**XML + JSON Serialization in ASP .NET Core**

By Shahed C on June 17, 2019

[2 Replies](https://wakeupandcode.com/xml-json-serialization-in-asp-net-core/#comments)

This is the **twenty-fourth**of a [series of posts](https://wakeupandcode.com/aspnetcore/#aspnetcore2019) on ASP .NET Core in 2019. In this series, we’ll cover 26 topics over a span of 26 weeks from January through June 2019, titled **A-Z of ASP .NET Core!**

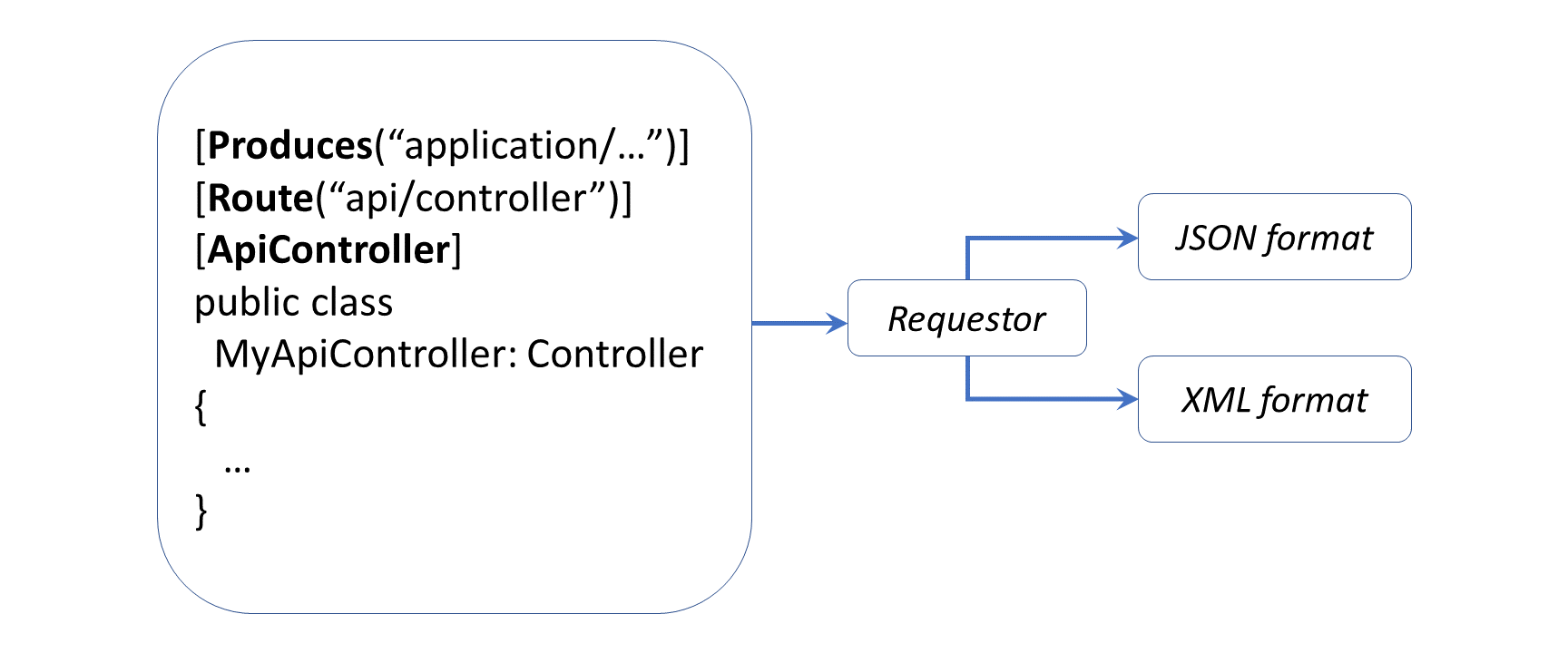
** A – Z of ASP .NET Core!**

**In this Article:**

* X is for XML + JSON Serialization
* Returning JsonResult and IActionResult
* Returning Complex Objects
* XML Serialization
* References

**X is for XML + JSON Serialization**

XML (eXtensible Markup Language) is a popular document format that has been used for a variety of applications over the years, including Microsoft Office documents, SOAP Web Services, application configuration and more. JSON (JavaScript Object Notation) was derived from JavaScript, but has also been used for storing data in both structured and unstructured formats, regardless of language used. In fact, ASP .NET Core applications switched from XML-based .config files to JSON-based .json settings files for application configuration.

[](https://wakeupandcode.com/wp-content/uploads/2019/06/XmlJsonSerialization.png)

This article will refer to the following sample code on GitHub, derived from the guidance provided in the [official documentation](https://docs.microsoft.com/en-us/aspnet/core/web-api/advanced/formatting) + [sample](https://github.com/aspnet/AspNetCore.Docs/tree/master/aspnetcore/web-api/advanced/formatting/sample):

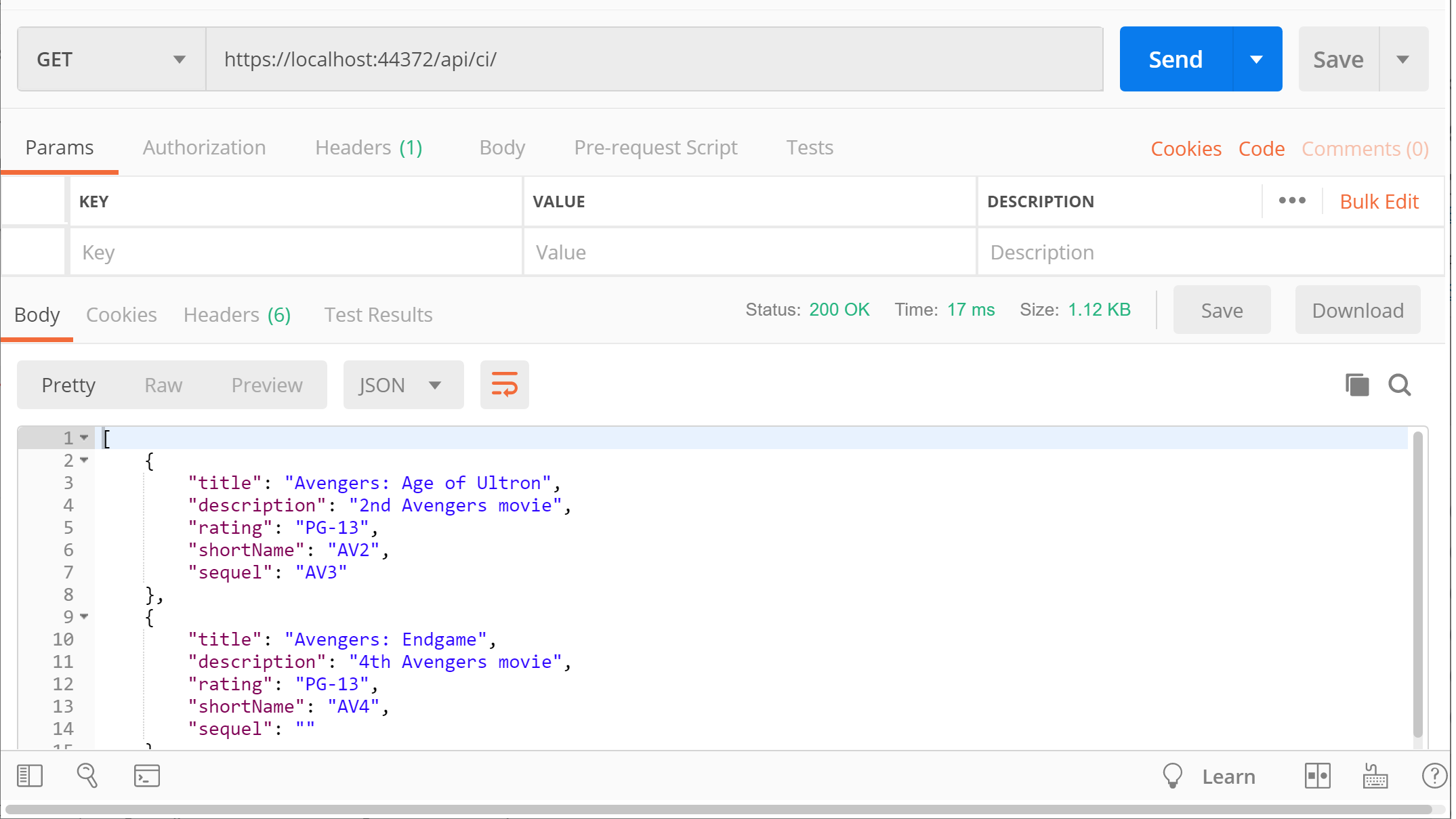
 XML + JSON Serialization : <https://github.com/shahedc/XmlJsonSerialization>

**Returning JsonResult and IActionResult**

Before we get into XML serialization, let’s start off with JSON Serialization first, and then we’ll get to XML. If you run the [Web API sample project](https://github.com/shahedc/XmlJsonSerialization) for this blog post, you’ll notice a [CIController.cs](https://github.com/shahedc/XmlJsonSerialization/blob/master/XmlJsonSerialization/Controllers/CIController.cs) file that represents a “Cinematic Item Controller” that expose API endpoints. These endpoints can serve up both JSON and XML results of Cinematic Items, i.e. movies, shows and shorts in a Cinematic Universe.

Run the application and navigate to the following endpoint in an API testing tool, e.g. [Postman](https://www.getpostman.com/):

* https://localhost:44372/api/ci/

[](https://wakeupandcode.com/wp-content/uploads/2019/06/Serialization-Get.png)

This triggers a GET request by calling the [CIController](https://github.com/shahedc/XmlJsonSerialization/blob/master/XmlJsonSerialization/Controllers/CIController.cs)‘s **Get**() method:

  // GET: api/ci  
 [HttpGet]  
 public JsonResult Get()  
 {  
 return Json(\_cinematicItemRepository.CinematicItems());  
 }

In this case, the **Json**() method returns a **JsonResult** object that serializes a list of Cinematic Items. For simplicity, the \_cinematicRepository object’s CinematicItems() method (in [CinematicItemRepository.cs](https://github.com/shahedc/XmlJsonSerialization/blob/master/XmlJsonSerialization/Infrastructure/CinematicItemRepository.cs)) returns a hard-coded list of **CinematicItem** objects. Its implementation here isn’t important, because you would typically retrieve such values from a persistent data store, preferably through some sort of service class.

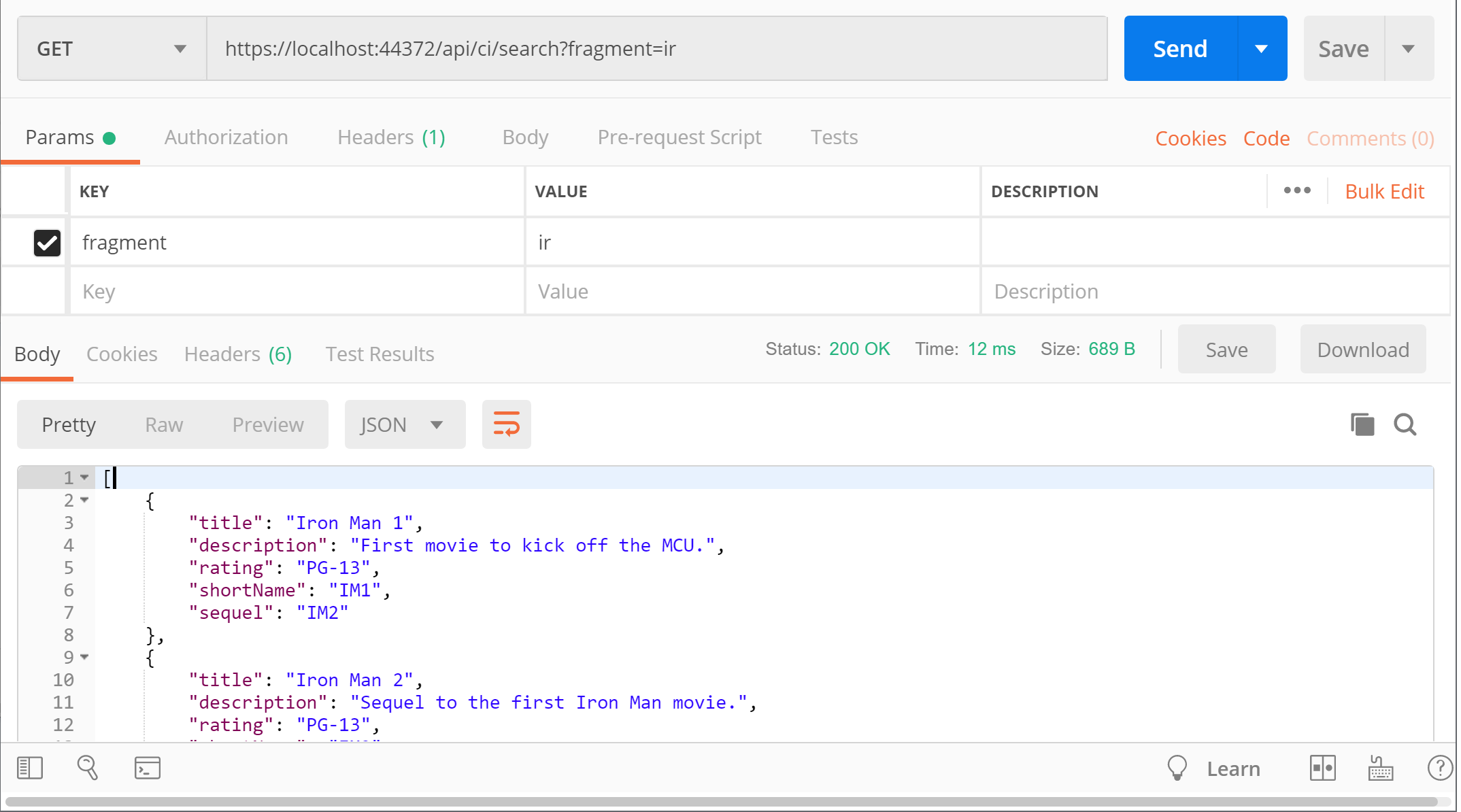
public List<CinematicItem> CinematicItems()  
{  
 return new List<CinematicItem>  
 {  
 new CinematicItem  
 {  
 Title = "Iron Man 1",  
 Description = "First movie to kick off the MCU.",  
 Rating = "PG-13",  
 ShortName = "IM1",  
 Sequel = "IM2"  
 },  
 ...   
 }  
}

The JSON result looks like the following, where a list of movies are returned:

[  
 {  
 "title": "Avengers: Age of Ultron",  
 "description": "2nd Avengers movie",  
 "rating": "PG-13",  
 "shortName": "AV2",  
 "sequel": "AV3"  
 },  
 {  
 "title": "Avengers: Endgame",  
 "description": "4th Avengers movie",  
 "rating": "PG-13",  
 "shortName": "AV4",  
 "sequel": ""  
 },  
 {  
 "title": "Avengers: Infinity War",  
 "description": "3rd Avengers movie",  
 "rating": "PG-13",  
 "shortName": "AV3",  
 "sequel": "AV4"  
 },  
 {  
 "title": "Iron Man 1",  
 "description": "First movie to kick off the MCU.",  
 "rating": "PG-13",  
 "shortName": "IM1",  
 "sequel": "IM2"  
 },  
 {  
 "title": "Iron Man 2",  
 "description": "Sequel to the first Iron Man movie.",  
 "rating": "PG-13",  
 "shortName": "IM2",  
 "sequel": "IM3"  
 },  
 {  
 "title": "Iron Man 3",  
 "description": "Wraps up the Iron Man trilogy.",  
 "rating": "PG-13",  
 "shortName": "IM3",  
 "sequel": ""  
 },  
 {  
 "title": "The Avengers",  
 "description": "End of MCU Phase 1",  
 "rating": "PG-13",  
 "shortName": "AV1",  
 "sequel": "AV2"  
 }  
]

Instead of specifically returning a **JsonResult**, you could also return a more generic **IActionResult**, which can still be interpreted as JSON. Run the application and navigate to the following endpoint, to include the action method “search” folllowed by a QueryString parameter “fragment” for a partial match.

* https://localhost:44372/api/ci/**search**?**fragment**=ir

[](https://wakeupandcode.com/wp-content/uploads/2019/06/Serialization-Get-Search.png)

This triggers a GET request by calling the [CIController](https://github.com/shahedc/XmlJsonSerialization/blob/master/XmlJsonSerialization/Controllers/CIController.cs)‘s **Search**() method, with its fragment parameter set to “ir” for a partial text search:

// GET: api/ci/search?fragment=ir  
[HttpGet("Search")]  
public IActionResult Search(string fragment)  
{  
 var result = \_cinematicItemRepository.GetByPartialName(fragment);  
 if (!result.Any())  
 {  
 return NotFound(fragment);  
 }  
 return Ok(result);  
}

In this case, the **GetByPartialName**() method returns a **List** of **CinematicItem**objects that are returned as JSON by default, with a HTTP 200 OK status. In case no results are found, the action method will return a 404 with the **NotFound**() method.

public List<CinematicItem> GetByPartialName(string titleFragment)  
{  
 return CinematicItems()  
 .Where(ci => ci.Title  
 .IndexOf(titleFragment, 0, StringComparison.CurrentCultureIgnoreCase) != -1)  
 .ToList();  
}

The JSON result looks like the following, where any movie title partially matches the string fragment provided:

[  
 {  
 "title": "Iron Man 1",  
 "description": "First movie to kick off the MCU.",  
 "rating": "PG-13",  
 "shortName": "IM1",  
 "sequel": "IM2"  
 },  
 {  
 "title": "Iron Man 2",  
 "description": "Sequel to the first Iron Man movie.",  
 "rating": "PG-13",  
 "shortName": "IM2",  
 "sequel": "IM3"  
 },  
 {  
 "title": "Iron Man 3",  
 "description": "Wraps up the Iron Man trilogy.",  
 "rating": "PG-13",  
 "shortName": "IM3",  
 "sequel": ""  
 }  
]

**Returning Complex Objects**

An overloaded version of the **Get**() method takes in a “**shortName**” string parameter to filter results by an alternate short name for each movie in the repository for the cinematic universe. Instead of returning a **JsonResult** or **IActionResult**, this one returns a complex object (**CinematicItem**) that contains properties that we’re interested in.

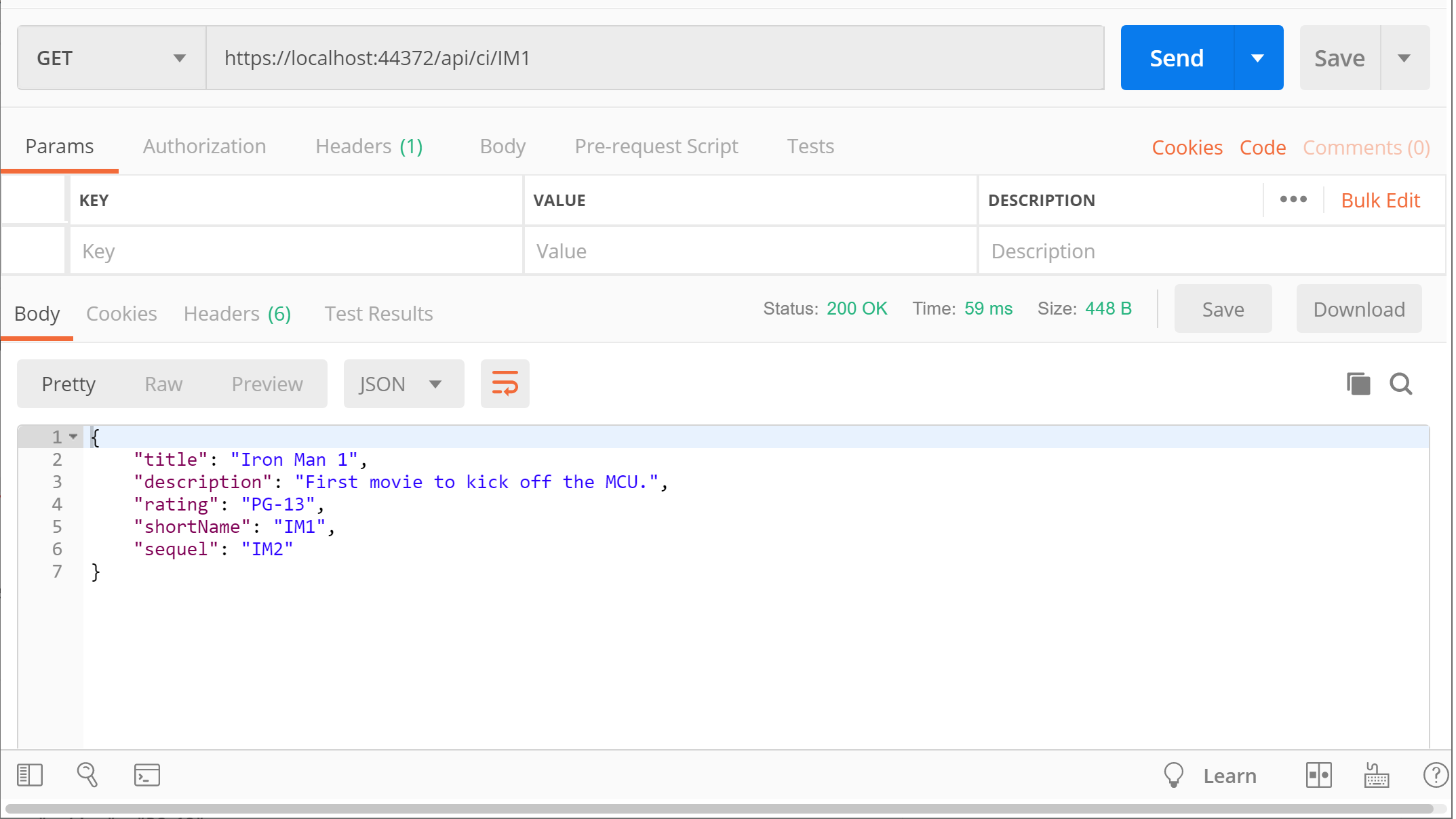
// GET api/ci/IM1  
[HttpGet("{shortName}")]  
public CinematicItem Get(string shortName)  
{  
 return \_cinematicItemRepository.GetByShortName(shortName);  
}

The **GetByShortName**() method in the [CinematicItemRepository.cs](https://github.com/shahedc/XmlJsonSerialization/blob/master/XmlJsonSerialization/Infrastructure/CinematicItemRepository.cs) class simply checks for a movie by the **shortName** parameter and returns the first match. Again, the implementation is not particularly important, but it illustrates how you can pass in parameters to get back JSON results.

public CinematicItem GetByShortName(string shortName)  
{  
 return CinematicItems().FirstOrDefault(ci => ci.ShortName == shortName);  
}

While the application is running, navigate to the following endpoint:

* https://localhost:44372/api/ci/IM1

[](https://wakeupandcode.com/wp-content/uploads/2019/06/Serialization-Get-ShortName.png)

This triggers another GET request by calling the [CIController](https://github.com/shahedc/XmlJsonSerialization/blob/master/XmlJsonSerialization/Controllers/CIController.cs)‘s overloaded **Get**() method, with the shortName parameter. When passing the short name “IM1”, this returns one item  “Iron Man 1”, as shown below:

{  
 "title": "Iron Man 1",  
 "description": "First movie to kick off the MCU.",  
 "rating": "PG-13",  
 "shortName": "IM1",  
 "sequel": "IM2"  
}

Another example with a complex result takes in a parameter via QueryString and checks for an exact match with a specific property. In this case the **Related**() action method calls the repository’s **GetBySequel**() method to find a specific movie by its sequel’s short name.

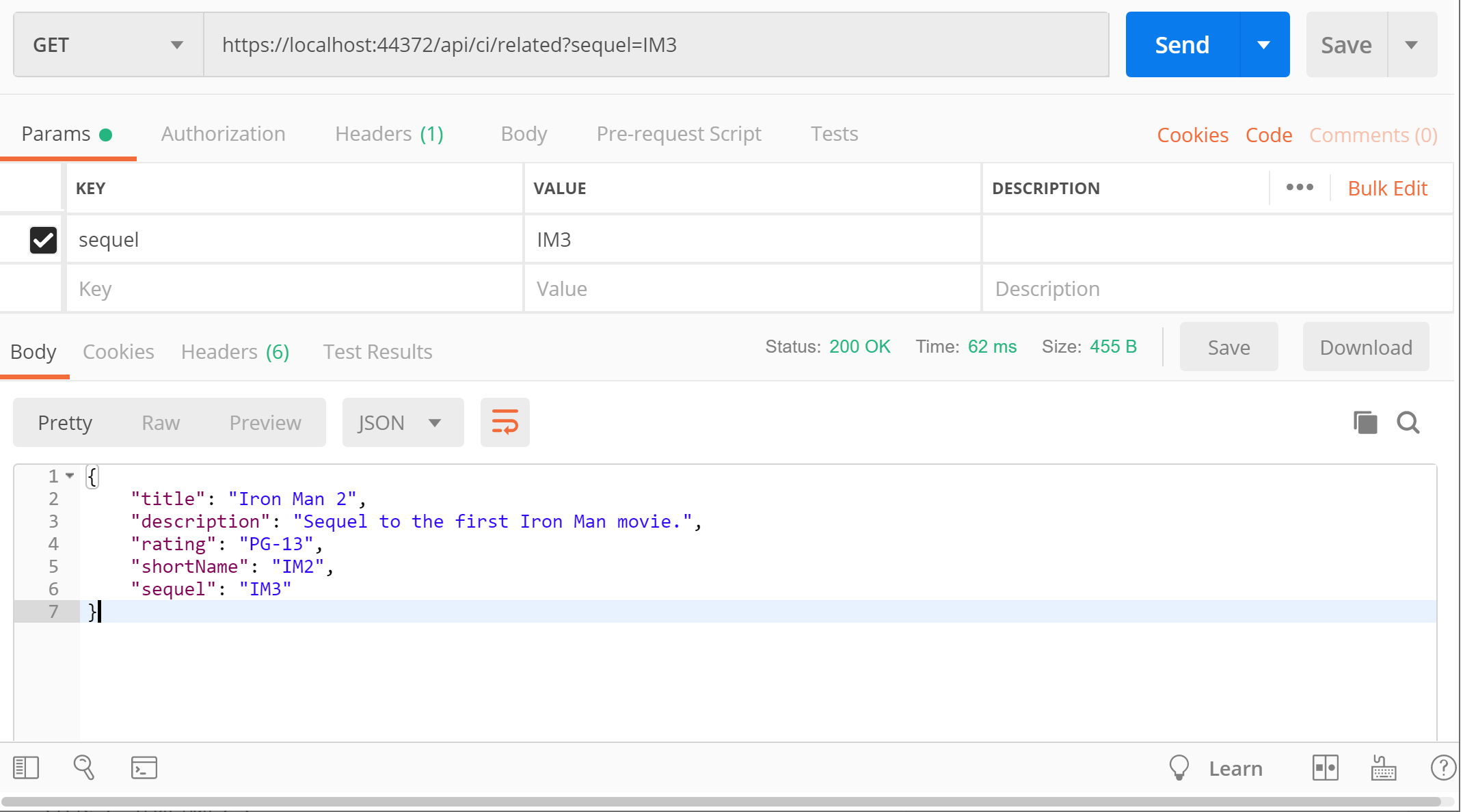
// GET: api/ci/related?sequel=IM2  
[HttpGet("Related")]  
public CinematicItem Related(string sequel)  
{  
 return \_cinematicItemRepository.GetBySequel(sequel);  
}

The **GetBySequel**() method in the [CinematicItemRepository.cs](https://github.com/shahedc/XmlJsonSerialization/blob/master/XmlJsonSerialization/Infrastructure/CinematicItemRepository.cs) class  checks for a movie’s sequel by the **shortName** parameter and returns the first match.

public CinematicItem GetBySequel(string sequelShortName)  
{  
 return CinematicItems().FirstOrDefault(ci => ci.Sequel == sequelShortName);  
}

While the application is running, navigate to the following endpoint:

* https://localhost:44372/api/ci/**related**?**sequel**=IM3

[](https://wakeupandcode.com/wp-content/uploads/2019/06/Serialization-Get-Sequel.png)

This triggers a GET request by calling the [CIController](https://github.com/shahedc/XmlJsonSerialization/blob/master/XmlJsonSerialization/Controllers/CIController.cs)‘s **Related**() method, with the **sequel** parameter. When passing the sequel’s short name “IM3”, this returns one item “Iron Man 2”, as shown below:

{  
 "title": "Iron Man 2",  
 "description": "Sequel to the first Iron Man movie.",  
 "rating": "PG-13",  
 "shortName": "IM2",  
 "sequel": "IM3"  
}

As you can see, the result is in JSON format for the returned object.

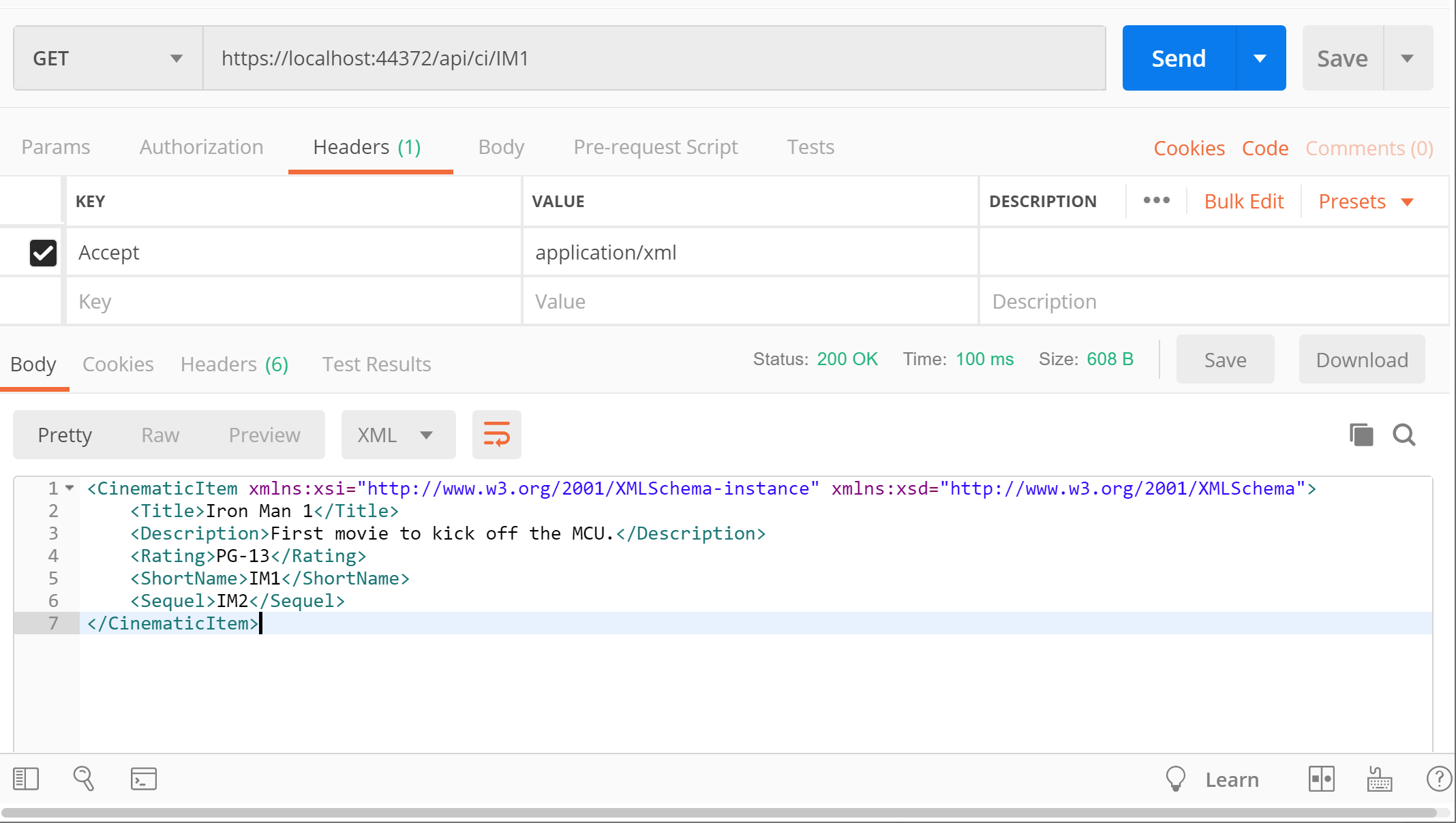
**XML Serialization**

*Wait a minute… with all these JSON results, when will we get to XML serialization?* Not to worry, there are multiple ways to get XML results while reusing the above code. First, add the NuGet package “[Microsoft.AspNetCore.Mvc.Formatters.Xml](https://www.nuget.org/packages/Microsoft.AspNetCore.Mvc.Formatters.Xml/)” to your project and then update your Startup.cs file’s **ConfigureServices**() to include a call to services.AddMvc.**AddXmlSeralizerFormatters**():

public void ConfigureServices(IServiceCollection services)  
{  
 ...  
 services.AddMvc()  
 .AddXmlSerializerFormatters();  
 ...  
}

Set the request’s Accept header value to “application/xml” *before* requesting the endpoint, then run the application and navigate to the following endpoint once again:

* https://localhost:44372/api/ci/IM1

[](https://wakeupandcode.com/wp-content/uploads/2019/06/Serialization-Get-XML.png)

This should provide the following XML results:

<CinematicItem xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">  
 <Title>Iron Man 1</Title>  
 <Description>First movie to kick off the MCU.</Description>  
 <Rating>PG-13</Rating>  
 <ShortName>IM1</ShortName>  
 <Sequel>IM2</Sequel>  
</CinematicItem>

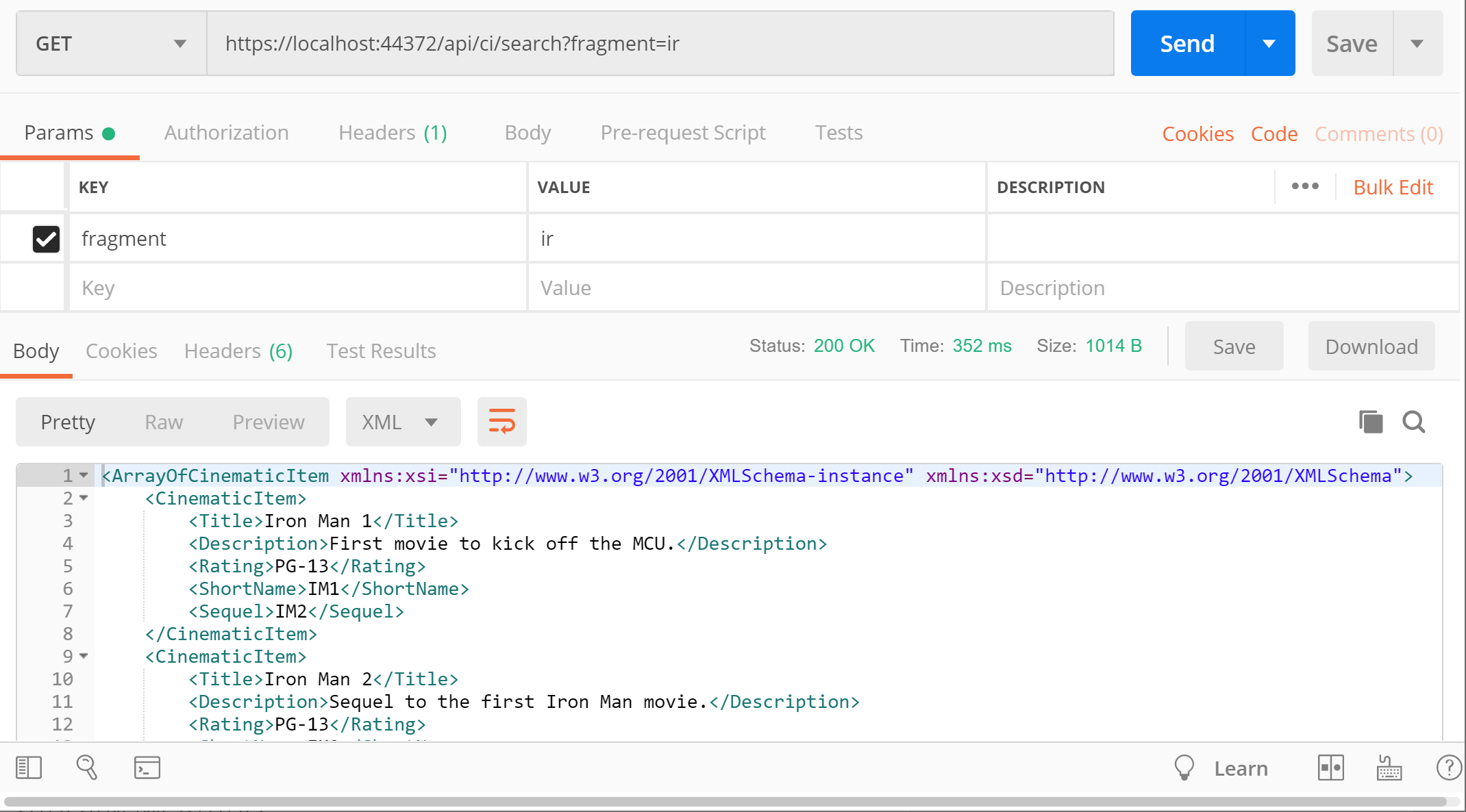
Since the action method returns a complex object, the result can easily be switched to XML simply by changing the Accept header value. In order to return XML using an IActionResult method, you should also use the [**Produces**] attribute, which can be set to “**application/xml**” at the Controller level

[Produces("application/xml")]  
[Route("api/[controller]")]  
[ApiController]  
public class CIController : Controller  
{  
 ...  
}

Then revisit the following endpoint, calling the search action method with the fragment parameter set to “ir”:

* https://localhost:44372/api/ci/**search**?**fragment**=ir

At this point, it is no longer necessary to set the Accept header to “application/xml” during the request, since the [**Produces**] attribute is given priority over it.

[](https://wakeupandcode.com/wp-content/uploads/2019/06/Serialization-Get-Search-XML.png)

This should produces the following result , with an array of **CinematicItem** objects in XML:

<ArrayOfCinematicItem xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">  
 <CinematicItem>  
 <Title>Iron Man 1</Title>  
 <Description>First movie to kick off the MCU.</Description>  
 <Rating>PG-13</Rating>  
 <ShortName>IM1</ShortName>  
 <Sequel>IM2</Sequel>  
 </CinematicItem>  
 <CinematicItem>  
 <Title>Iron Man 2</Title>  
 <Description>Sequel to the first Iron Man movie.</Description>  
 <Rating>PG-13</Rating>  
 <ShortName>IM2</ShortName>  
 <Sequel>IM3</Sequel>  
 </CinematicItem>  
 <CinematicItem>  
 <Title>Iron Man 3</Title>  
 <Description>Wraps up the Iron Man trilogy.</Description>  
 <Rating>PG-13</Rating>  
 <ShortName>IM3</ShortName>  
 <Sequel />  
 </CinematicItem>  
</ArrayOfCinematicItem>

As for the first **Get**() method returning **JsonResult**, you can’t override it with the [**Produces**] attribute or the **Accept** header value to change the result to XML format.

To recap, the order of precedence is as follows:

1. public **JsonResult** Get()
2. [**Produces**(“application/…”)]
3. **Accept**: “application/…”

**References**

* JSON and XML Serialization in ASP.NET Web API – ASP.NET 4.x | Microsoft Docs: <https://docs.microsoft.com/en-us/aspnet/web-api/overview/formats-and-model-binding/json-and-xml-serialization#xml-media-type-formatter>
* How to format response data as XML or JSON, based on the request URL in ASP.NET Core: <https://andrewlock.net/formatting-response-data-as-xml-or-json-based-on-the-url-in-asp-net-core/>
* Format response data in ASP.NET Core Web API | Microsoft Docs: <https://docs.microsoft.com/en-us/aspnet/core/web-api/advanced/formatting?view=aspnetcore-2.2>

This entry was posted in [ASP.NET](https://wakeupandcode.com/category/asp-net/), [Visual Studio](https://wakeupandcode.com/category/visual-studio/), [Web Development](https://wakeupandcode.com/category/web-development/) and tagged [.net](https://wakeupandcode.com/tag/net/), [.NET Core](https://wakeupandcode.com/tag/net-core/), [ASP.NET](https://wakeupandcode.com/tag/asp-net/), [ASP.NET Core](https://wakeupandcode.com/tag/asp-net-core/), [JSON](https://wakeupandcode.com/tag/json/), [Serialization](https://wakeupandcode.com/tag/serialization/), [XML](https://wakeupandcode.com/tag/xml/) on [June 17, 2019](https://wakeupandcode.com/xml-json-serialization-in-asp-net-core/) by [Shahed C](https://wakeupandcode.com/author/admin/).

**Post navigation**

[← Worker Service in ASP .NET Core](https://wakeupandcode.com/worker-service-in-asp-net-core/) [YAML-defined CI/CD for ASP .NET Core →](https://wakeupandcode.com/yaml-defined-cicd-for-asp-net-core/)

**2 thoughts on “XML + JSON Serialization in ASP .NET Core”**

1. Pingback: [The Morning Brew - Chris Alcock » The Morning Brew #2768](http://blog.cwa.me.uk/2019/06/18/the-morning-brew-2768/)
2. Pingback: [Dew Drop – June 19, 2019 (#2982) | Morning Dew](https://www.alvinashcraft.com/2019/06/19/dew-drop-june-19-2019-2982/)

**Leave a Reply**

This site uses Akismet to reduce spam. [Learn how your comment data is processed](https://akismet.com/privacy/).