REST services routing

In de Controller klasse definiëren we (business) acties die we wensen uit te voeren in reactie op een bepaald verzoek/request. Dit proces waarbij we inkomende verzoeken mappen naar applicatielogica noemen we 'routing' waarbij de controller gebruik maakt van de routing middleware.

Er worden twee type routing onderscheiden, namelijk conventional routing en attribute routing. Voor REST services maken we gebruik van attribute routing, waarbij elke actie direct aan één of meerdere specifieke URLs wordt gekoppeld door middel van een attribuut. Bij attribute routing wordt een dictionary opgebouwd met als sleutel een URL en als value de daarbij horende actie. Deze dictionary wordt opgebouwd door voor alle controllers de verschillende methodes af te lopen die een [Route] attribuut hebben. Een bepaalde actie kan wel vanaf verschillende URLs worden uitgevoerd.

Om gebruik te maken van routing voegen we in de Configer methode van de Startup klasse app.UseRouting() en app.UseEndpoints() toe. De eerste methode koppelt de request aan een endpoint, terwijl de tweede methode het betreffende endpoint uitvoert. Op deze manier is er een ontkoppeling tussen het matchen van de route en de uitvoering van de daaraan gekoppelde logica. Tussen beide middleware vinden we app.UseAuthorization() terug die nu gebruik kan maken van de matching info alvorens een beslissing te nemen aangaande authorizatie.

De MapControllers() methode mapt de attributen die in de controller worden gevonden aan de bijhorende acties.

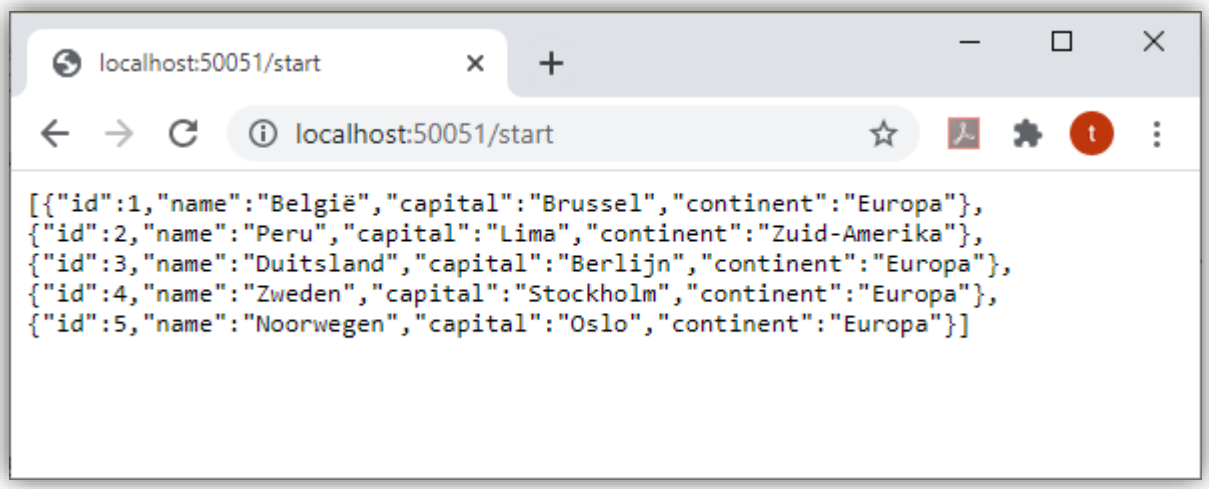
```
app.UseRouting();

app.UseAuthorization();

app.UseEndpoints(endpoints =>
{
    endpoints.MapControllers();
});
```

Route

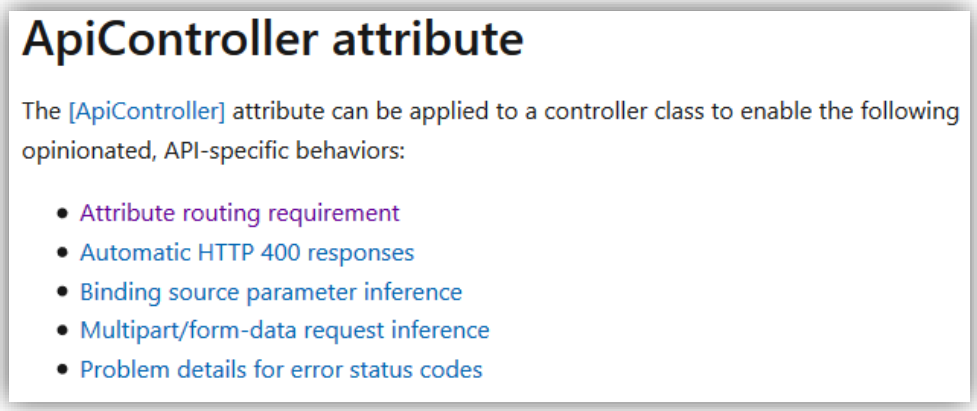
Met het attribuut [Route("...")] geven we aan welke URL er aan de volgende methode wordt gekoppeld. In het volgende voorbeeld hebben we een methode Start die we koppelen aan de URL /start.

```
[ApiController]
1 reference
public class CountryController : ControllerBase
{
    private ICountryRepository repo;

    0 references
    public CountryController(ICountryRepository repo)...

    [Route("start")]
    0 references
    public IActionResult Start()
    {
        return Ok(repo.GetAll());
    }
```

Bekijken we deze URL in een browser, dan krijgen we het volgende resultaat :



Onze Controller klasse heeft ook nog een attribuut [ApiController] waarmee de volgende eigenschappen worden ingesteld :

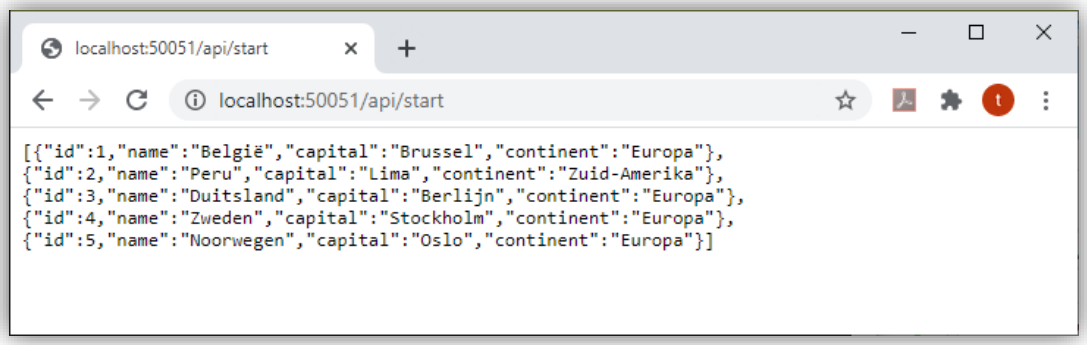


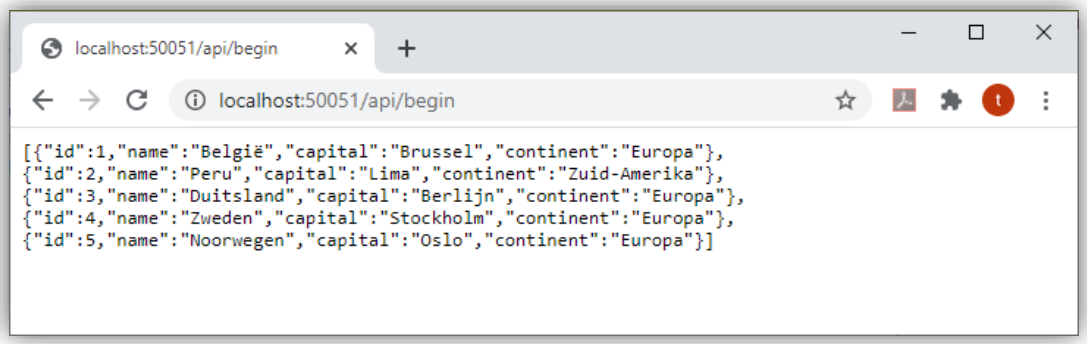
Meerdere URLs

Zoals reeds vermeld in het begin van dit hoofdstuk, is het ook mogelijk om meerdere URLs te koppelen aan één methode. Dit kan door verschillende [Route] attributen te gebruiken. In het volgende voorbeeld kan de methode Start zowel via de URL /api/start als /api/begin worden uitgevoerd.

```
[ApiController]
1 reference
public class CountryController : ControllerBase
{
    private ICountryRepository repo;

    0 references
    public CountryController(ICountryRepository repo)...

    [Route("api/start")]
    [Route("api/begin")]
    0 references
    public IActionResult Start()
    {
        return Ok(repo.GetAll());
    }
```

In onze browser proberen we beiden uit en zien we hetzelfde resultaat.





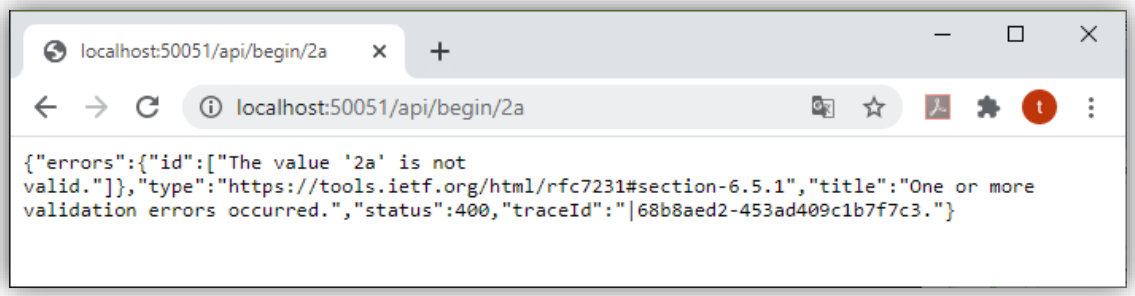
Parameters

Aan methodes kunnen ook parameters worden meegegeven die uit de URL worden afgeleid. Dit kan worden aangegeven in de 'route' bij het [Routing] attribuut door de betreffende parameter tussen accolades te plaatsen, zoals {id} in het voorbeeld.

```
[ApiController]
1 reference
public class CountryController : ControllerBase
{
    private ICountryRepository repo;

    0 references
    public CountryController(ICountryRepository repo)...

    [Route("api/start/{id}")]
    [Route("api/begin/{id}")]
    0 references
    public IActionResult Start(int id)
    {
        return Ok(repo.GetCountry(id));
    }
```

De parameter geven we mee in de URL zoals in de route is beschreven, in dit voorbeeld na api/start/. De parameter wordt afgeleid uit de URL en aan de methode meegegeven.

Geven we een niet bestaande parameter mee, dan krijgen we de volgende foutboodschap terug :



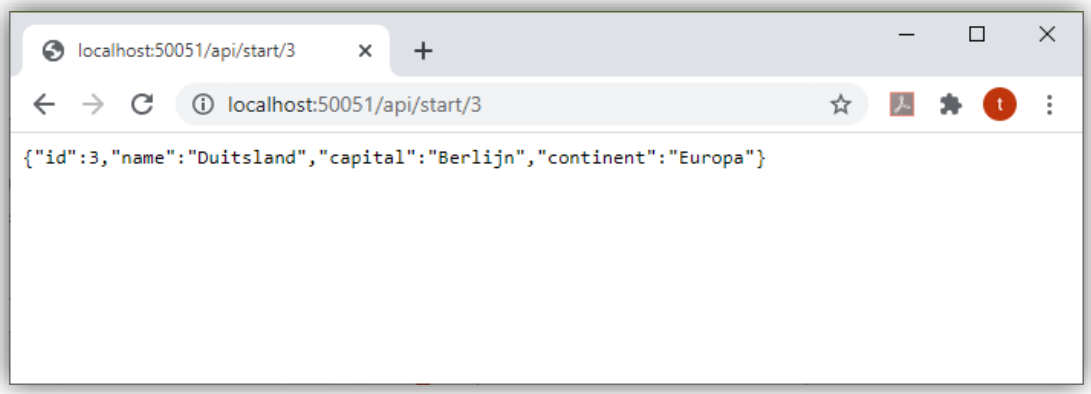
Attribute constraints and defaults

We kunnen ook beperkingen opleggen aan parameters en eventueel een default waarde toekennen. Dat kan zoals in het volgende voorbeeld, waarbij we eisen dat de parameter van het type int moet zijn en indien geen waarde wordt meegegeven dat deze de waarde 3 krijgt.

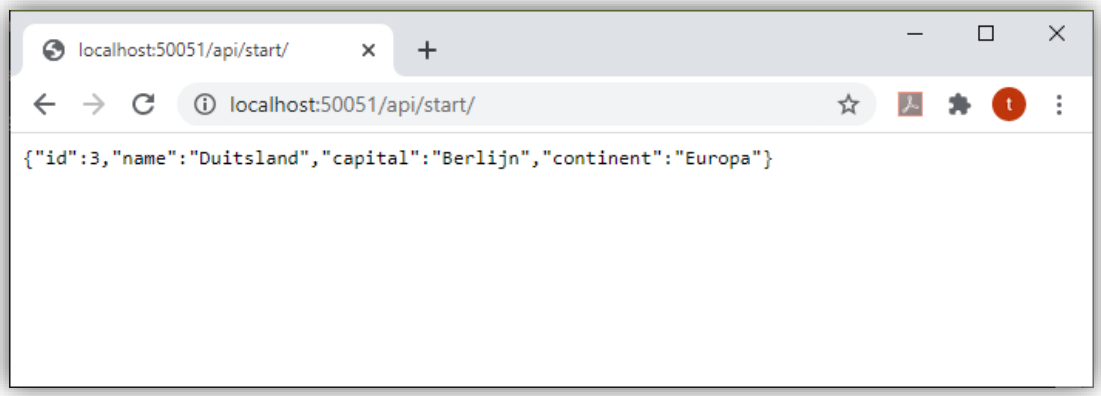
```
[ApiController]
1 reference
public class CountryController : ControllerBase
{
    private ICountryRepository repo;

    0 references
    public CountryController(ICountryRepository repo)...

    [Route("api/start/{id:int=3}")]
    [Route("api/begin/{id:int=3}")]
    0 references
    public IActionResult Start(int id)
    {
        return Ok(repo.GetCountry(id));
    }
```

We testen dit even uit door middel van de volgende URLs :

We geven een correcte parameter mee en krijgen het overeenkomstig element terug.



We geven geen parameter mee en dus wordt de default waarde van 3 gebruikt.



We geven nu een parameter mee die niet voldoet aan de opgelegde voorwaarde, namelijk dat het type int moet zijn. We krijgen nu geen foutmelding dat de parameter niet correct is zoals in de vorige paragraaf, maar een melding dat de pagina niet kan worden gevonden. De routing middleware kan de meegegeven parameter niet omzetten naar een int en dus ook niet koppelen aan de URL /api/begin/{id} en leidt hieruit af dat het hier te maken heeft met de route /api/begin/2a die niet voorkomt in onze applicatie, en geeft dus een page not found error.

Controller route

Indien we over verschillende methodes beschikken en telkens via het [Route] attribuut de bijhorende URL moeten definiëren dan kan dat gauw veel werk opleveren en makkelijk fouten introduceren. Daarom bestaat er ook de mogelijkheid om een gemeenschappelijk deel van de route toe te kennen aan de controller zelf. De [Route] attributen bij de verschillende methodes moeten dan enkel het specifieke/relatieve deel aanduiden ten opzichte van dit gemeenschappelijk deel. In het volgende voorbeeld kennen we de route 'api/country' toe aan de controller en definiëren we enkel het deel van de URL die daaraan moet worden toegevoegd bij elke methode. Voor de methode start wordt dit dan /api/country/start/{id}.

```
[Route("api/Country")]
[ApiController]
1 reference
public class CountryController : ControllerBase
{
    private ICountryRepository repo;

    0 references
    public CountryController(ICountryRepository repo)...

    [Route("start/{id:int=3}")]
    0 references
    public IActionResult Start(int id)
    {
        return Ok(repo.GetCountry(id));
    }
```

We testen dit even uit :

Er bestaat echter nog altijd een mogelijkheid om een absolute URL te definiëren door de meegegeven string in het [Routing] attribuut te laten beginnen met ”/”.

```
[Route("start/{id:int=3}")]
[Route("/start/{id:int=3}")]
0 references
public IActionResult Start(int id)
{
    return Ok(repo.GetCountry(id));
}
```

Even testen :

Token replacement

In het vorige voorbeeld hebben we steeds een string gebruikt met een reeds ingestelde waarde, maar we kunnen ook gebruik maken van tokens (parameters) die automatisch een bepaalde waarde aannemen. De meest gebruikte hierbij zijn [controller] en [Action]. Door in het [Route] attribuut bij de controller het token [controller] op te nemen, wordt automatisch de naam van de controller klasse gebruikt (zonder het deel controller). In het volgende voorbeeld zal dat dus worden /api/Country. Ook de naam van de methode kan automatisch worden overgenomen door middel van het token [Action].

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

    0 references
    public CountryController(ICountryRepository repo)...

    [Route("[Action]/{id:int=3}")]
    0 references
    public IActionResult Start(int id)
    {
        return Ok(repo.GetCountry(id));
    }
```

We proberen dit uit en zien dat 'country' en 'start' de plaats van de tokens hebben ingenomen.

REST werkwoorden

Als laatste punt bij routing hebben we het over de mapping van de REST werkwoorden (GET, POST, PATCH, …). Aan een bepaalde URL, bijvoorbeeld /api/country/1, kunnen namelijk meerdere acties worden gekoppeld, het zou hier bijvoorbeeld kunnen gaan om het opvragen van een element, maar eveneens zou het kunnen gaan over het verwijderen van een element. Om dit onderscheid te maken kan er ook hier gebruik gemaakt worden van attributen. De specifieke attributen die hierbij horen zijn onder andere [HttpGet], [HttpPost], … We gaan hier niet verder op in omdat we deze reeds in de vorige hoofdstukken hebben besproken.

Referenties

https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/routing?view=aspnetcore-3.1

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