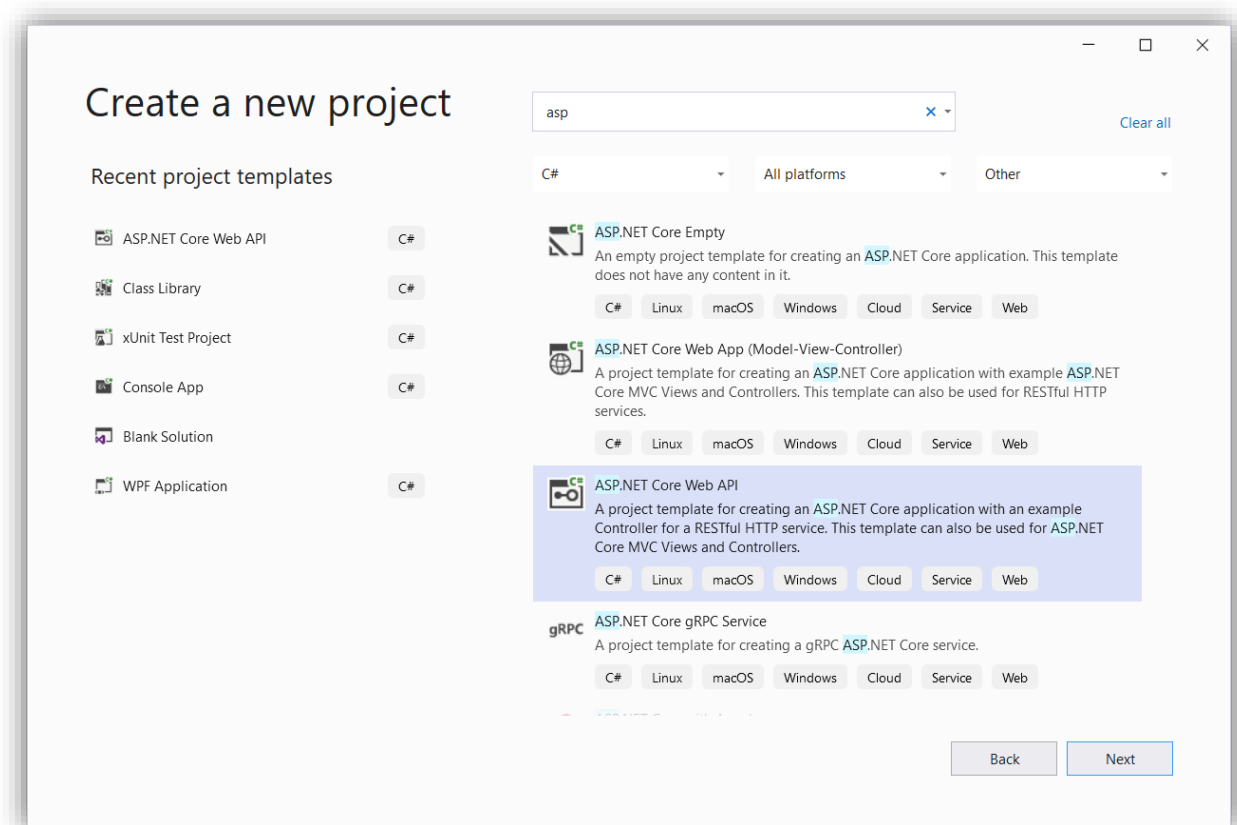


GET started

In dit hoofdstuk tonen we hoe een service kan worden opgezet en implementeren we enkel de GET methode. We bespreken hier nog niet de verschillende details en settings, de bedoeling is hier om een eenvoudig GET request te bekijken met de verschillende mogelijke antwoorden (responses).

Opzetten Visual Studio project

We starten een nieuw project en kiezen voor een ASP.NET Core Web Application. Aangezien we enkel een REST-service willen bouwen selecteren we daarna de optie API.



—□×

Configure your new project

ASP.NET Core Web API C# Linux macOS Windows Cloud Service Web

Project name

CountryService

Location

C:\VisualStudioProjects

...

Solution name ⓘ

CountryService

☒ Place solution and project in the same directory

Back

Next

—□×

Additional information

ASP.NET Core Web API C# Linux macOS Windows API Cloud Service Web Web API

Framework ⓘ

.NET 8.0 (Long Term Support)

Authentication type ⓘ

None

☐ Configure for HTTPS ⓘ

☐ Enable Docker ⓘ

Docker OS ⓘ

Linux

☒ Enable OpenAPI support ⓘ

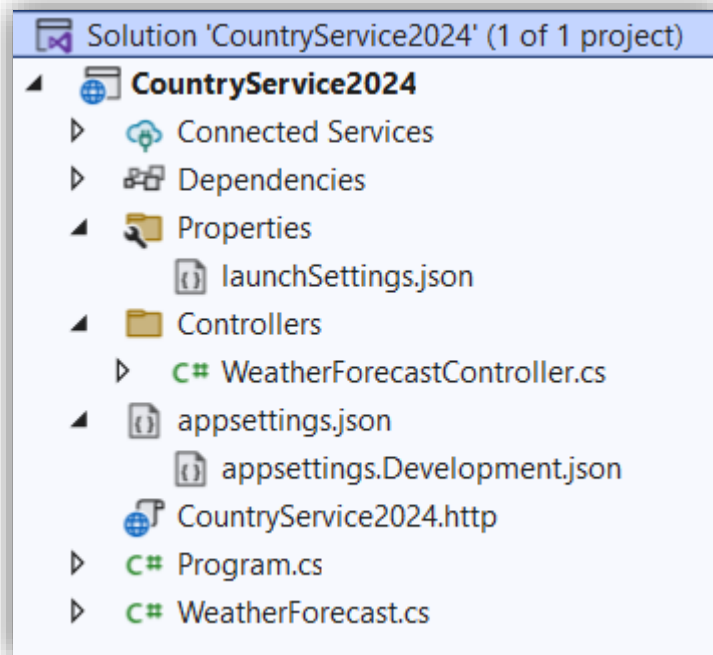
☒ Do not use top-level statements ⓘ

☒ Use controllers ⓘ

Back

Create

Het resultaat is een project met de volgende folders en bestanden :

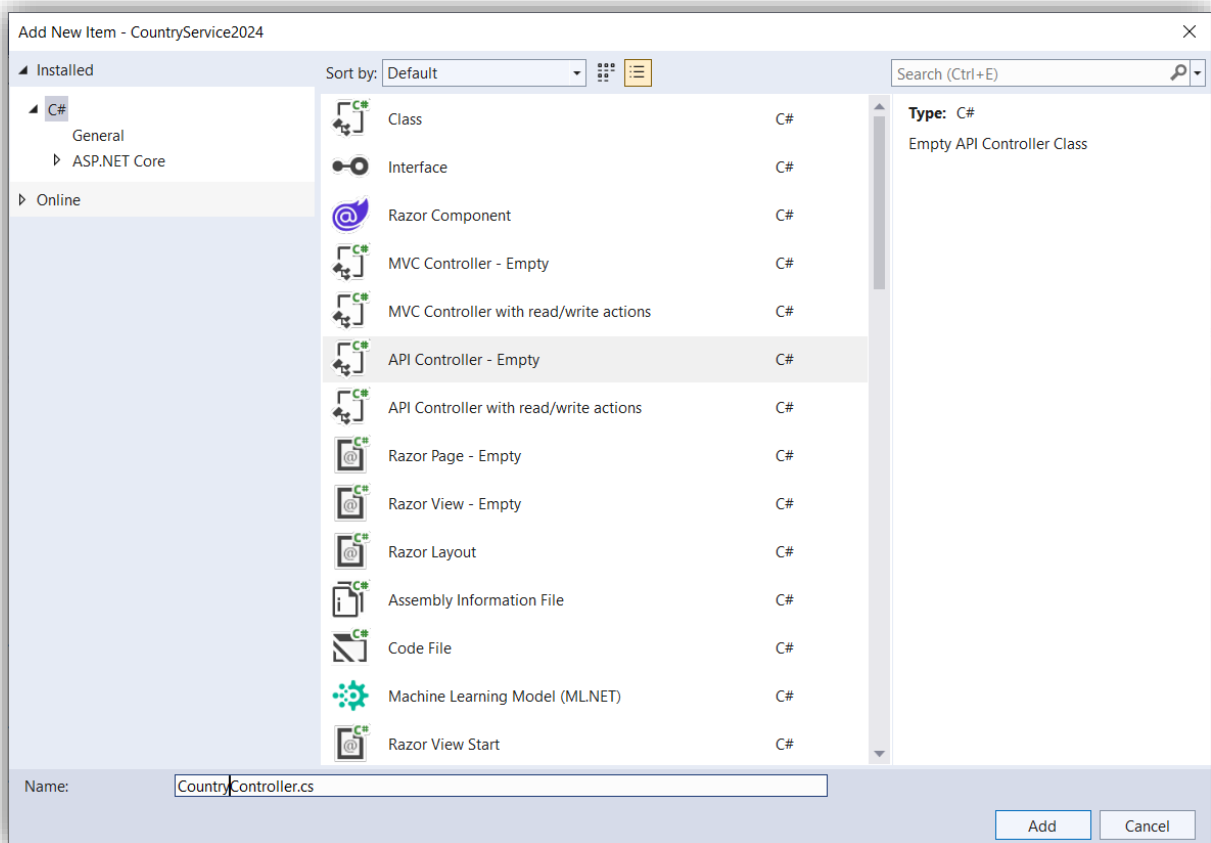
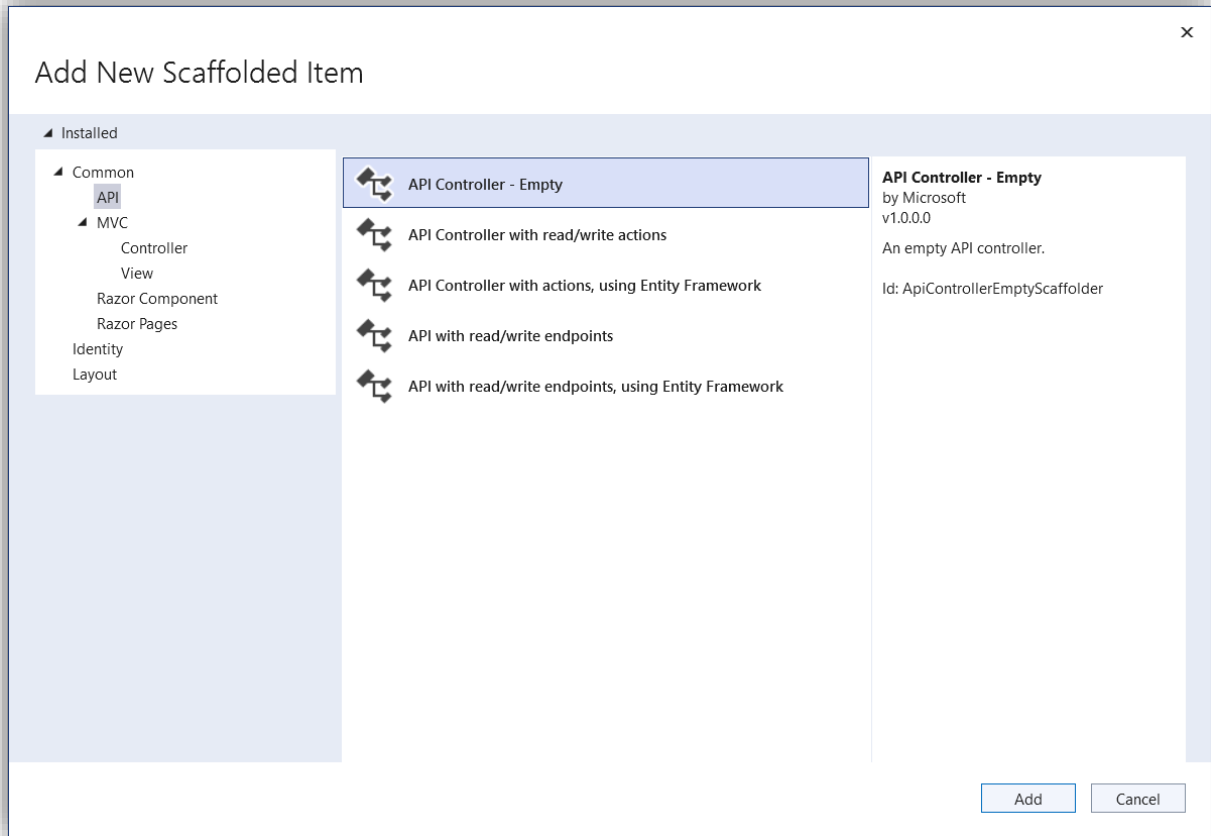


We verwijderen daarbij de WeatherForecast.cs en WeatherForecastController.cs bestanden die standaard als voorbeeld worden aangemaakt, maar die we niet gaan gebruiken.

Controller

De volgende stap is het toevoegen van een 'controller', deze zorgt ervoor dat de bronnen of resources via de API ter beschikking worden gesteld aan de gebruikers. We kunnen de controller zien als een klasse die C# vertaalt naar HTTP.

Om een controller toe te voegen selecteren we de folder controllers en met de rechtermuisknop kunnen we een menu oproepen waarin we via 'Add' en 'Controller' het volgende scherm kunnen oproepen.



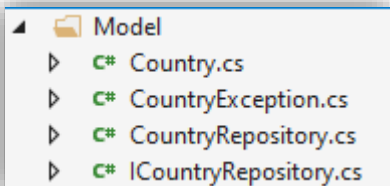
In dit scherm kiezen we voor een lege API Controller (omdat we nu toch enkel de GET requests gaan implementeren) en geven deze de naam CountryController.

Onze controller klasse erft van de klasse ControllerBase en bevat ook de annotaties [ApiController] en [Route] die aangeeft bij welke URL de controller zal worden opgeroepen (ook hierover later meer).

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

Model

Het is natuurlijk de bedoeling om via onze service bronnen/resources ter beschikking te stellen en dus is het ook noodzakelijk om een bron te definiëren en een klasse te voorzien die deze kan bewerken. We voegen daarom de volgende 4 klassen toe in de folder 'Model' :



```
Model
├── Country.cs
├── CountryException.cs
├── CountryRepository.cs
└── ICountryRepository.cs
```

Een eenvoudige klasse Country die slechts een beperkt aantal attributen bevat.

```

public class Country
{
    0 references
    public Country(string name, string capital, string continent)
    {
        Name = name;
        Capital = capital;
        Continent = continent;
    }
    5 references
    public Country(int id, string name, string capital, string continent)
    {
        Id = id;
        Name = name;
        Capital = capital;
        Continent = continent;
    }

    7 references
    public int Id { get; set; }
    2 references
    public string Name { get; set; }
    2 references
    public string Capital { get; set; }
    2 references
    public string Continent { get; set; }
}

```

Een Exception klasse voor als er iets fout gaat :

```

public class CountryException : Exception
{
    4 references
    public CountryException(string message) : base(message)
    {
    }
}

```

De bewerkingen op de collectie van onze objecten implementeren we door middel van de CountryRepository en om deze klasse los te kunnen koppelen definiëren we ook de interface ICountryRepository.

```

public interface ICountryRepository
{
    0 references
    void AddCountry(Country country);
    0 references
    Country GetCountry(int id);
    0 references
    IEnumerable<Country> GetAll();
    0 references
    void RemoveCountry(Country country);
    0 references
    void UpdateCountry(Country country);
}

```

We houden de implementatie van de repository bewust heel eenvoudig zodat we ons kunnen focussen op de werking van de service. Er is daarom gekozen om dictionary te gebruiken die slechts 5 objecten bevat en deze in de constructor te initialiseren.

```

public class CountryRepository : ICountryRepository
{
    private Dictionary<int, Country> data = new Dictionary<int, Country>();

    0 references
    public CountryRepository()
    {
        data.Add(1, new Country(1, "België", "Brussel", "Europa"));
        data.Add(2, new Country(2, "Peru", "Lima", "Zuid-Amerika"));
        data.Add(3, new Country(3, "Duitsland", "Berlijn", "Europa"));
        data.Add(4, new Country(4, "Zweden", "Stockholm", "Europa"));
        data.Add(5, new Country(5, "Noorwegen", "Oslo", "Europa"));
    }
}

```

```

public IEnumerable<Country> GetAll()
{
    return data.Values;
}
2 references
public Country GetCountry(int id)
{
    if (data.ContainsKey(id))
        return data[id];
    else
        throw new CountryException("country doesn't exist");
}

```

```

public void AddCountry(Country country)
{
    if (!data.ContainsKey(country.Id))
        data.Add(country.Id, country);
    else
        throw new CountryException("country already added");
}
1 reference
public void RemoveCountry(Country country)
{
    if (data.ContainsKey(country.Id))
        data.Remove(country.Id);
    else
        throw new CountryException("country doesn't exist");
}
1 reference
public void UpdateCountry(Country country)
{
    if (data.ContainsKey(country.Id))
        data[country.Id] = country;
    else
        throw new CountryException("country doesn't exist");
}

```

Om de repository te kunnen gebruiken in de controller voegen we een variabele van het type `ICountryRepository` toe in de klasse `CountryController` en initialiseren deze in de constructor.

```

public class CountryController : ControllerBase
{
    private ICountryRepository repo;

    0 references
    public CountryController(ICountryRepository repo)
    {
        this.repo = repo;
    }
}

```

Via dependency injection (hierover later meer) geven we de klasse `CountryRepository` mee in de constructor van de controller. Om dit te realiseren voegen we de volgende lijn code toe in de program klasse :

```
builder.Services.AddSingleton<ICountryRepository, CountryRepository>();
```



```

using CountryService.Model;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddSingleton<ICountryRepository, CountryRepository>();

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

```

GET return types

Specifiek return type

Om een methode te linken aan een GET request volstaat het om de annotatie [HttpGet] toe te voegen voor de methode die moet worden uitgevoerd. (Routing zal in één van de volgende hoofdstukken worden besproken)

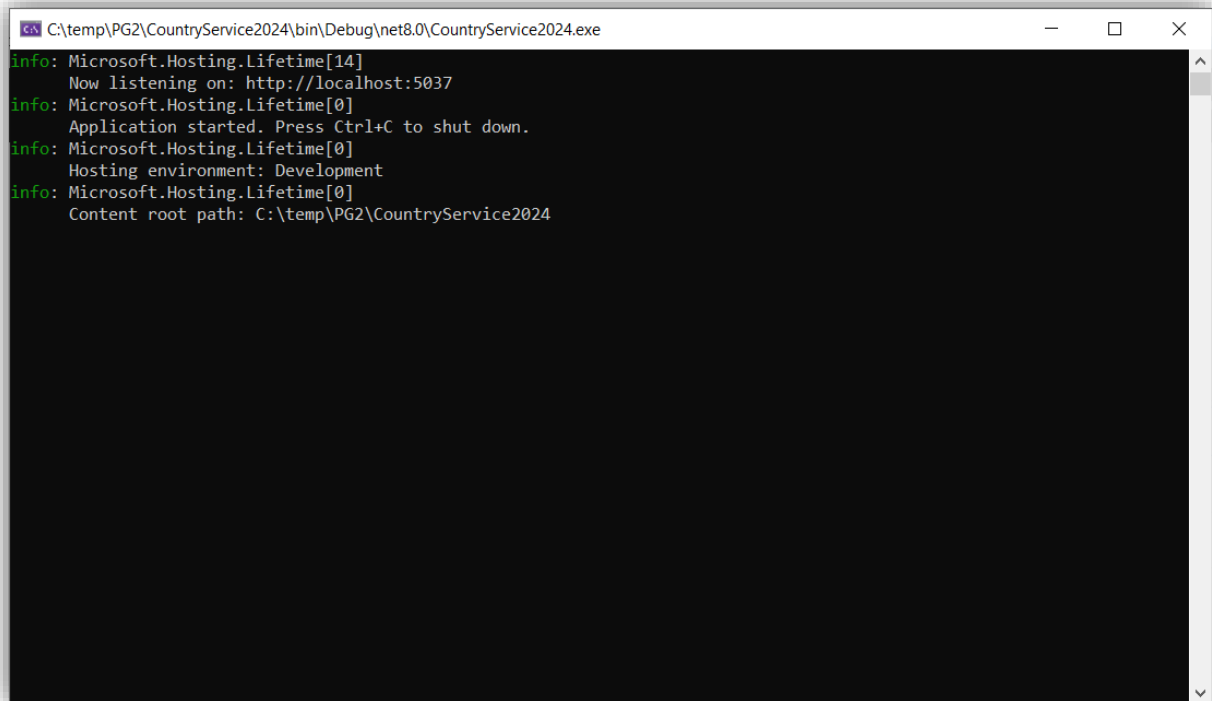
Specifiëren wat de methode precies teruggeeft kan door middel van een specifiek return type. In het volgende voorbeeld geven we een IEnumerable<Country> terug als antwoord.

```

// GET: api/Country
[HttpGet]
0 references
public IEnumerable<Country> Get()
{
    return repo.GetAll();
}

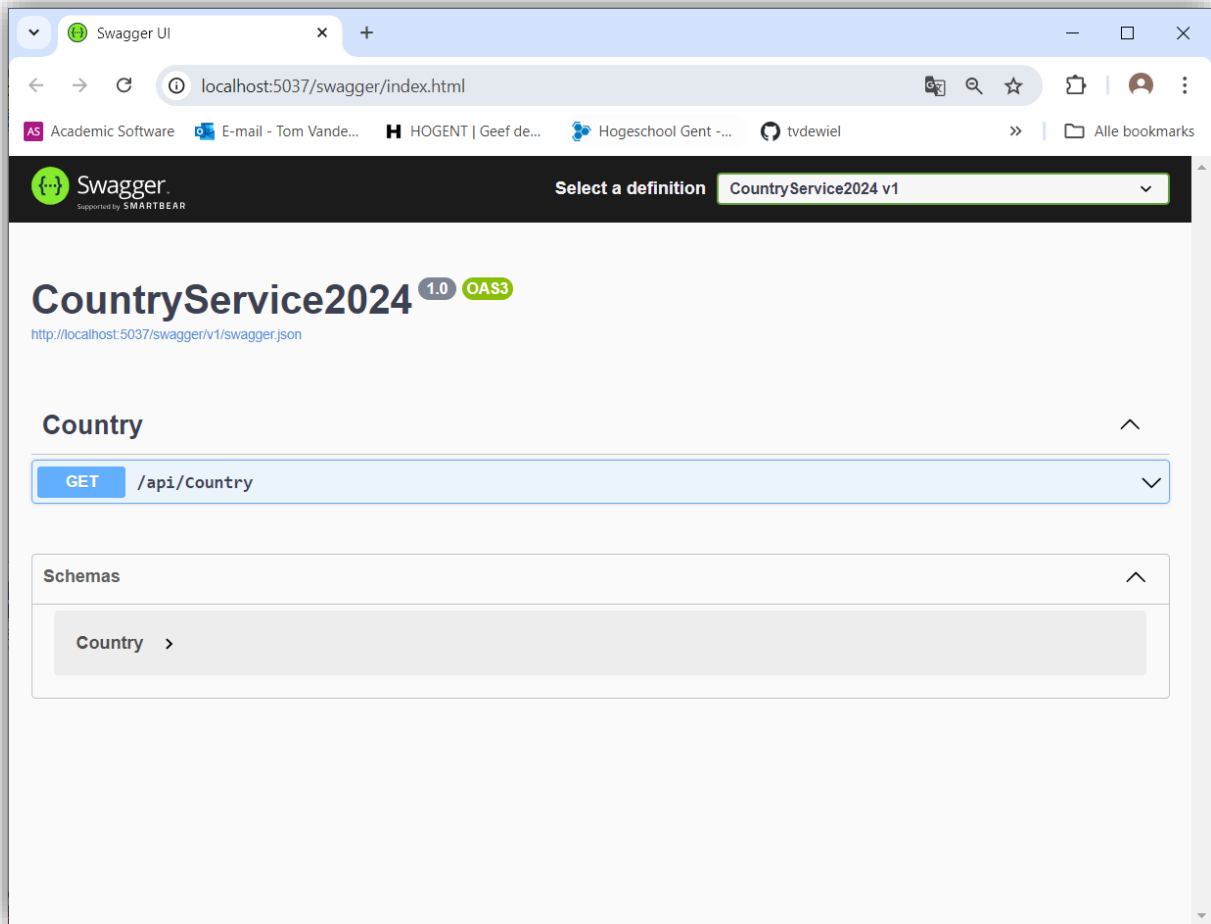
```

Voeren we onze code uit dan verschijnt het volgende console-venster :



```
C:\temp\PG2\CountryService2024\bin\Debug\net8.0\CountryService2024.exe
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5037
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\temp\PG2\CountryService2024
```

Automatisch zal de browser worden opgestart met de Swagger UI. Daarmee kunnen we zien welke controllers en methodes er beschikbaar zijn. Dit is een eenvoudige manier om de service uit te proberen. Er zijn ook andere tools beschikbaar om de service te testen, zoals Postman die handiger is als je je requests wenst op te slaan om uit te voeren. Je hoeft dan niet telkens de gegevens in te voeren.



Wanneer we op het pijltje klikken in het balkje van de GET-methode dan krijgen we verdere info te zien zoals hoe de response (antwoord) er uit zal zien.

CountryService20241.0OAS3

http://localhost:5037/swagger/v1/swagger.json

Country

GET/api/Country

Try it out

Parameters

No parameters

Responses

Code	Description	Links
200	Success	No links

Media type

text/plain

Controls Accept header.

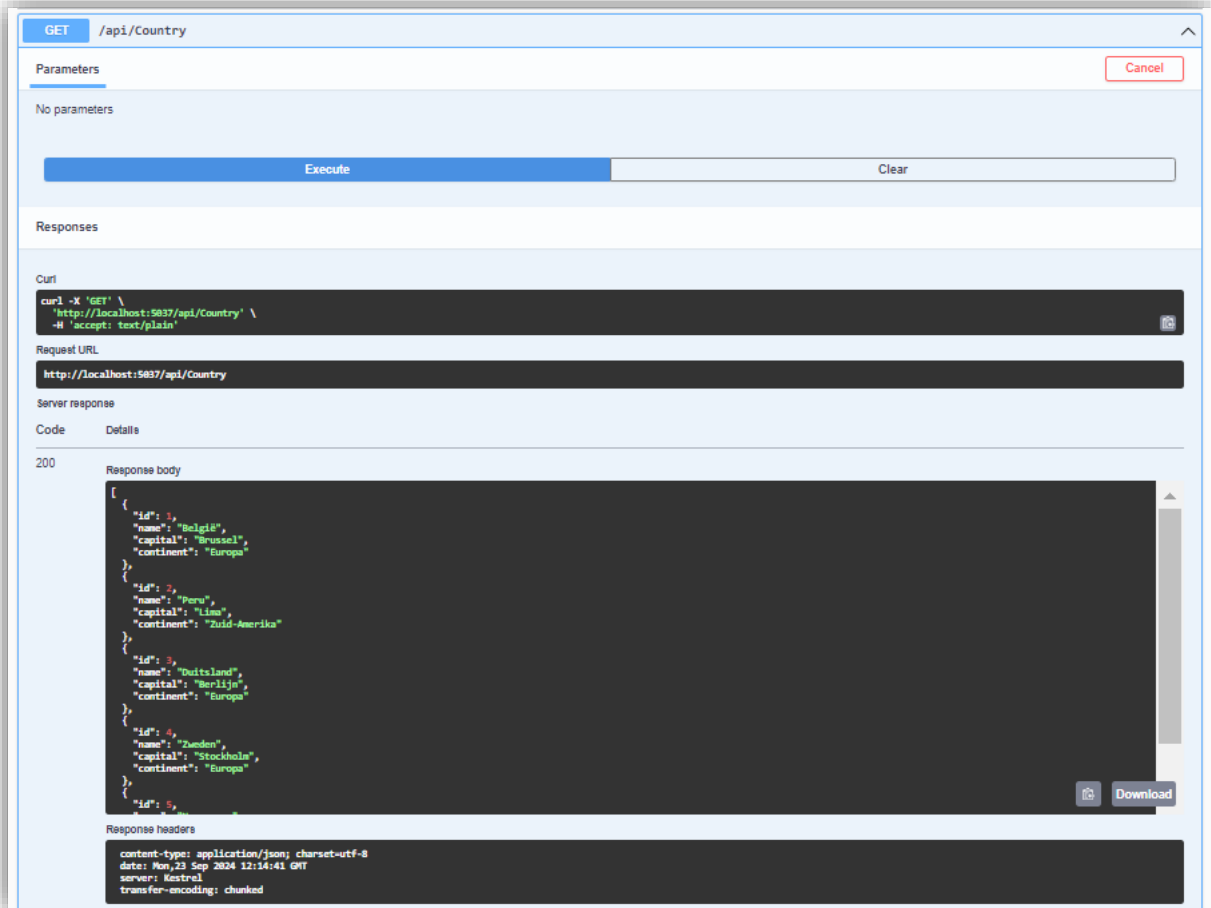
Example ValueSchemas

```
{  "id": 0,  "name": "string",  "capital": "string",  "continent": "string"}
```

Schemas

Country { id integer(\$int32) name string nullable: true capital string nullable: true continent string nullable: true}

Klikken we op “try it out” en daarna “execute” dan wordt de GET-request uitgevoerd en krijgen we het antwoord te zien. Het antwoord dat we terugkrijgen is de lijst van Country-objecten (in JSON formaat) en de status-code 200 Ok.



Je kan de GET-request ook oproepen door de URL in je browser in te typen, het antwoord is dan een JSON-bericht met de gegevens.

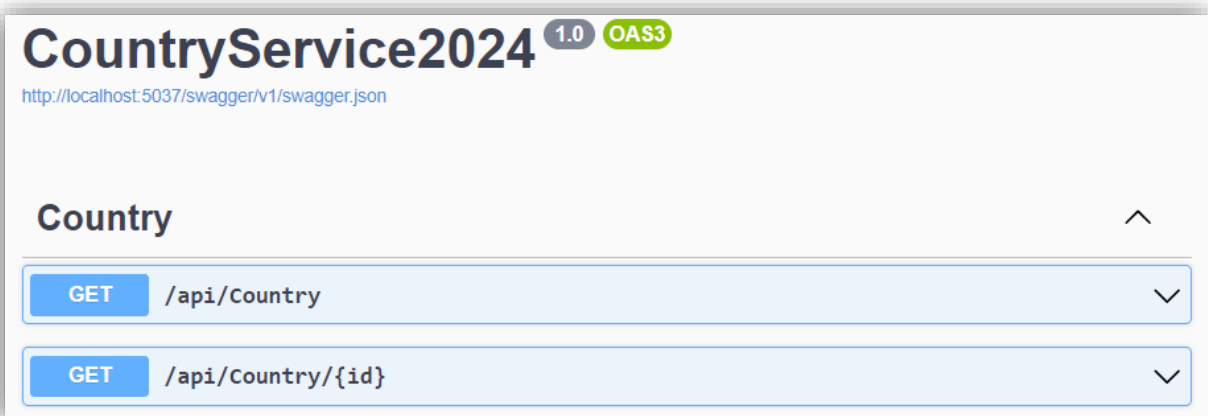


Om een specifiek Country-object op te vragen op basis van zijn id voegen we een tweede methode toe in de controller klasse die er als volgt uit ziet :

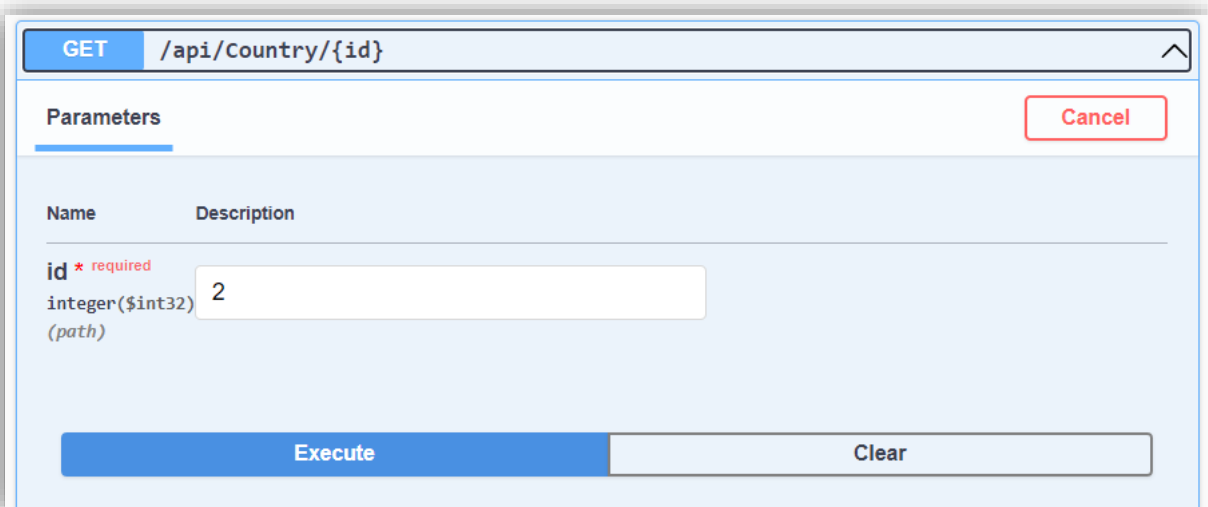
```
// GET: api/Country/5
[HttpGet("{id}")]
public Country Get(int id)
{
    return repo.GetCountry(id);
}
```

De HttpGet notatie bevat nu ook de string "{id}" die aangeeft dat er een parameter id in de URL aanwezig zal zijn en die aan de methode zal worden doorgegeven. De parameter wordt achteraan de 'route' van de controller toegevoegd. De methode geeft nu slechts 1 object meer terug die behoort bij de meegegeven id.

Starten we de service dan toont de Swagger UI nu ook de tweede methode.



We kunnen deze nu uitvoeren en een parameter meegeven.



Curl

```
curl -X 'GET' \
  'http://localhost:5037/api/Country/2' \
  -H 'accept: text/plain'
```

Request URL

http://localhost:5037/api/Country/2

Server response

Code	Details
200	<p>Response body</p> <pre>{ "id": 2, "name": "Peru", "capital": "Lima", "continent": "Zuid-Amerika" }</pre> <p>Response headers</p> <pre>content-type: application/json; charset=utf-8 date: Mon, 23 Sep 2024 14:30:52 GMT server: Kestrel transfer-encoding: chunked</pre>

De URL bevat nu de ingegeven ID en het antwoord is nu enkel het Country-object met id=2.

Wat gebeurt er nu als de betreffende id niet kan gevonden worden ? We proberen het uit.

http://localhost:5037/api/Country/25

Server response

Code	Details
500	<p>Error: Internal Server Error</p> <p>Response body</p> <pre>CountryService2024.Model.CountryException: country doesn't exist at CountryService2024.Model.CountryRepository.GetCountry(Int32 id) in C:\temp\PG2\CountryService2024\Model\CountryRepository.cs:line 32 at CountryService2024.Controllers.CountryController.Get(Int32 id) in C:\temp\PG2\CountryService2024\Controllers\CountryController.cs:line 34 at lambda method2(Closure, Object, Object[]) at Microsoft.AspNetCore.Mvc.Infrastructure.ActionMethodExecutor.SyncObjectResultExecutor.Execute(ActionContext actionContext, IActionResultTypeMapper mapper, ObjectMethodExecutor executor, Object controller, Object[] arguments) at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.<InvokeActionMethodAsync>g__Logged 12_1(ControllerActionInvoker invoker) at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.<InvokeNextActionFilterAsync>g__Awaited 10_0(ControllerActionInvoker invoker, Task lastTask, State next, Scope scope, Object state, Boolean isCompleted) at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.Rethrow(ActionExecutedContextSealed context) at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.Next(State& next, Scope& scope, Object& state, Boolean& isCompleted) at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.InvokeInnerFilterAsync() --- End of stack trace from previous location --- at Microsoft.AspNetCore.Mvc.Infrastructure.ResourceInvoker.<InvokeFilterPipelineAsync>g__Awaited 20_0(ResourceInvoker invoker, Task lastTask, State next, Scope scope, Object state, Boolean isCompleted) at Microsoft.AspNetCore.Mvc.Infrastructure.ResourceInvoker.<InvokeAsync>g__Logged 17_1(ResourceInvoker invoker) at Microsoft.AspNetCore.Mvc.Infrastructure.ResourceInvoker.<InvokeAsync>g__Logged 17_1(ResourceInvoker invoker) at Microsoft.AspNetCore.Authorization.AuthorizationMiddleware.Invoke(HttpContext context) at Swashbuckle.AspNetCore.SwaggerUI.SwaggerUIMiddleware.Invoke(HttpContext httpContext) at Swashbuckle.AspNetCore.Swagger.SwaggerMiddleware.Invoke(HttpContext httpContext, ISwaggerProvider swaggerProvider) at Microsoft.AspNetCore.Authentication.AuthenticationMiddleware.Invoke(HttpContext context) at Microsoft.AspNetCore.Diagnostics.DeveloperExceptionPageMiddlewareImpl.Invoke(HttpContext context)</pre> <p>HEADERS</p> <pre>Accept: text/plain Connection: keep-alive Host: localhost:5037 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/128.0.0.0 Safari/537.36 Accept-Encoding: gzip, deflate, br, q=*</pre>

Het ingeven van een niet bestaande id geeft als resultaat een statuscode 500 (internal server error) en in de body van de message vinden we de exception. Dit is echter niet de bedoeling. In de eerste plaats is dit geen fout van de server, maar een fout van de gebruiker (client) en ten tweede willen we zeker niet de exception weergeven in de message (dit geeft namelijk info over de interne werking en kan tot beveiligingsproblemen leiden).

Een mogelijke oplossing is om de fout op te vangen in de Get methode en een null-waarde terug te geven als er geen object kan worden gevonden voor de opgegeven id.

```
// GET: api/Country/5
[HttpGet("{id}")]
public Country Get(int id)
{
    try
    {
        return repo.GetCountry(id);
    }
    catch(CountryException ex) { return null; }
}
```

Request URL

http://localhost:5037/api/Country/25

Server response

Code	Details
------	---------

204

Undocumented

Response headers

date: Mon, 23 Sep 2024 14:39:27 GMT
server: Kestrel

In plaats van een internal server error krijgen we nu de statuscode 204 No Content. Ook dit is niet de oplossing die we zoeken. In plaats daarvan wensen we een status code 400 die aangeeft dat het om een client probleem gaat. Om dit te bereiken kan de code worden aangepast door expliciet de statuscode in te stellen bij de response, zoals in het volgende voorbeeld.


```
// GET: api/Country/5
[HttpGet("{id}")]
public Country Get(int id)
{
    try
    {
        return repo.GetCountry(id);
    }
    catch(CountryException ex)
    {
        Response.StatusCode = 400;
        return null;
    }
}
```

Request URL

http://localhost:5037/api/Country/25

Server response

Code	Details
400	<div> <div>Undocumented</div> <div>Error: Bad Request</div> </div> <div>Response headers</div> <div> content-length: 0 date: Mon, 23 Sep 2024 14:42:16 GMT server: Kestrel </div>

IActionResult en IActionResult<T>

In plaats van een specifiek type terug te geven kunnen we ook een IActionResult teruggeven die een statuscode combineert met het resultaat (data). Deze interface wordt door een aantal klassen geïmplementeerd zoals Ok(), NotFound(), NoContent() en BadRequest(). De methode kan dan herschreven worden zoals in het volgende voorbeeld :

```
// GET: api/Country/5
[HttpGet("{id}")]
public IActionResult Get(int id)
{
    try
    {
        return Ok(repo.GetCountry(id));
    }
    catch(CountryException ex)
    {
        return NotFound();
    }
}
```

En de resultaten zien er dan als volgt uit :

Request URL

http://localhost:5037/api/Country/2

Server response


Code

Details

200

Response body

```
{
  "id": 2,
  "name": "Peru",
  "capital": "Lima",
  "continent": "Zuid-Amerika"
}
```


Download

Response headers

```
content-type: application/json; charset=utf-8
date: Mon, 23 Sep 2024 14:47:37 GMT
server: Kestrel
transfer-encoding: chunked
```

Responses

Code

Description

Links

200

Success

No links

Request URL

<http://localhost:5037/api/Country/25>

Server response

Code	Details
404	Error: Not Found

Undocumented

Response body

```
{  "type": "https://tools.ietf.org/html/rfc9110#section-15.5.5",  "title": "Not Found",  "status": 404,  "traceId": "00-ed7d2185321354b5d26abe8e8936dd48-42ea4dcb331e6de5-00"}
```

Response headers

```
content-type: application/problem+json; charset=utf-8
date: Mon, 23 Sep 2024 14:49:08 GMT
server: Kestrel
transfer-encoding: chunked
```

Responses

Code	Description	Links
200	Success	No links

Door gebruik te maken van `ActionResult<T>` hoeven we het resultaat niet meer via de `Ok()` methode in te pakken en terug te geven, we kunnen nu rechtstreeks een object van het type `<T>` teruggeven. Ook aan client zijde is het nu duidelijk welk object er wordt teruggegeven en is er geen casting meer nodig.

```
[HttpGet("{id}")]
public ActionResult<Country> Get(int id)
{
    try
    {
        return repo.GetCountry(id);
    }
    catch (CountryException ex)
    {
        return NotFound();
    }
}
```

Request URL

`http://localhost:5037/api/Country/2`

Server response

Code	Details
200	<p>Response body</p> <pre>{ "id": 2, "name": "Peru", "capital": "Lima", "continent": "Zuid-Amerika" }</pre> <p>Response headers</p> <pre>content-type: application/json; charset=utf-8 date: Mon, 23 Sep 2024 14:51:40 GMT server: Kestrel transfer-encoding: chunked</pre>

Responses

Code	Description	Links
200	<p>Success</p> <p>Media type</p> <p><input type="text" value="text/plain"/></p> <p>Controls Accept header.</p> <p>Example Value Schema</p> <pre>{ "id": 0, "name": "string", "capital": "string", "continent": "string" }</pre>	No links

Wensen we extra info mee te geven met de NotFound status code dan kunnen we dat op de volgende manier doen :

```
[HttpGet("{id}")]
public ActionResult<Country> Get(int id)
{
    try
    {
        return repo.GetCountry(id);
    }
    catch (CountryException ex)
    {
        return NotFound(ex.Message);
    }
}
```

Het resultaat is dan het volgende :

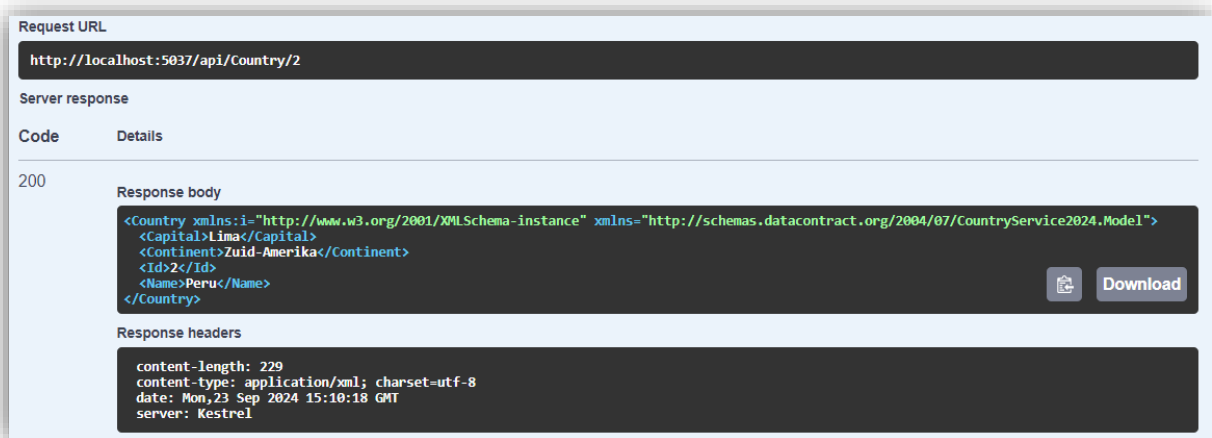
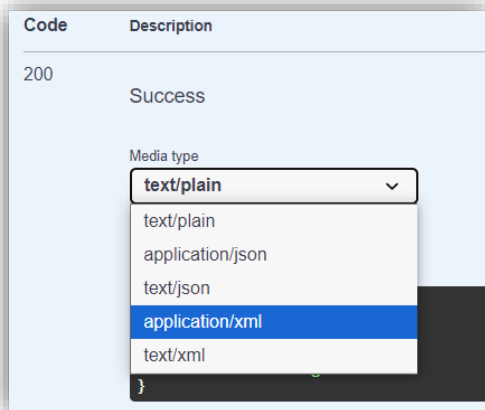
The screenshot shows the Swagger UI interface. At the top, the 'Request URL' is set to `http://localhost:5037/api/Country/25`. Below this, the 'Server response' section is expanded, showing a '404' status code with the message 'Error: Not Found'. The 'Response body' is displayed as 'country doesn't exist'. The 'Response headers' section shows the following information: `content-type: text/plain; charset=utf-8`, `date: Mon, 23 Sep 2024 14:54:08 GMT`, `server: Kestrel`, and `transfer-encoding: chunked`. There is a 'Download' button next to the response body.

GET return format

Tot nu toe hebben we de data in de response body steeds als JSON meegegeven. Maar er zijn nog andere mogelijkheden natuurlijk en één veelgebruikte daarvan is XML. Om het antwoord in XML terug te krijgen moeten we bij het sturen van het verzoek aangeven dat we een XML antwoord wensen. Dat kunnen we doen door in de header van het verzoek het key value paar `Accept : application/xml` mee te geven. Dat kan via de Swagger UI door te kiezen voor xml. Maar vooraleer we dit kunnen kiezen moeten we bij de Services aangeven dat we Xml als output wensen en dat kan door de volgende lijn code te voorzien.

```
builder.Services.AddControllers().AddXmlDataContractSerializerFormatters();
```

In de Swagger UI kiezen we nu xml en drukken opnieuw op de execute-knop, het resultaat wordt nu als XML weergegeven.



Opmerking : voeg lege constructor toe in klasse country.

HEAD

Een HEAD request lijkt op een GET request maar geeft enkel de headers terug en niet de body. Deze request wordt voornamelijk gebruikt om te kijken of een bepaalde request een bestaand antwoord zal geven (zonder de eigenlijke data op te vragen). We kunnen hiermee bijvoorbeeld controleren of een bepaalde bron bestaat, wat interessant kan zijn als het bijvoorbeeld over een grote bron gaat aangezien het HEAD verzoek dan een stuk sneller zal zijn dan het GET verzoek. HEAD requests kunnen ook interessant zijn voor caching, indien het document is aangepast (vb LastModified / ContentLength header info is anders) kan een nieuwe versie worden afgehaald.

In code is het vrij eenvoudig om een HEAD request toe te voegen, we hoeven enkel een extra annotatie toe te voegen.

```
// GET: api/Country
[HttpGet]
[HttpHead]
0 references
public IEnumerable<Country> GetAll()
{
    return repo.GetAll();
}
```

```
[HttpGet("{id}")]
[HttpHead("{id}")]
0 references
public ActionResult<Country> Get(int id)
{
    try
    {
        return repo.GetCountry(id);
    }
    catch (CountryException ex)
    {
        return NotFound(ex.Message);
    }
}
```

CountryService2024 1.0 OAS3

<http://localhost:5037/swagger/v1/swagger.json>

Country ^

GET /api/Country



HEAD /api/Country



GET /api/Country/{id}



HEAD /api/Country/{id}





Filtering en Searching

Indien we niet alle landen wensen te selecteren, maar enkel een selectie daarvan kunnen we gebruik maken van filtering. De GetAll methode moet nu worden aangepast zodat er met een filter kan worden gewerkt.

In ons voorbeeld implementeren we nu de mogelijkheid om te filteren op basis van de variabele continent. Naast de parameter en zijn type voegen we ook nog toe vanwaar deze parameter zal komen uit het GET request via een annotatie. Hier is geopteerd om de parameter uit de Query input te halen.

```
// GET: api/Country
[HttpGet]
[HttpHead]
0 references
public IEnumerable<Country> Getall([FromQuery] string continent)
{
    if (!string.IsNullOrEmpty(continent))
        return repo.GetAll(continent);
    else
        return repo.GetAll();
}
```

Aangezien we wensen te filteren, moeten we dit ook implementeren in onze repository en voegen we volgende functie toe :

```
public IEnumerable<Country> GetAll(string continent)
{
    return data.Values.Where(x => x.Continent == continent);
}
```

Het GET request ziet er dan als volgt uit : <http://localhost:50051/api/country?Continent=Europa>. De query parameters worden aan de URL toegevoegd door middel van een vraagteken en een naam/waarde paar.

GET

/api/Country

⌵

Parameters

Cancel

Name	Description
continent string (query)	<input type="text" value="Europa"/>

Execute

Clear

Request URL

http://localhost:5037/api/Country?continent=Europa

Server response

Code	Details
200	<div>Response body</div> <pre>[{ "id": 1, "name": "België", "capital": "Brussel", "continent": "Europa" }, { "id": 3, "name": "Duitsland", "capital": "Berlijn", "continent": "Europa" }, { "id": 4, "name": "Zweden", "capital": "Stockholm", "continent": "Europa" }, { "id": 5, "name": "Noorwegen", "capital": "Oslo", "continent": "Europa" }]</pre> <div><div>Download</div></div>

Wensen we meerdere parameters te gebruiken dan kan dat bijvoorbeeld als volgt :

```
// GET: api/Country
[HttpGet]
[HttpHead]
0 references
public IEnumerable<Country> GetAll([FromQuery] string continent, [FromQuery] string capital)
{
    if (!string.IsNullOrEmpty(continent))
    {
        if (!string.IsNullOrEmpty(capital.Trim()))
            return repo.GetAll(continent, capital);
        else
            return repo.GetAll(continent);
    }
    else
        return repo.GetAll();
}
```

```
public IEnumerable<Country> GetAll(string continent, string capital)
{
    return data.Values.Where(x => x.Continent == continent && x.Capital==capital);
}
```

GET /api/Country

Parameters
Cancel

Name	Description
continent string (query)	<input type="text" value="Europa"/>
capital string (query)	<input type="text" value="Oslo"/>

Execute

Request URL

http://localhost:5037/api/Country?continent=Europa&capital=Oslo

Server response

Code

Details

200

Response body

```
[
  {
    "id": 5,
    "name": "Noorwegen",
    "capital": "Oslo",
    "continent": "Europa"
  }
]
```



Download

Response headers

```
content-type: application/json; charset=utf-8
date: Mon, 23 Sep 2024 15:26:35 GMT
server: Kestrel
transfer-encoding: chunked
```