

REST methoden

In dit hoofdstuk bespreken we de meest gebruikte REST methoden, namelijk POST, DELETE, PUT en PATCH.

POST

De methode POST wordt gebruikt om een element toe te voegen. De code die we daarvoor gebruiken ziet er als volgt uit :

```
[HttpPost]
0 references
public ActionResult<Country> Post([FromBody] Country country)
{
    repo.AddCountry(country);
    return CreatedAtAction(nameof(Get), new { id = country.Id }, country);
}
```

Via de annotatie [HttpPost] duiden we aan dat de volgende methode in onze controller zal gelinkt worden aan het POST request. Als parameter verwachten we een Country object dat uit de body van de request zal worden gehaald. Als resultaat geven we een ActionResult<Country> dat we aanmaken met de methode CreatedAtAction. Deze laatste functie maakt een antwoord (response) met een statuscode 201 (Created), een Location header met een URL die verwijst naar de zojuist aangemaakte bron en een body die het nieuw aangemaakte element bevat. De eerste twee parameters die worden meegegeven (nameof(Get) en new {id=country.Id}) worden gebruikt om de URL aan te maken en de laatste parameter (country) dient om de body van de response te maken.

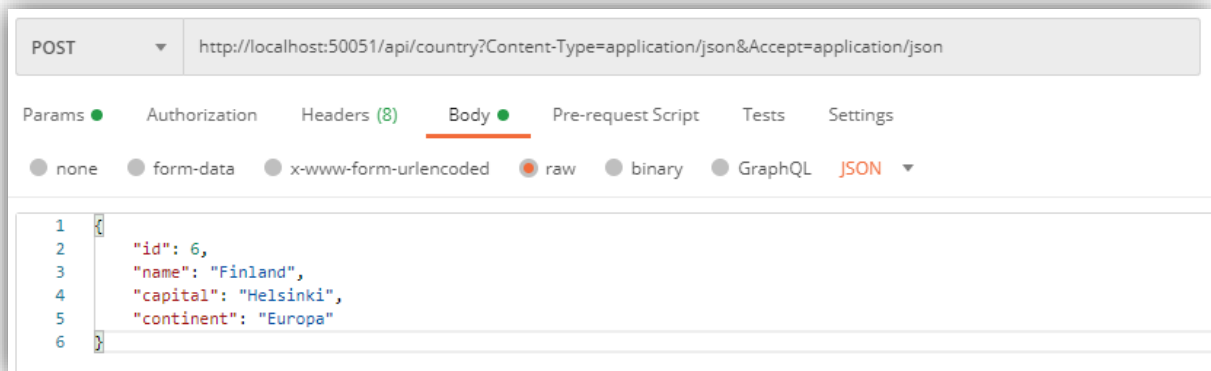
Opmerking : nameof(Get) verwijst naar de Get-methode uit onze controller.

De request die we sturen bevat de volgende parameters : Content-Type en Accept, beiden met de waarde application/json waarmee we aangeven dat de info in de body van de request in json zal zijn en dat we een antwoord verwachten eveneens in json.

Params ●	Authorization	Headers (8)	Body ●	Pre-request Script	Tests	Settings
Query Params						
	KEY	VALUE				
<input checked="" type="checkbox"/>	Content-Type	application/json				
<input checked="" type="checkbox"/>	Accept	application/json				

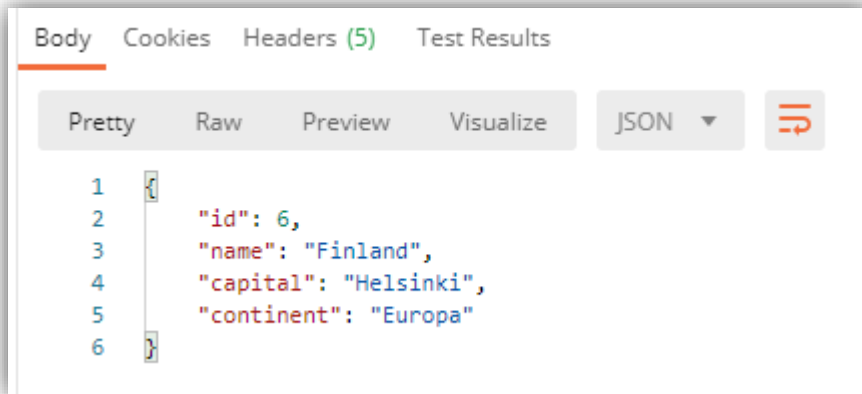
Voor het request zelf gebruiken we de methode POST en verwijst de URL naar de route van de controller voorafgegaan door de server en het protocol (http). In de Body van het request geven we het object/element mee in json formaat.

```
[Route("api/[controller]")]
[ApiController]
1 reference
public class CountryController : ControllerBase
{
```



Als resultaat krijgen we een antwoord met statuscode 201 Created, met daarbij een location header die de URL naar het zopas aangemaakte element bevat. Verder zien we nog dat de Body van de response ook het nieuw aangemaakte element bevat.

Body		Cookies	Headers (5)	Test Results	Status: 201 Created	
		KEY			VALUE	
		Date			Fri, 21 Aug 2020 07:13:24 GMT	
		Content-Type			application/json; charset=utf-8	
		Server			Kestrel	
		Transfer-Encoding			chunked	
		Location			http://localhost:50051/api/Country/6	



DELETE

De tweede methode die we bespreken is DELETE, die dient om elementen te verwijderen. In onze controller maken we bijvoorbeeld de volgende methode aan :

```
[HttpDelete("{id}")]
0 references
public IActionResult Delete(int id)
{
    if (!repo.ExistsCountry(id))
    {
        return NotFound();
    }

    repo.RemoveCountry(repo.GetCountry(id));

    return NoContent();
}
```

De annotatie [HttpDelete] bevat nu een parameter {id} die aangeeft welk element dat moet worden verwijderd. Op deze manier wordt ook de link gelegd tussen de id die we in de URL gaan meegeven en de parameter id in de methode Delete. Eerst controleren we of het te verwijderen element bestaat (we voegen daarom een extra functie ExistsCountry toe aan onze repository), indien het element niet bestaat geven we als antwoord een ActionResult met statuscode 404 Not Found terug.

Bestaat het element wel, dan verwijderen we het via de RemoveCountry functie en geven als antwoord een NoContent met statuscode 204 No Content.

```
public bool ExistsCountry(int id)
{
    if (data.ContainsKey(id)) return true;
    else return false;
}
```

We wensen het element met id = 1 te verwijderen. Om te controleren of het element wel degelijk wordt verwijderd voeren we eerst een GET request uit om te zien of het element initieel wel bestaat.

GET http://localhost:50051/api/country/1

Params Authorization Headers (6) Body Pre-request Script Tests Settings

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (4) Test Results Status: 200 OK Time: 111 ms

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 1,
3   "name": "België",
4   "capital": "Brussel",
5   "continent": "Europa"
6 }
```

Daarna voeren we het DELETE request uit, met de URL die verwijst naar het betreffende element. Het resultaat is een response met statuscode 204 (No Content) en een lege Body.

DELETE http://localhost:50051/api/country/1

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

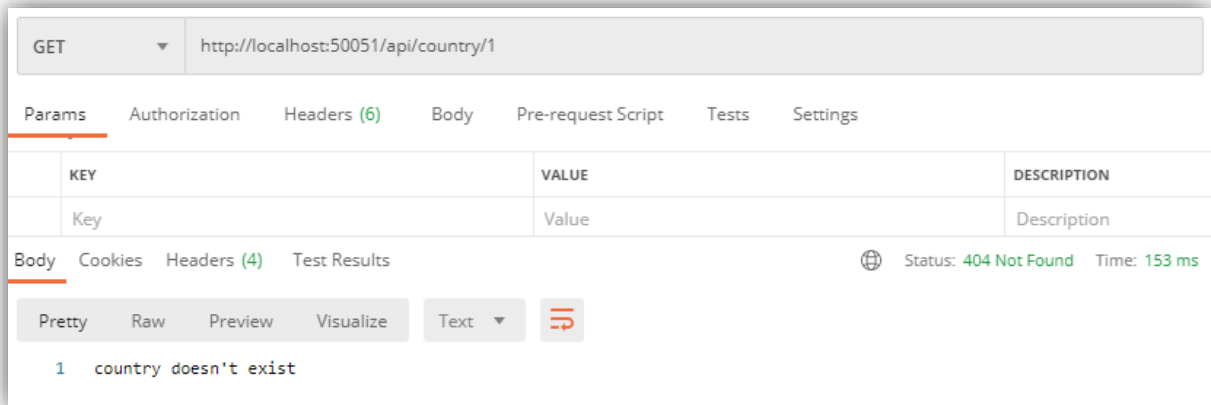
KEY	VALUE	DESCRIPTION
-----	-------	-------------

Body Cookies Headers (2) Test Results Status: 204 No Content Time: 116 ms

Pretty Raw Preview Visualize Text

```
1
```

Vervolgens controleren we of het element wel degelijk is verwijderd, door opnieuw het GET request met id=1 uit te voeren. Het antwoord is nu een 404 Not Found waaruit we kunnen afleiden dat het element wel degelijk is verwijderd.



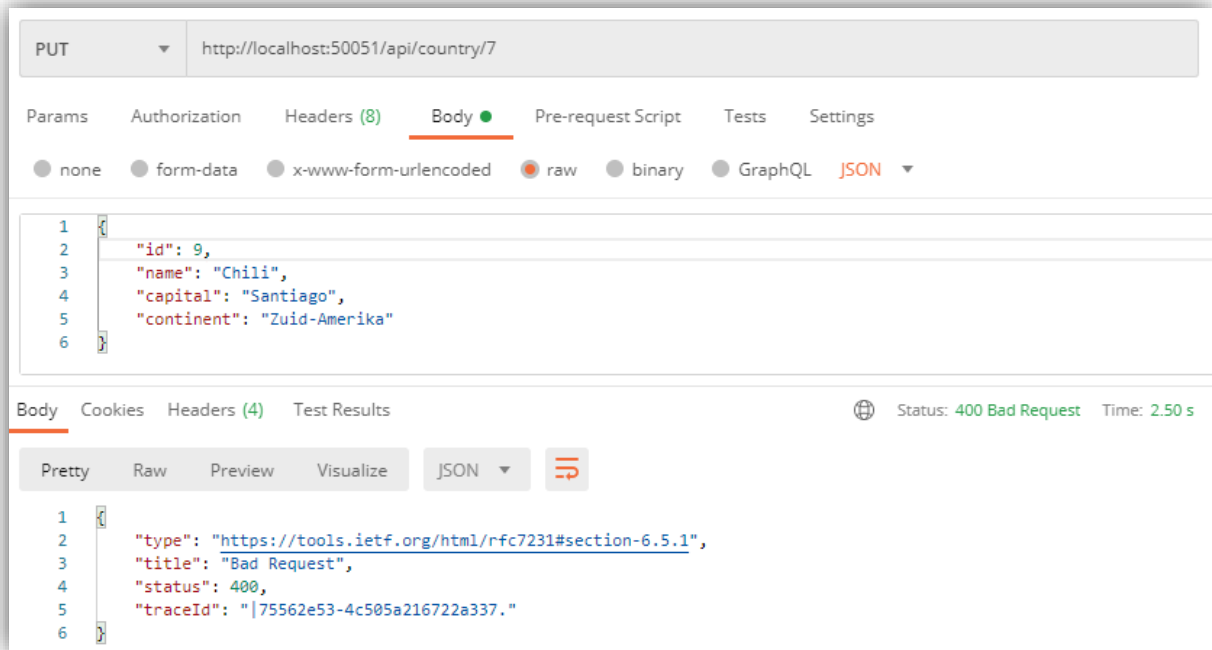
PUT

Met PUT kunnen we zowel een nieuw element aanmaken als een element updaten. Deze update is dan wel een full update, dat wil zeggen dat alle velden worden vervangen en indien geen info aanwezig worden default values gebruikt. Vanuit het PUT request verwachten we dus zowel een id (in de URL) als de info voor het element (in de Body).

De methode kan er dan als volgt uit zien :

```
[HttpPut("{id}")]
0 references
public IActionResult Put(int id, [FromBody] Country country)
{
    if (country == null || country.Id != id)
    {
        return BadRequest();
    }
    if (!repo.ExistsCountry(id))
    {
        repo.AddCountry(country);
        return CreatedAtAction(nameof(Get), new { id = country.Id }, country);
    }
    var countryDB = repo.GetCountry(id);
    repo.UpdateCountry(country);
    return new NoContentResult();
}
```

We controleren eerst of het meegegeven element niet leeg is en of de Id in het element wel overeenkomt met de Id uit de URL. Indien dit niet het geval is geven we een Bad Request ActionResult terug. In het voorbeeld geven we een element mee met id = 9, terwijl we in de URL verwijzen naar element met id = 7.



lin het tweede voorbeeld verwijst de id naar een nog niet bestaand element en zal er een nieuw element worden toegevoegd. Het resultaat is nu analoog als zouden we een POST request hebben verstuurd. Het antwoord bestaat nu uit een Body met het nieuwe element, een statuscode 201 Created en een Location header met de link naar het element.

PUT ▼ http://localhost:50051/api/country/7

Params Authorization Headers (8) **Body** ● Pre-request Script Tests Settings

● none ● form-data ● x-www-form-urlencoded ● **raw** ● binary ● GraphQL **JSON** ▼

```
1 {
2   "id": 7,
3   "name": "Chili",
4   "capital": "Santiago",
5   "continent": "Zuid-Amerika"
6 }
```

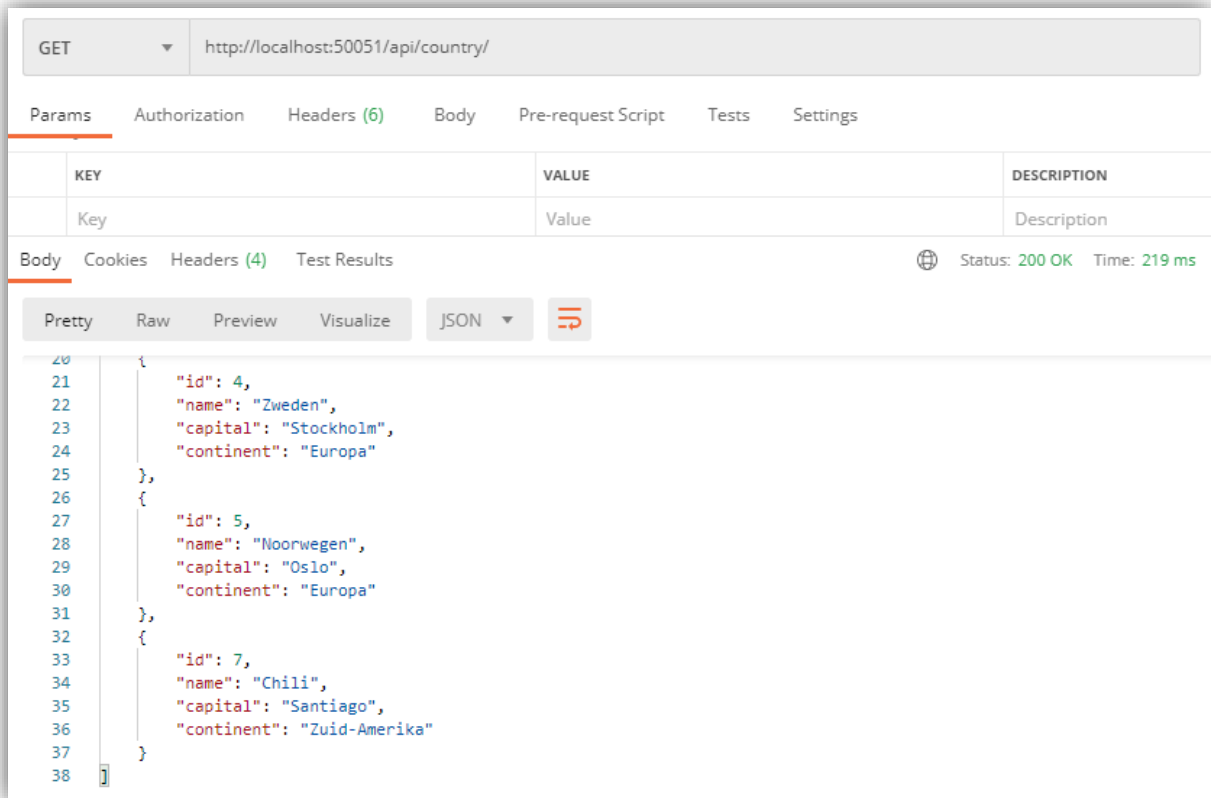
Body Cookies Headers (5) Test Results 🌐 Status: 201 Created Time: 963 ms

Pretty Raw Preview Visualize JSON ▼ ↺

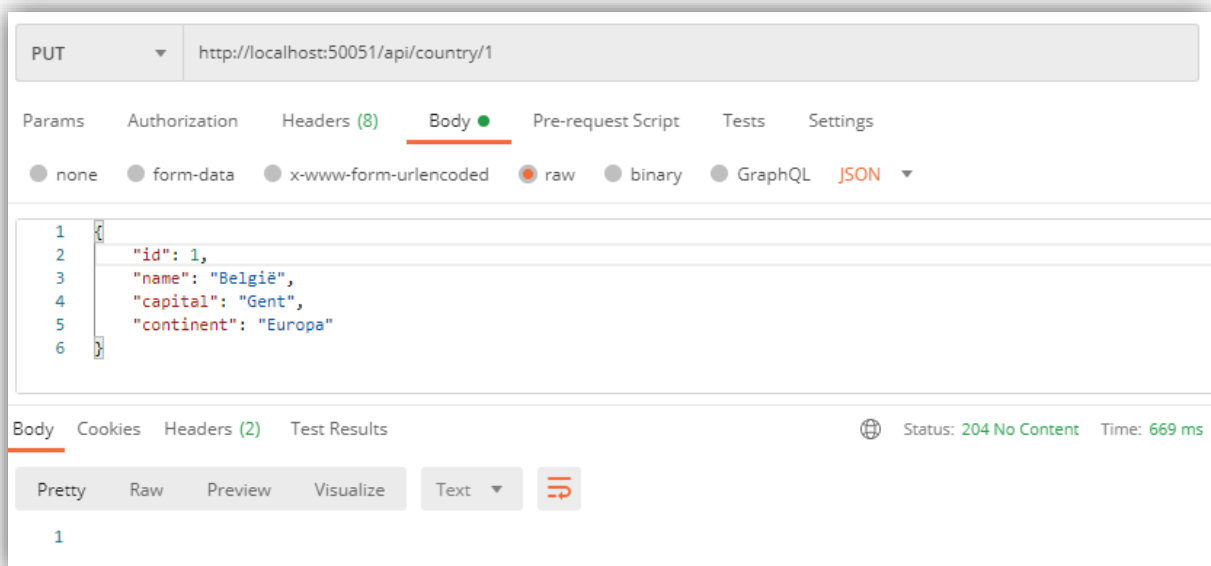
```
1 {
2   "id": 7,
3   "name": "Chili",
4   "capital": "Santiago",
5   "continent": "Zuid-Amerika"
6 }
```

Body		Cookies	Headers (5)	Test Results	🌐 Status: 201 Created Time	
				KEY	VALUE	
				Date ①	Wed, 26 Aug 2020 11:38:35 GMT	
				Content-Type ①	application/json; charset=utf-8	
				Server ①	Kestrel	
				Content-Length ①	71	
				Location ①	http://localhost:50051/api/Country/7	

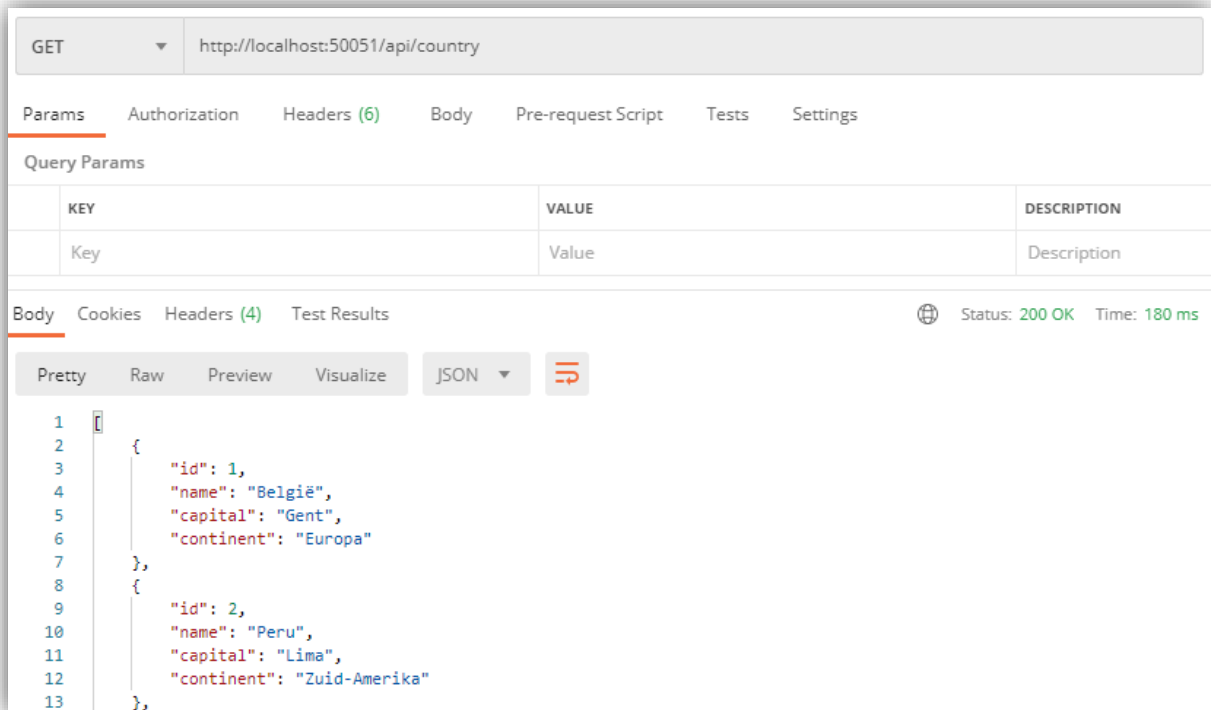
We controleren met het GET request of het element wel degelijk is toegevoegd.



In dit derde voorbeeld voeren we een klassieke update uit. Het element wordt opgehaald uit de repository, waarna de update wordt uitgevoerd. We geven als antwoord een ActionResult met statuscode 204 No Content.



Via de GET methode controleren we nog of de update effectief heeft plaatsgevonden.



PATCH

De laatste methode die we hier bespreken is PATCH. Deze methode dient om gedeeltelijke updates door te voeren. We geven nu niet het volledige element mee, maar enkel de aanpassingen/acties die er moeten op gebeuren. Een overzicht van de mogelijke acties vinden we weer in de volgende figuur.

Operation	Notes
add	Add a property or array element. For existing property: set value.
remove	Remove a property or array element.
replace	Same as <code>remove</code> followed by <code>add</code> at same location.
move	Same as <code>remove</code> from source followed by <code>add</code> to destination using value from source.
copy	Same as <code>add</code> to destination using value from source.
test	Return success status code if value at <code>path</code> = provided <code>value</code> .

De Patch methode zelf kunnen we als volgt implementeren. Als antwoord geven we een `ActionResult<Country>` zodat het geupdate element wordt teruggegeven. Als input verwachten we een Id via de URL en de uit te voeren acties als een `JsonPatchDocument`.



Indien het te updaten element niet kan worden gevonden bevat het antwoord een Not Found code 404. In het andere geval proberen we het opgevraagde element te updaten met de info uit het patch

document, dit kunnen we doen door de ApplyTo methode op te roepen van het JsonPatchDocument object. Lukt dit niet dan geven we een BadRequest terug als antwoord.



```
[HttpPatch("{id}")]
0 references
public ActionResult<Country> Patch(int id,[FromBody]JsonPatchDocument<Country> countryPatch)
{
    var countryDB = repo.GetCountry(id);
    if (countryDB == null)
    {
        return NotFound();
    }
    try
    {
        countryPatch.ApplyTo(countryDB);
        return countryDB;
    }
    catch (Exception ex)
    {
        return BadRequest();
    }
}
```

De PATCH methode maakt dus gebruik van een object JsonPatchDocument en om hiervan gebruik te maken moeten we eerst de volgende NuGet

Operation	Notes
add	Add a property or array element. For existing property: set value.
remove	Remove a property or array element.
replace	Same as remove followed by add at same location.
move	Same as remove from source followed by add to destination using value from source.
copy	Same as add to destination using value from source.
test	Return success status code if value at path = provided value.

**Microsoft.AspNetCore.JsonPatch**  by Microsoft, **131M** downloads v3.1.7

ASP.NET Core support for JSON PATCH.

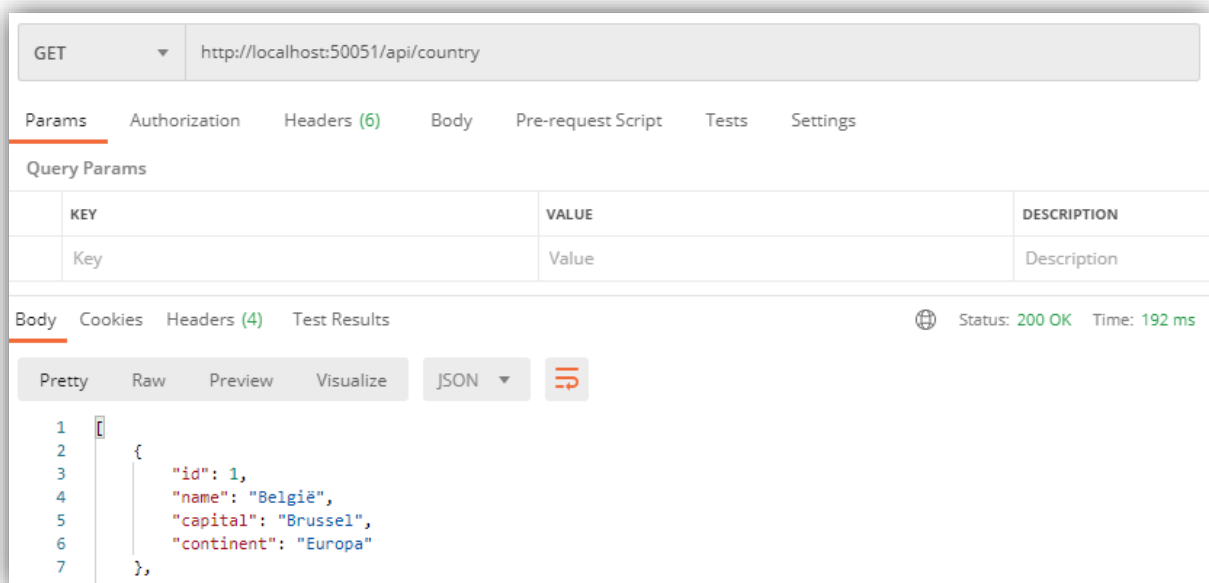
**Microsoft.AspNetCore.Mvc.NewtonsoftJson**  by Microsoft, **21,4M** downloads v3.1.7

ASP.NET Core MVC features that use Newtonsoft.Json. Includes input and output formatters for JSON and JSON PATCH.

Daarna moeten we de service nog configureren door de extension method `AddNewtonsoftJson` op te roepen en dit doen we in de `ConfigureServices` methode (Startup klasse).

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers(setup => {
        setup.ReturnHttpNotAcceptable = true;
    }).AddNewtonsoftJson().AddXmlDataContractSerializerFormatters();
    services.AddSingleton<ICountryRepository, CountryRepository>();
}
```

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers(setup => {
        setup.ReturnHttpNotAcceptable = true;
    }).AddNewtonsoftJson().AddXmlDataContractSerializerFormatters();
    services.AddSingleton<ICountryRepository, CountryRepository>();
}
```

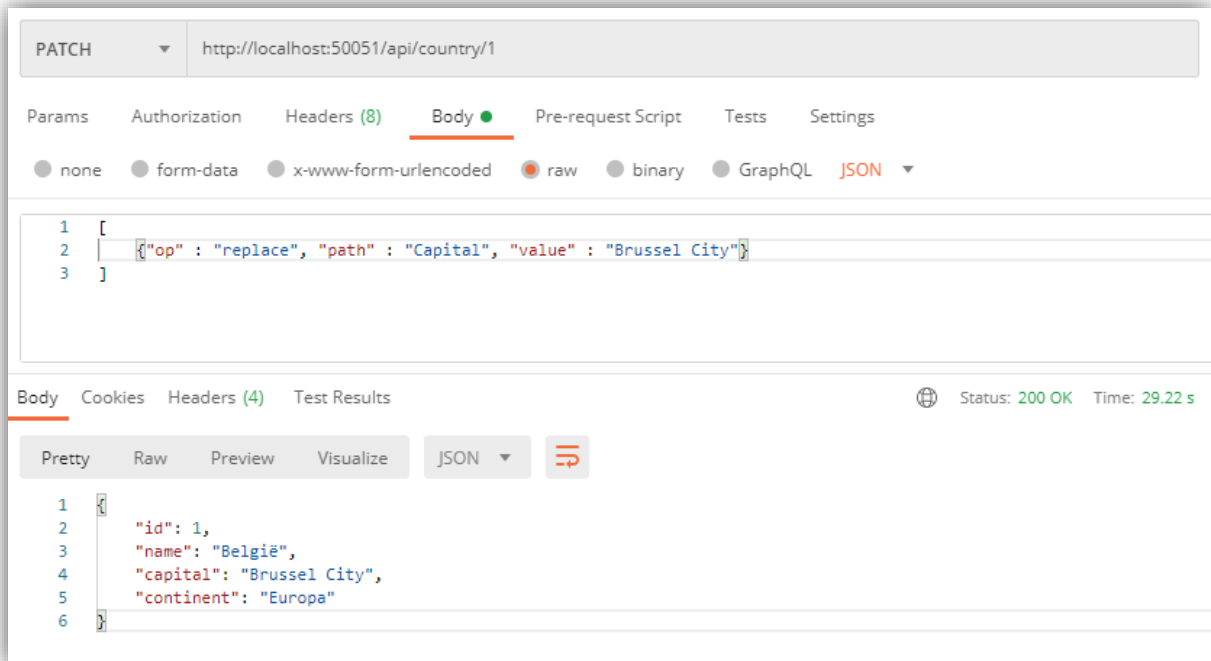


The screenshot shows a web browser interface for a REST client. The top bar indicates a GET request to `http://localhost:50051/api/country`. Below the bar, there are tabs for Params, Authorization, Headers (6), Body, Pre-request Script, Tests, and Settings. The Params tab is active, showing a table with columns KEY, VALUE, and DESCRIPTION. The table has one row with KEY 'Key', VALUE 'Value', and DESCRIPTION 'Description'. Below the table, there are tabs for Body, Cookies, Headers (4), and Test Results. The Body tab is active, showing a JSON response in the Pretty view. The JSON response is:

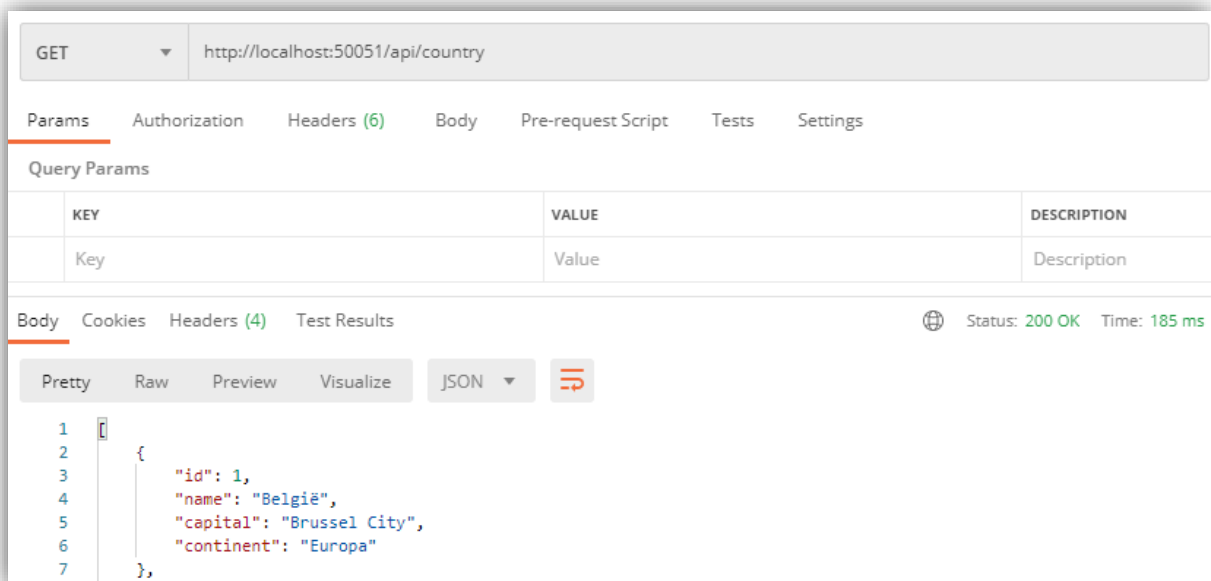
```
{
  "id": 1,
  "name": "België",
  "capital": "Brussel",
  "continent": "Europa"
}
```

. The status bar at the bottom right shows a status of 200 OK and a time of 192 ms.

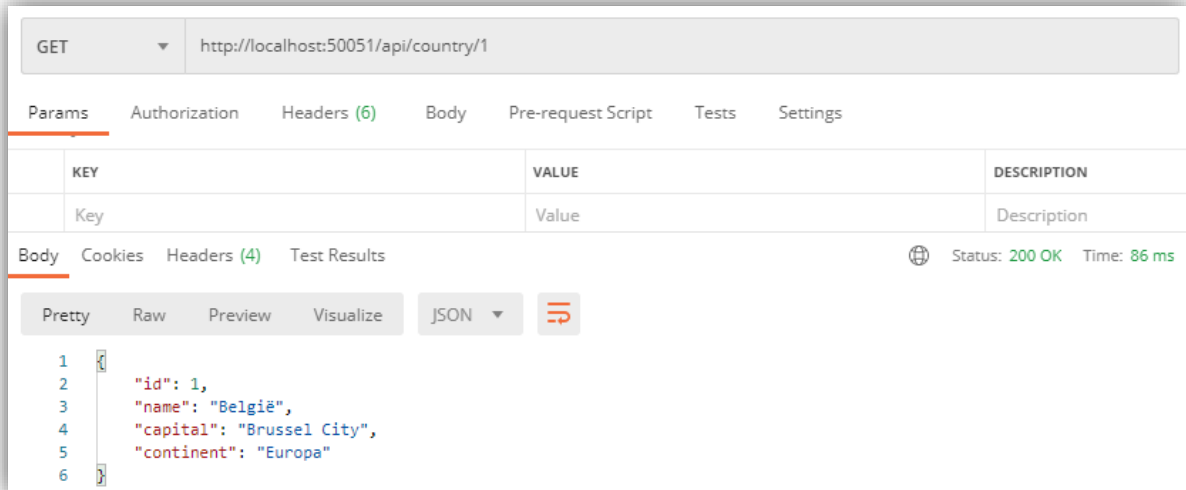
Daarna voeren we het PATCH request uit. In de URL verwijzen we naar het te updaten element en in de body van het request beschrijven we wat er moet gebeuren. In dit geval geven we aan dat de operatie ("op") een 'replace' operatie is, dat het te updaten attribuut ("path") de property 'Capital' is en dat de nieuwe waarde ("value") 'Brussel City' is.



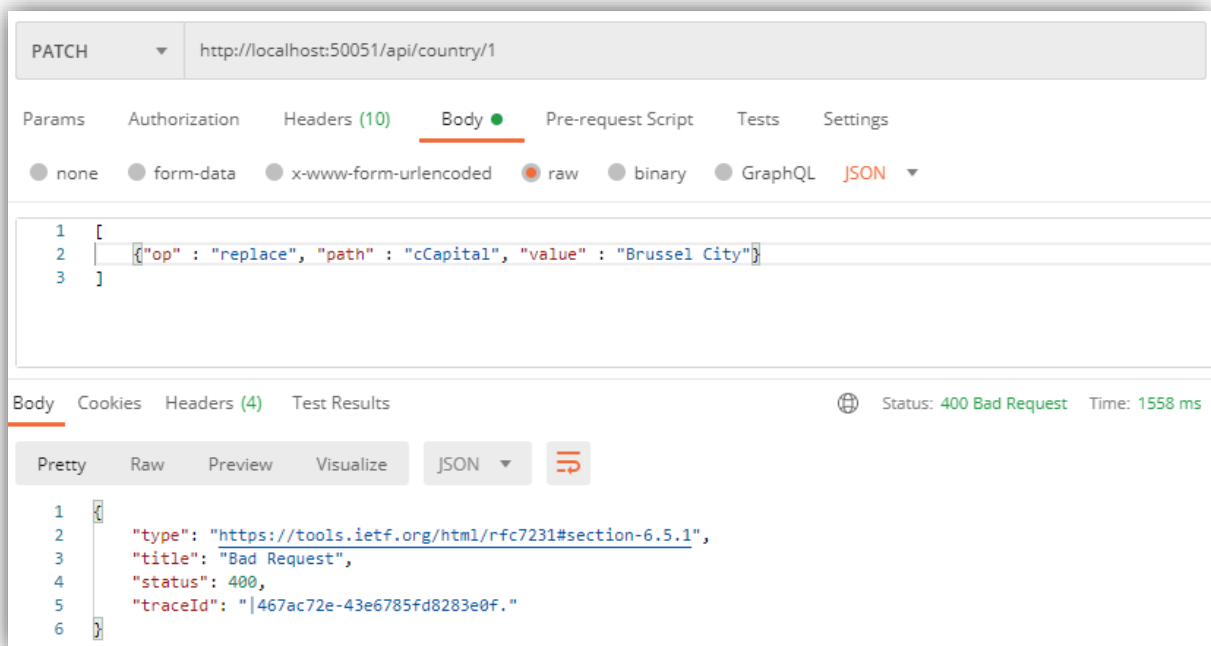
Het antwoord dat we terug krijgen is het geupdate element (in de Body) en de status code 200 Ok.



We controleren nog even met de GET methode of de update is gelukt.



Indien het PATCH request fouten bevat, zoals in het volgende voorbeeld (spelfout in path) dan krijgen we als antwoord een 400 Bad Request.



Referenties

<https://dotnetcoretutorials.com/2017/11/29/json-patch-asp-net-core/>

<https://docs.microsoft.com/en-us/aspnet/core/web-api/jsonpatch?view=aspnetcore-3.1>

<https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.controllerbase.createdataaction?view=aspnetcore-3.1>

<https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/nameof>

<https://restfulapi.net/rest-put-vs-post/>

