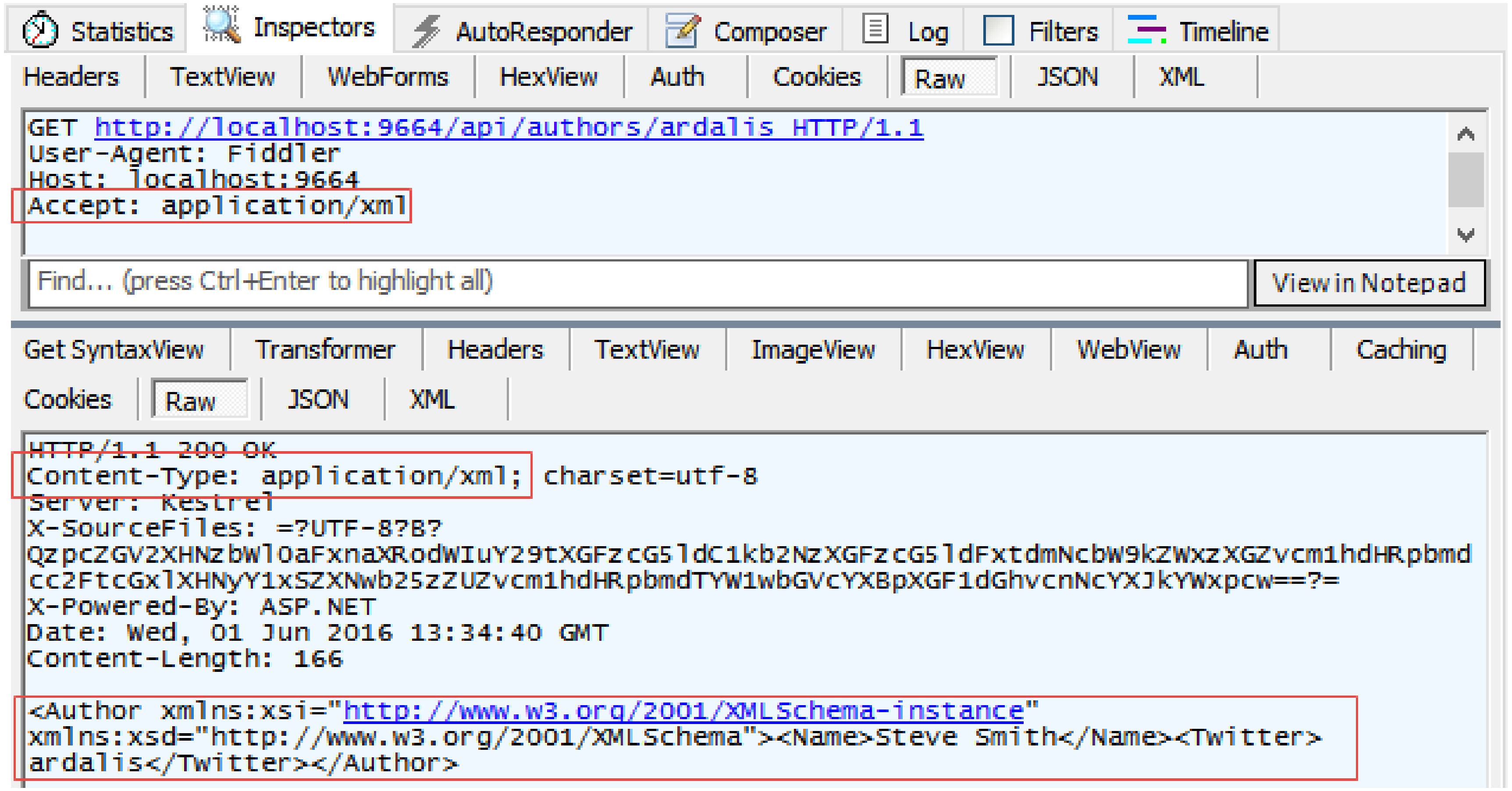
Web API will attempt to comply with specified content format

Support JSON (default) and XML:

Add XML Serializers when adding MVC in ConfigureServices:

```
services.AddMvc()
    .AddXmlSerializerFormatters();
```

Content Negotiation In Action

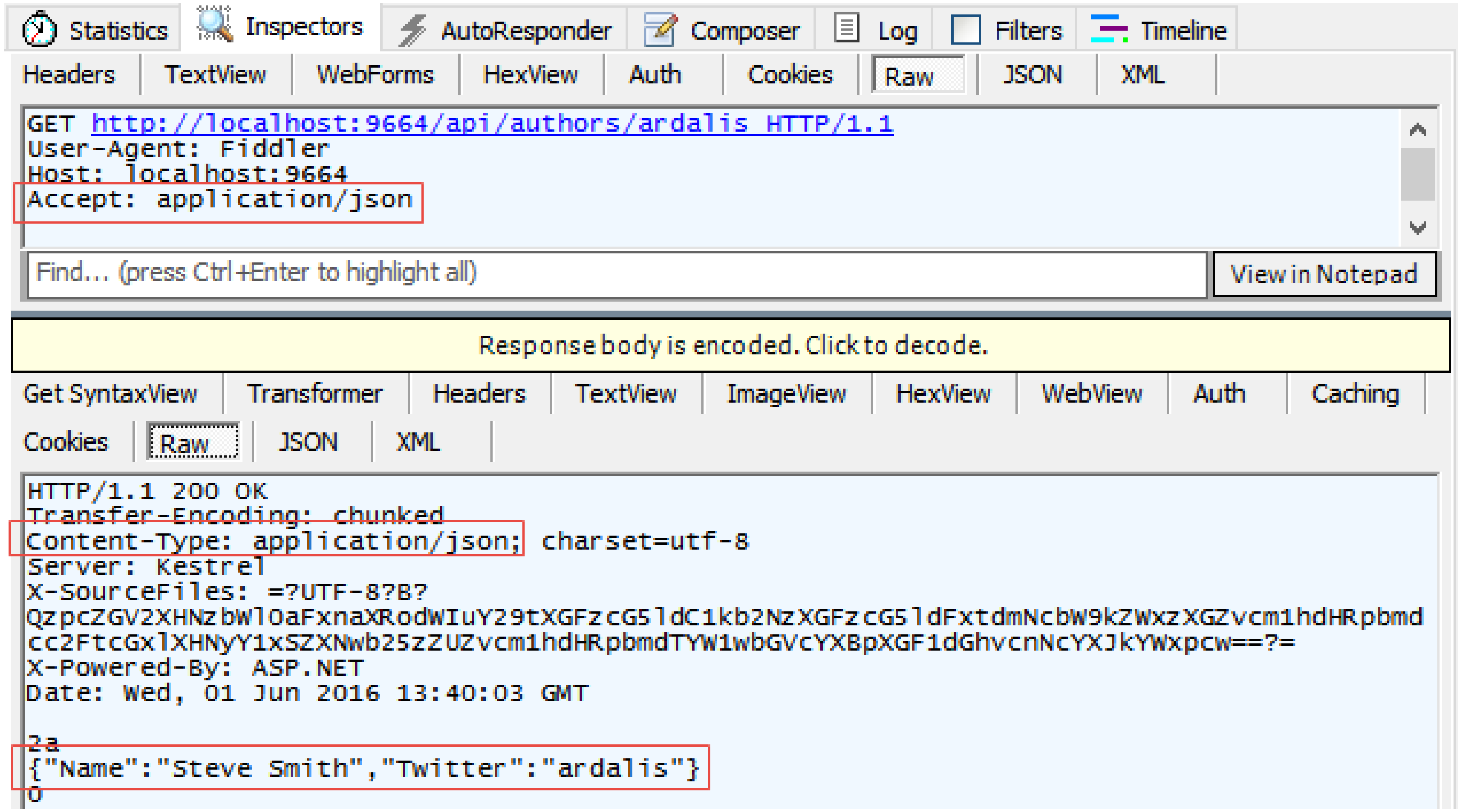


```
GET http://localhost:9664/api/authors/ardalis HTTP/1.1
User-Agent: Fiddler
Host: localhost:9664
Accept: application/xml
```

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=utf-8
Server: Kestrel
X-SourceFiles: =?UTF-8?B?
QzpcZGV2XHNzbWl0aFxnaXRodWIuY29tXGFzcG5ldC1kb2NzXGFzcG5ldFxtdmNcbW9kZWxzXGZvcm1hdHRpbmd
cc2FtcGxlXHNyY1xSZXNwb25zZUZvcm1hdHRpbmdTYW1wbGVcYXBpXGF1dGhvcnNcYXJkYWxpcw==?=
X-Powered-By: ASP.NET
Date: Wed, 01 Jun 2016 13:34:40 GMT
Content-Length: 166

<Author xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"><Name>Steve Smith</Name><Twitter>
ardalis</Twitter></Author>
```

Content Negotiation In Action



```
GET http://localhost:9664/api/authors/ardalis HTTP/1.1
User-Agent: Fiddler
Host: localhost:9664
Accept: application/json
```

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Content-Type: application/json; charset=utf-8
Server: Kestrel
X-SourceFiles: =?UTF-8?B?
QzpcZGV2XHNzbWl0aFxnaXRodWIuY29tXGFzcG5ldC1kb2NzXGFzcG5ldFxtdmNcbW9kZWxzXGZvcm1hdHRpbmd
cc2FtcGxlXHNyY1xSZXNwb25zZUZvcm1hdHRpbmdTYW1wbGVcYXBpXGF1dGhvcnNcYXJkYWxpcw==?=
X-Powered-By: ASP.NET
Date: Wed, 01 Jun 2016 13:40:03 GMT

2a
{"Name":"Steve Smith","Twitter":"ardalis"}
0
```

Documentation / Discoverability

Swagger   http://swagger.io

Now the **OpenAPI Specification**

Provide live, runtime documentation of your APIs

Ability to generate client libraries to assist in consuming your API

- NSwag - https://github.com/RSuter/NSwag

Adding Swagger to your Web API

Add Nuget package **Swashbuckle.AspNetCore**

Add Services in ConfigureServices:

```
services.AddSwaggerGen(c =>
    {
        c.SwaggerDoc("v1", new Info { Title = "My API", Version = "v1" });
    });
```

Add Middleware to Configure() (next slide)

Adding Swagger to your Web API (cont.)

```
public void Configure(IApplicationBuilder app)
{
    app.UseSwagger(); // Enable middleware to serve generated Swagger as a JSON
endpoint.

    // Enable middleware to serve swagger-ui specifying the Swagger JSON
endpoint.
    app.UseSwaggerUI(c =>
    {
        c.SwaggerEndpoint("/swagger/v1/swagger.json", "My API V1");
    });
    app.UseMvc();
}
```

Demo

WORKING WITH SWAGGER

Testing Web APIs

Kinds of Tests

Unit Tests

- Test a single unit – typically a method
- Only test your code, not infrastructure
- Limited usefulness for testing APIs

Integration Tests

- Test several methods and/or classes working together
- Useful for verifying infrastructure code works correctly

Functional Tests

- Test full application stack
- Slowest, often most brittle, but provide greatest confidence a particular user scenario works fully

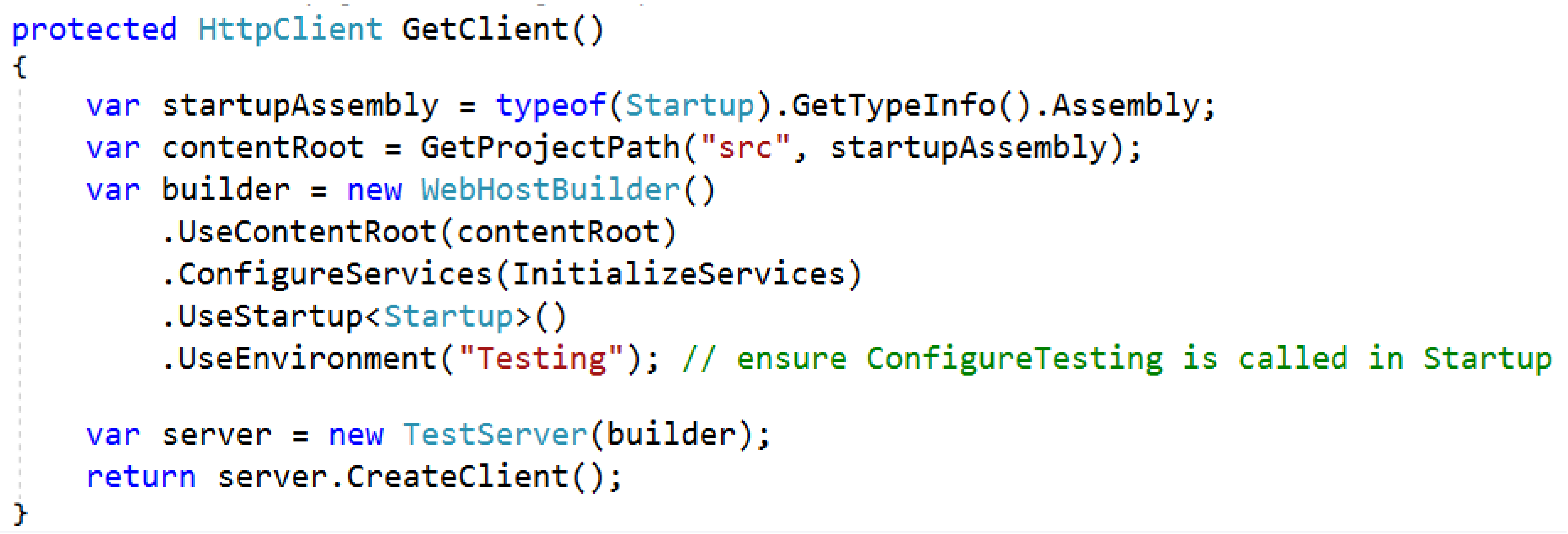
Unit test vs. Integration test

2 UNIT TESTS, 0 INTEGRATION TESTS
via reddit.com/r/programmerhumor

Test APIs with TestServer

Install `Microsoft.AspNetCore.TestHost` Nuget Package

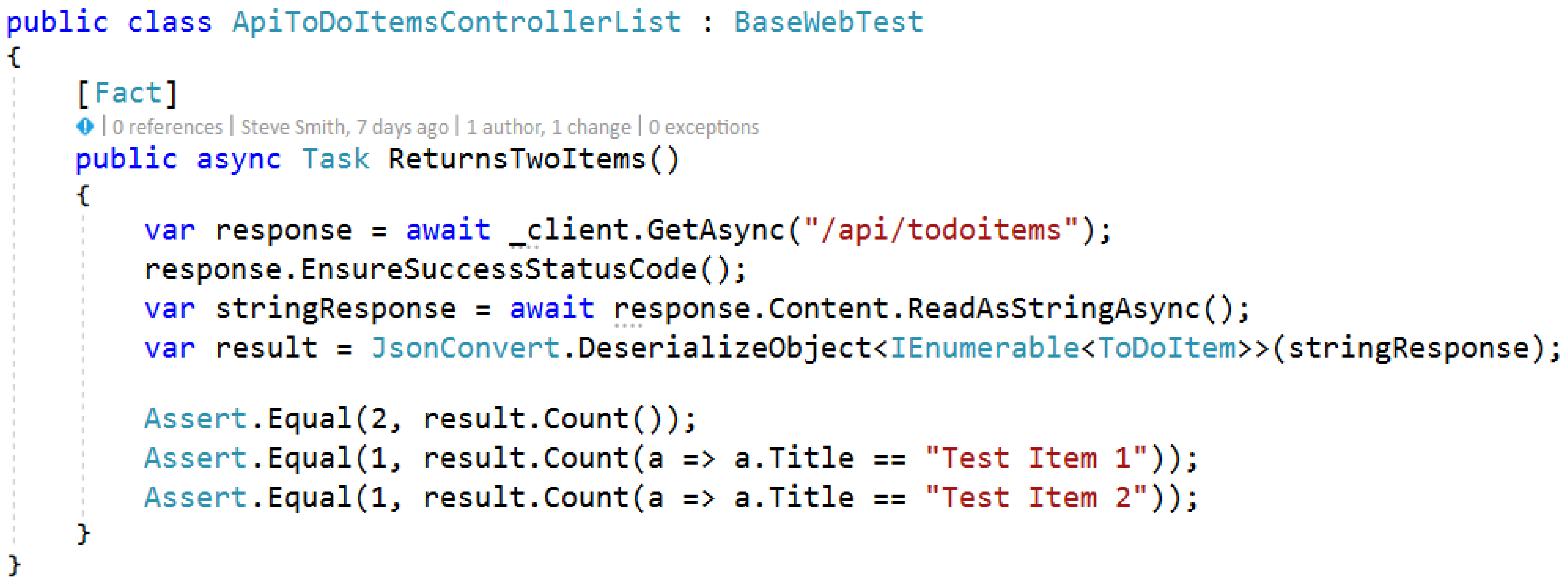
Configure with `WebHostBuilder`; use `HttpClient` to make requests to `TestServer` instance.



```
protected HttpClient GetClient()
{
    var startupAssembly = typeof(Startup).GetTypeInfo().Assembly;
    var contentRoot = GetProjectPath("src", startupAssembly);
    var builder = new WebHostBuilder()
        .UseContentRoot(contentRoot)
        .ConfigureServices(InitializeServices)
        .UseStartup<Startup>()
        .UseEnvironment("Testing"); // ensure ConfigureTesting is called in Startup

    var server = new TestServer(builder);
    return server.CreateClient();
}
```

Example Web API Test



```
public class ApiToDoItemsControllerList : BaseWebTest
{
    [Fact]
    | 0 references | Steve Smith, 7 days ago | 1 author, 1 change | 0 exceptions
    public async Task ReturnsTwoItems()
    {
        var response = await _client.GetAsync("/api/todoitems");
        response.EnsureSuccessStatusCode();
        var stringResponse = await response.Content.ReadAsStringAsync();
        var result = JsonConvert.DeserializeObject<IEnumerable<ToDoItem>>(stringResponse);

        Assert.Equal(2, result.Count());
        Assert.Equal(1, result.Count(a => a.Title == "Test Item 1"));
        Assert.Equal(1, result.Count(a => a.Title == "Test Item 2"));
    }
}
```

Demo

VALIDATING FILTERS PRODUCE SAME RESULTS AS INLINE CODE

Versioning Web APIs

No Versioning

Limit updates to non-destructive wherever possible

Coordinate with clients on breaking changes

Works best with internal APIs

URI Versioning

Example: `api.domain.com/v2/customers/1`

Previous versions work as before

Results in multiple URIs corresponding to same resource

Can complicate HATEOAS links

Can be unwieldy if API evolves quickly/frequently

Querystring Versioning

Example: `api.domain.com/customers/1?ver=2`

Previous versions work as before (default to 1 if omitted)

Can complicate HATEOAS links

Can be unwieldy if API evolves quickly/frequently

Header Versioning

Example: GET `api.domain.com/customers/1`
`Version-Header: 2`

Previous versions work as before (default to 1 if omitted)

HATEOAS links must use same header

Can be unwieldy if API evolves quickly/frequently

Media Type Versioning

Example: GET `api.domain.com/customers/1`
`Accept: vnd.domain.v2+json`

Response includes header indicating version provided

Previous versions work as before (default to 1 if omitted)

Works well with HATEOAS links (can include MIME types)

Versioning Considerations

Consider performance impact, especially for web server and proxy server caching.

- Header and Media Type versioning is less cache friendly than other techniques

Consider whether you will version your entire API (simplest) or resource by resource (generally not recommended).

Avoid making breaking changes to your API as much as possible. **No versioning option is without its problems.**

Securing Web APIs

Use HTTPS

(seriously, just use it)

Windows Auth

Simplest

Well-known

Only works on Windows and within an intranet.

IdentityServer 4

An OpenID Connect and OAuth 2.0 framework for ASP.NET Core 2.

Separate Authentication Service

Single Sign-On Support

Access Control for APIs, including tokens for:

- Server-to-Server clients
- Web clients and SPAs
- Native/Mobile apps

Free, Open Source

Learn more: http://docs.identityserver.io/en/release/

Web Tokens (JWT)

Roll your own using available packages:

```
<PackageReference Include="Microsoft.AspNetCore.Authentication.JwtBearer" Version="2.0.0" />
<PackageReference Include="System.IdentityModel.Tokens.Jwt" Version="5.1.4" />
```

Great article on this topic

- http://www.blinkingcaret.com/2017/09/06/secure-web-api-in-asp-net-core/

Steps

- Authenticate user and issue token. Store in client (local storage for browser).
- Add token in header on subsequent requests
- Validate token on server using middleware; return 401 if not valid

JWT Demo

Resources

Online Courses (Pluralsight and DevIQ)

SOLID Principles of OO Design	http://bit.ly/SOLID-OOP
N-Tier Architecture in C#	http://bit.ly/PS-NTier1 and http://bit.ly/PS-NTier2
DDD Fundamentals	http://bit.ly/ddd-fundamentals
ASP.NET Core Quick Start	http://aspnetcorequickstart.com/ **DEVINTFALL17** 20% OFF!

Other Resources

Weekly Dev Tips Podcast	http://www.weeklydevtips.com/
Microsoft Architecture eBook/sample	http://aka.ms/WebAppArchitecture

- Securing Web API in ASP.NET Core
  - http://www.blinkingcaret.com/2017/09/06/secure-web-api-in-asp-net-core/