



samen sterk voor werk

# C#2013 Web API

Deze cursus is eigendom van de VDAB©

## Inhoudsopgave

<b>1</b>	<b>INLEIDING.....</b>	<b>4</b>
1.1	Doelstelling.....	4
1.2	Vereiste voorkennis.....	4
1.3	Nodige software .....	4
1.4	De voorbeelddata in deze cursus .....	4
1.5	Configuration by Convention .....	4
<b>2</b>	<b>WEB API IS EEN IMPLEMENTATIE VAN REST .....</b>	<b>5</b>
2.1	Algemeen .....	5
2.2	Entities identificeren met URI's.....	5
2.2.1	De verzameling met alle entities van hetzelfde type .....	6
2.2.2	Één entity, aangeduid met zijn unieke identificatie .....	6
2.2.3	Een verzameling entities, aangeduid met een groepsnaam .....	6
2.2.4	Een verzameling entities die je zoekt op andere zoekcriteria.....	6
2.2.5	Associaties tussen entities .....	7
2.2.6	URI templates .....	7
2.3	De client duidt met de standaard HTTP methods de handeling aan die hij op de entities wil uitvoeren.....	7
2.3.1	De client voegt een entity toe met HTTP POST .....	7
2.3.2	De client leest één of meerdere entities met HTTP GET .....	7
2.3.3	De client wijzigt een entity met HTTP PUT .....	8
2.3.4	De client verwijdert een entity met HTTP DELETE .....	8
2.4	Data uitwisselen in meerdere formaten .....	8
2.5	REST gebruikt de response status code om aan te geven of de request wel of niet correct verwerkt werd .....	9
2.6	REST data bevat URI's naar nieuwe data .....	10
<b>3</b>	<b>DE REST SERVICE.....</b>	<b>11</b>
3.1	Algemeen .....	11
3.2	De database.....	11
3.3	Controllers en URI's.....	12
<b>4</b>	<b>GET REQUESTS: DATA LEZEN .....</b>	<b>13</b>

4.1	Algemeen .....	13
4.2	Een GET request naar een verzameling .....	13
4.3	De service uittesten.....	13
4.4	Een GET request naar één resource aan de hand van zijn unieke identificatie .....	14
4.5	Een GET request naar resources aan de hand van andere criteria dan de unieke identificatie .....	15
4.6	Namen van XML elementen en namespace.....	15
4.7	Namen van XML elementen die verzameling voorstellen .....	16
4.8	Responses met hyperlinks.....	17
<b>5</b>	<b>REST SERVICES: ANDERE OPERATIES.....</b>	<b>19</b>
5.1	DELETE requests: data verwijderen .....	19
5.2	POST request: data toevoegen.....	19
5.3	PUT request: data wijzigen todo tot hier .....	20
5.4	Validaties .....	21
5.4.1	Algemeen .....	21
5.4.2	Met validatie attributen aangeven wat er moet gevalideerd worden .....	22
5.4.3	De validatie zelf .....	22
5.4.4	Een response met foutinformatie naar de browser sturen .....	23
5.4.5	Uitgebreidere foutinformatie aanbieden aan de client .....	23
5.5	Automatische online documentatie van je REST service .....	24
<b>6</b>	<b>AFWIJKEN VAN CONFIGURATION BY CONVENTION .....</b>	<b>26</b>
6.1	Algemeen .....	26
6.2	De naam van een method in een controller class.....	26
6.3	De naam van de controller class .....	26
<b>7</b>	<b>REST CLIENTS .....</b>	<b>28</b>
7.1	Algemeen .....	28
7.2	Data uit requests en responses .....	28
7.3	Data lezen met GET requests .....	29
7.4	Data verwijderen met een DELETE request .....	30
7.5	Data toevoegen met een POST request .....	31

---

7.6	Data wijzigen met een PUT request .....	32
<b>8</b>	<b>COLOFON.....</b>	<b>33</b>

# 1 INLEIDING

## 1.1 Doelstelling

Je leert in deze cursus werken met Web API.

Je gebruikt de Web API om:

- vanuit je applicatie de diensten van andere applicaties aan te spreken.
- de diensten van je applicatie aan te bieden aan andere applicaties.

De applicatie die diensten aanspreekt heet de client.

De applicatie die diensten aanbiedt heet de server.

Een browser is een client en de website is een service.

Maar ook andere applicaties roepen mekaars diensten.

Een client van een reisbureau (geschreven in Java) roept de diensten op van een service van een vliegtuigmaatschappij (geschreven in .net).

## 1.2 Vereiste voorkennis

- C# PF
- Entity Framework

## 1.3 Nodige software

- Visual Studio 2013 (met Update 4)
- SQL Server / SQL Server Express
- SQL Server Management Studio Express

## 1.4 De voorbeelddata in deze cursus

De voorbeelddata zal bestaan uit brouwers.

Brouwer
-id: int
-naam: string
-postcode: int
-gemeente: string

Een object of een verzameling objecten (zoals een Brouwer object of een verzameling Brouwer objecten heet bij de Web API ook een *resource*.

## 1.5 Configuration by Convention

Bij de Web API moet je geen configuratie instellen in een XML bestand.

De Web API configureert zichzelf op basis van de namen van classes en methods.

Je moet bij die naamgeving bepaalde conventies volgen.

Dit heet "Configuration by Convention" en zie je stap per stap in de cursus.

## 2 WEB API IS EEN IMPLEMENTATIE VAN REST

### 2.1 Algemeen

De Web API is een implementatie van REST (Representational State Transfer).

- Met REST kunnen applicaties mekaars diensten aanspreken.
- Ook in andere programmeertalen (bvb. java) bestaan REST implementaties.
- REST is een alternatief voor SOAP.  
Met SOAP kunnen applicaties ook mekaars diensten aanspreken. SOAP is echter ingewikkelder dan REST en daardoor moeilijk aan te spreken vanuit JavaScript.

Bij REST wisselen de client en de service data uit over het HTTP protocol.

REST is gebaseerd op volgende principes.

- De service identificeert de resources die hij aanbiedt met URI's.
- De client doet requests naar deze URI's.  
De client duidt met de HTTP method (POST, GET, PUT, DELETE) van de request aan welke handeling (toevoegen, lezen, wijzigen, verwijderen) hij op de resource wil uitvoeren.
- Services en clients wisselen data over resources uit in meerdere formaten
  - (X)HTML als de client een browser is
  - XML als de client geschreven is in Java, .net, Cobol, ...
  - JSON als de client geschreven is in JavaScript  
(JSON is het standaard dataformaat van JavaScript).
- REST gebruikt de HTTP status code van de response om aan te geven of de request wel of niet correct verwerkt werd.
- De data die de service aanbiedt, bevat URI's waarmee de client terug diensten aan de service kan vragen. De browser krijgt op de URI /browsers data over alle browsers. Deze data bevat per browser een hyperlink met de URI waar je de detailinformatie van die ene browser vindt. De gebruiker kan met zijn browser deze hyperlink volgen om die detailinformatie te zien. De gebruiker kreeg de URI met deze detailinformatie aangereikt en hoefde die niet zelf te verzinnen.
- De service is stateless.  
Dit houdt in dat de service geen data bijhoudt als HTTP session variabelen.

Je leert de eerste vier REST principes hier onder.

### 2.2 Entities identificeren met URI's

Elke resource krijgt een URI.

In theorie is deze URI vrij te kiezen. De URI /a1bd89 zou kunnen browser 1 voorstellen en de URI /9zdb3 zou kunnen browser 2 voorstellen.

De meeste websites gebruiken echter betekenisvolle URI's.

Een betekenisvolle URI verduidelijkt aan mensen én zoekrobots welke data de URI aanbiedt.

De opbouw van een betekenisvolle URI verschilt volgens de resource die de URI voorstelt.

Je maakt hierbij onderscheid tussen

- de verzameling met alle entities van hetzelfde type
- één entity, aangeduid met zijn unieke identificatie
- een verzameling entities, aangeduid met een groepsnaam
- een verzameling entities die je zoekt op andere zoekcriteria

### 2.2.1 De verzameling met alle entities van hetzelfde type

Syntax van de bijbehorende URI: /meervoudsvormentities

Voorbeeld URI	Voorgestelde entities
/brouwers	Alle brouwers
/bieren	Alle bieren

### 2.2.2 Één entity, aangeduid met zijn unieke identificatie

Syntax van de bijbehorende URI: /meervoudsvormentities/uniekeidentificatie

Voorbeeld URI	Voorgestelde entity
/brouwers/3	De brouwer met het nummer 3
/bieren/6	Het bier met het nummer 6

Variabele onderdelen in de URI (3 in de URI /brouwers/3) heten path variabelen.



Opmerking: je stelt de unieke identificatie niet voor als een request parameter (/brouwers?id=3)



Opmerking: sommige websites gebruiken als unieke identificatie data uit de werkelijkheid, bvb. de unieke brouwernaam: /brouwers/liefmans

Voordeel: deze URI's hebben een menselijker betekenis.

Nadeel: als de brouwernaam wijzigt, krijgen de applicaties die nog een request doen naar de oude URI een response met HTTP fout 404 (Not found).

### 2.2.3 Een verzameling entities, aangeduid met een groepsnaam

Soms heeft een verzameling entities een groepsnaam in de werkelijkheid.

Voorbeelden: hobbybrouwers, commerciëlebrouwers.

Syntax URI: /meervoudsvormentities/groepsnaam

Voorbeeld URI	Voorgestelde entities
/brouwers/hobbybrouwers	Alle hobbybrouwers
/brouwers/commerciëlebrouwers	Alle commerciële brouwers

### 2.2.4 Een verzameling entities die je zoekt op andere zoekcriteria

Je zoekt soms een verzameling entities met andere zoekcriteria dan hier boven.

Je gebruikt dan request parameters.

Syntax URI: /meervoudsvormentities?requestparameters...

Voorbeeld URI	Voorgestelde entities
/brouwers?beginnaam=de	Brouwers waarvan de naam begint met de
/bieren?vanafAlcohol=6	Bieren met een alcohol >= 6

Je kan request parameters ook gebruiken om de sorteervolgorde aan te geven

Voorbeeld URI	Voorgestelde entities
/brouwers?sort=Naam	Alle brouwers, gesorteerd op naam
/brouwers?sort=aantalBieren	Alle brouwers, gesorteerd op aantal bieren

### 2.2.5 Associaties tussen entities

Je kan een associatie tussen entities uitdrukken met een URI.

Syntax: /meervoudsvormentities/uniekeidentificatie/naamvandeassociatie

Voorbeeld URI	Voorgestelde entities
/brouwers/3/bieren	De bieren van de brouwer met het nummer 3
/bieren/6/brouwer	De brouwer van het bier met het nummer 6



Opmerking: meerdere URI's kunnen naar dezelfde entity verwijzen.

De URI /bieren/6 én de URI /brouwers/3/bieren/6 verwijzen allebei naar dezelfde entity: het bier met het nummer 6.

### 2.2.6 URI templates

Een URI template stelt een URI met path variabelen voor.

Deze path variabelen staan in de URI template tussen accolades.

Je maakt een URI op basis van een URI template door de path variabelen te vervangen door waarden.

URI template	Voorbeeld bijbehorende URI
/brouwers/{brouwerid}/bieren	/brouwers/3/bieren
/brouwers/{brouwerid}/bieren/{bierid}	/brouwers/2/bieren/6

## 2.3 De client duidt met de standaard HTTP methods de handeling aan die hij op de entities wil uitvoeren.

De client doet een request naar de URI van een resource.

De client duidt met de HTTP method (GET, POST, PUT of DELETE) van de request de handeling aan die hij wil uitvoeren op deze resource.

- GET resource lezen
- POST resource toevoegen
- PUT resource wijzigen
- DELETE resource verwijderen

### 2.3.1 De client voegt een entity toe met HTTP POST

De client doet een POST request naar de URI van een resource.

De request body bevat de data van de toe te voegen entity.

Voorbeelden

- De client doet een POST request naar /brouwers om een brouwer toe te voegen.  
De request body bevat de data van de toe te voegen brouwer.
- De client doet een POST request naar /bieren om een bier toe te voegen.  
De request body bevat de data van het toe te voegen bier.

### 2.3.2 De client leest één of meerdere entities met HTTP GET

De client doet een GET request naar de URI van een resource.

Hij krijgt een response terug met de data van die entity of entities.

Voorbeelden

- De client doet een GET request naar /brouwers om alle brouwers te lezen.
- De client doet een GET request naar /brouwers/3 om brouwer 3 te lezen.



### 2.3.3 De client wijzigt een entity met HTTP PUT

De client doet een PUT request naar de URI van de te wijzigen resource.

De request body bevat de te wijzigen data van de entity.

Voorbeeld

- De client doet een PUT request naar `/brouwers/3` om brouwer 3 te wijzigen.  
De request body bevat de te wijzigen brouwerdata.

### 2.3.4 De client verwijdert een entity met HTTP DELETE

De client doet een DELETE request naar de URI van de te verwijderen resource.

Voorbeeld

- De client doet een DELETE request naar `/brouwers/3` om brouwer 3 te schrappen.

## 2.4 Data uitwisselen in meerdere formaten

De client kan bij een GET request op drie manieren het gewenste formaat (HTML, XML, JSON, ...) aangeven waarmee hij de data in de response wilt ontvangen.

- Het gewenste formaat aangeven in de request header Accept

Request header Accept	Data formaat response
text/html	HTML
application/xml	XML
application/json	JSON

- Het gewenste formaat aangeven in een extensie op de URI

URI extensie	Data formaat response
Geen	HTML
.xml	XML
.json	JSON

- Het gewenste formaat aangeven in een query parameter

Query parameter	Data formaat response
Geen	HTML
Xml	XML
json	JSON

Voorbeeld: de client doet een GET request naar `/brouwers/3` met `Accept:application/xml`

De response bevat de brouwer in XML formaat

```
<?xml version="1.0" encoding="UTF-8"?>
<brouwer id="3">
  <naam>Bavik</naam>
  <postcode>8531</postcode>
  <gemeente>Bavikhove</gemeente>
  <link rel="self" href="http://www.mysite.org/brouwers/3"/>
</brouwer>
```

De response bevat altijd een header Content-type. Die geeft het formaat van de response data aan. Deze header kan bijvoorbeeld `application/xml` bevatten.



Opmerking: de client kan de request header Accept vullen met meerdere waarden, gescheiden door een komma. Voorbeeld: `application/json, application/xml`. Hij geeft daarmee aan dat de service de response in één van deze formaten mag terugsturen. De service kiest dan één van deze formaten. Hij geeft het gekozen formaat aan in de response header Content-type.

Bij een POST request (entity toevoegen) of een PUT request (entity wijzigen), bevat de request body de toe te voegen of te wijzigen resource.

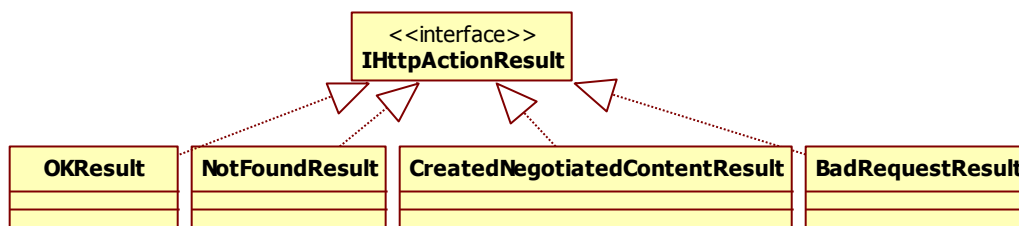
De client geeft dan in de request header Content-Type aan in welk formaat deze resource is uitgedrukt (bvb. application/xml), als de resource is uitgedrukt in XML).

## 2.5 REST gebruikt de response status code om aan te geven of de request wel of niet correct verwerkt werd

De meest gebruikte response status codes

Status code	Korte Betekenis	Betekenis
200	OK	De request is correct verwerkt.
201	Created	De client doet een POST request om een entity toe te voegen. De response header Location bevat de URI van de nieuwe resource.
400	Bad request	De client doet een POST request of een PUT request. De request body bevat verkeerde data (bvb. een bier met een negatieve alcohol). De response body kan meer informatie over de fout bevatten
401	Unauthorized	De client doet een request op een beveiligde URI. De client heeft zich nog niet geauthenticeerd, of heeft een verkeerde gebruikersnaam of paswoord meegegeven.
403	Forbidden	De client doet een request op een beveiligde URI. De client heeft zich correct geauthenticeerd, maar heeft niet de juiste rechten om een request te doen op die URI.
404	Not found	De client doet een request naar een URI die de service niet kent.
405	Method not allowed	De client doet een request op een URI. De service laat de HTTP method van die request niet toe op die URI. Voorbeeld: een DELETE request naar <code>http://voorbeeld.com/brouwers</code> De service laat enkel GET requests toe op deze URI. De response header Allow bevat de lijst met HTTP methods die de URI wél ondersteunt.
406	Not acceptable	De service ondersteunt de waarde in request header Accept niet. Voorbeeld: de request header Accept bevat <code>application/xml</code> . De service kan echter enkel responses in JSON formaat produceren.
409	Conflict	De request probeert een entity in een onmogelijke toestand te plaatsen. Voorbeeld: een DELETE request naar <code>http://voorbeeld.com/brouwers/3</code> Brouwer 3 heeft echter nog bieren en kan daarom niet verwijderd worden.
415	Unsupported Media Type	Een request bevat data in een formaat aangegeven in de header Content-Type. De service ondersteunt dit formaat echter niet. Voorbeeld een POST request bevat data in JSON formaat. De service kan echter enkel data in XML formaat verwerken.
500	Internal server error	De client doet een request. De service heeft interne problemen om die request te verwerken (de database is bijvoorbeeld uitgevallen).

De interface `IHttpActionResult` uit de Web API library stelt een response voor. Meerdere classes implementeren deze interface. Je ziet hier de belangrijkste classes.



- `OKResult` stelt een response voor met status code 200 (OK)
- `NotFoundResult` stelt een response voor met status code 404 (Not Found)
- `CreatedNegotiated...` stelt een response voor met status code 201 (Created)
- `BadRequestResult` stelt een response voor met status code 400 (Bad Request)

## 2.6 REST data bevat URI's naar nieuwe data

De response, die de client krijgt, bevat URI's waarmee hij nieuwe requests kan maken naar de service. De client moet de URI's van deze nieuwe requests niet zelf verzinnen, maar krijgt ze aangereikt.

Voorbeeld:

Een client doet een GET request naar `/brouwers` met `Accept:application/xml`.

De response kan er als volgt uitzien

```
<?xml version="1.0" encoding="UTF-8"?>
<brouwers>
  <brouwer id="1" naam="Achouffe">
    <link rel="self" href="http://www.mysite.org/brouwers/1">
  </brouwer>
  <brouwer id="2" naam="Alken">
    <link rel="self" href="http://www.mysite.org/brouwers/2">
  </brouwer>
  <link rel="self" href="http://www.mysite.org/brouwers">
</brouwers>
```

De client moet de URI om de detail van één van de brouwers op te vragen niet verzinnen, maar krijgt deze URI aangereikt in een element `link`.

Dit stap per stap volgen van URI's in responses heet HATEOAS (hypermedia as the engine of application state).

## 3 DE REST SERVICE

### 3.1 Algemeen

Je maakt een REST service in de gedaante van een website.

Je maakt die website als een ASP.NET MVC 4 project

- Je kiest in het menu FILE, New, Project
- Je kiest links Visual C#, Web en rechts ASP.NET Web Application
- Je tikt BrouwersService bij Name
- Je tikt BrouwersSolution bij Solution name
- Je kiest OK
- Je kiest Web API bij Select a template
- Je kiest OK

Je maakt in het onderdeel Models een class Brouwer

```
namespace BrouwersService.Models
{
    public class Brouwer
    {
        public int ID { get; set; }
        public string Naam { get; set; }
        public int Postcode { get; set; }
        public string Gemeente { get; set; }
    }
}
```

### 3.2 De database

Je zal de data niet uit een echte database halen, maar uit het interne geheugen.

Je maakt daartoe in het onderdeel Models een class InMemoryDataBase

```
using System.Collections.Generic;
namespace BrouwersService.Models
{
    public class InMemoryDataBase
    {
        private static Dictionary<int, Brouwer> brouwersValue;
        static InMemoryDataBase()
        {
            var achouffe = new Brouwer {ID = 1, Naam = "Achouffe", Postcode = 6666,
                Gemeente = "Achouffe"};
            var alken = new Brouwer {ID = 2, Naam = "Alken", Postcode = 3570,
                Gemeente = "Alken"};
            var bavik = new Brouwer {ID = 3, Naam = "Bavik", Postcode = 8531,
                Gemeente = "Bavikhove"};
            brouwersValue = new Dictionary<int, Brouwer> {
                {achouffe.ID, achouffe}, {alken.ID, alken}, {bavik.ID, bavik}};
        }
        public static Dictionary<int, Brouwer> Brouwers
        {
            get { return brouwersValue; }
        }
    }
}
```

### 3.3 Controllers en URI's

Het project bevat een onderdeel Controllers.

Je zal met deze controllers communiceren met de REST clients.

Je maakt één controller per soort gegeven (resource) dat je uitwisselt met de REST client.

In ons voorbeeld is er één resource: brouwers. Je maakt dus één controller.

Je kiest de naam van die controller zorgvuldig, want die naam bepaalt de URL van de resource

- De Web API verwijdt achteraan in de naam het woord Controller.  
BrouwersController wordt dus brouwers
- De Web API voegt vooraan api/ toe.  
brouwers wordt dus api/brouwers

De Web API gebruikt dit resultaat als het begin van de URI van brouwer resources.

Dit valt onder de noemer "Configuration by convention".

Je maakt de controller

- Je klikt in de Solution Explorer met de rechtermuisknop op Controllers
- Je kiest Add, Controller
- Je kiest Web API 2 Controller - Empty en je kiest Add
- Je tikt BrouwersController bij Controller Name
- Je kiest Add.

Je ziet dat een API controller een class is die erft van ApiController.

## 4 GET REQUESTS: DATA LEZEN

### 4.1 Algemeen

De client doet op een URL een HTTP request met de method GET om de resource(s) op die URL te lezen.

### 4.2 Een GET request naar een verzameling

Je maakt in BrowsersController een method die GET requests naar /Browsers verwerkt en als response de verzameling met alle browsers teruggeeft.

```
public IActionResult GetAll() (1)
{
    return this.Ok(InMemoryDataBase.Browsers.Values.ToList()); (2)
}
```

- (1) De returnwaarde van de method is IActionResult. Zoals eerder vermeld in de cursus stelt die interface de response van een Http request voor. De methodnaam begint met Get Web API associeert een method zonder parameters, met een naam die begint met Get, met een GET request naar de URI die bij de huidige Controller hoort (api/browsers). Dit valt terug onder "Configuration by convention".
- (2) Je haalt alle browsers op en geeft die door aan de method Ok. Je erft die method van je base class ApiController. De method geeft een object van het type OKResult terug. Zoals eerder vermeld in de cursus implementeert de class OKResult de interface IActionResult en stelt een response voor met status code 200 (OK). De method Ok vult de response body met de data die je meegeeft als zijn parameter: de browsers.

### 4.3 De service uittesten

Je start de website. Je laat de browser openstaan, zodat de website blijft draaien.

Je zal later in de cursus leren hoe je een REST client schrijft.

Je gebruikt voorlopig een REST client waarmee je als ontwikkelaar REST services kan uitproberen: Fiddler. Je downloadt deze tool op <http://www.telerik.com/download/fiddler>

De tool bestaat uit een linker deel en een rechter deel.

- In het linker deel zie je een lijst met HTTP requests die je computer doet. Zodra Fiddler opstart, doet hij zelf enkele requests om te controleren of er een nieuwe versie van de tool beschikbaar is. Daarom bevat de lijst met HTTP requests al item(s):



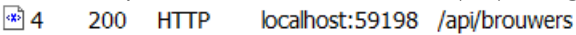

Result	Protocol	Host	URL
200	HTTP	www.fiddler2.com	/fiddler2/updatecheck.asp...

De kolom Result bevat de status code van de response (200=OK).

Je kan deze lijst leeg maken via rechtermuisknop, Remove, All Sessions.

- In het rechter deel kan je zelf requests versturen en de bijbehorende responses inzien.
  - Je kiest in het rechter deel het tabblad Composer.
  - Je ziet dat het rechter deel een uitvallijst bevat met HTTP methods. De HTTP method GET is geselecteerd: GET
  - Je tikt daarnaast de request URI `http://localhost:59198/api/browsers` Misschien draait de Visual Studio webserver bij jou niet op poort 59198. Je kan het poort nummer zien in de adresbalk van de browser die geopend werd bij het starten van de service host.
  - Je klikt op de knop Execute of je drukt Enter.
  - Je ziet in het linker deel dat Fiddler deze request verstuurd en een response met HTTP code 200 (OK) terug kreeg:

#	Result	Prot...	Host	URL
1	200	HTTP	localhost:59198	/api/browsers

- Voorin in deze regel zie je . Dit geeft aan dat de response body data bevat in het default formaat van de Web API: JSON.
- Je dubbelklikt deze request in het linker deel.
- Je ziet in het rechterdeel de brouwers in een tabblad JSON.
- Je kiest in het rechterdeel het tabblad Raw.  
Je ziet daar alle detail van de response (ook de headers).
- Je klikt in het rechterdeel het tabblad Composer
- Je tikt in het tekstvak onder , onder de bestaande request headers, `accept: application/xml`  
Je geeft daarmee aan dat je een response verwacht in XML formaat.
- Je kiest Execute om de request te versturen.
- Je ziet in het linker deel dat Fiddler deze request verstuurd en een response met HTTP code 200 (OK) terug kreeg:  

- Voorin in deze regel zie je . Dit geeft aan dat de response body data bevat in XML formaat.
- Je dubbelklikt deze request in het linker deel.
- Je ziet in het rechterdeel de brouwers in een tabblad XML.
- Je kiest in het rechterdeel het tabblad Raw. Je ziet daar alle detail van de response.

#### 4.4 Een GET request naar één resource aan de hand van zijn unieke identificatie

Een GET request naar `/api/brouwers/1` moet brouwer 1 teruggeven.

De veranderlijke waarde 1 in de URI heet een path variable.

Je voegt volgende method toe aan `BrouwersController`

```
public IActionResult Get(int id) (1)
{
    if (InMemoryDataBase.Brouwers.ContainsKey(id)) (2)
    {
        return this.Ok(InMemoryDataBase.Brouwers[id]); (3)
    }
    return this.NotFound(); (4)
}
```

- (1) Web API associeert een method, waarvan de naam begint met `Get` en een parameter met de naam `id` heeft, met een GET request naar de URI die bij de huidige Controller hoort en één path variabele heeft (`api/brouwers/1`). Als zo'n request binnenkomt roept de Web API deze method op en vult de parameter `id` met 1.
- (2) Je controleert of een brouwer met deze `id` voorkomt in de database.
- (3) Je geeft die brouwer terug.
- (4) Je hebt de brouwer niet gevonden in de database. Je roept de method `NotFound` op. Je erft die method van je base class `ApiController`. De method geeft een object van het type `NotFoundResult` terug. Zoals eerder vermeld in de cursus implementeert de class `NotFoundResult` de interface `IActionResult` en stelt een response voor met status code 404 (Not Found).

Je test dit uit

- Je start de applicatie.
- Je zoekt met Fiddler een bestaande brouwer (`/api/brouwers/1`).  
Dit geeft een response met status code 200 (OK) en een body met data over brouwer 1.
- Je zoekt een niet-bestaande brouwer (`/api/brouwers/666`).  
Dit geeft een response met status code 404 (Not Found) en een lege body.

## 4.5 Een GET request naar resources aan de hand van andere criteria dan de unieke identificatie

Een GET request naar `/api/brouwers?beginnaam=a` moet een lijst van brouwers teruggeven waarvan de naam begint met a. `beginnaam` in deze URI heet een query parameter

Je voegt volgende method toe aan `BrouwersController`

```
public IHttpActionResult GetByBeginNaam(string beginNaam)
{
    beginNaam = beginNaam.ToLower();
    return Ok((from brouwer in InMemoryDataBase.Brouwers.Values
                where brouwer.Naam.ToLower().StartsWith(beginNaam)
                orderby brouwer.Naam
                select brouwer).ToList());
}
```

- (1) De Web API associeert een method, waarvan de naam begint met `Get` en een parameter met een naam verschillend van `id`, met een GET request naar de URI die bij de huidige Controller hoort en een request parameter heeft met dezelfde naam als de method parameter (`api/brouwers?beginnaam=a`). Als zo'n request binnenkomt roept de Web API deze method op en vult de parameter `beginnaam` met a.
- (2) Je zoekt met een LINQ query de brouwers met de gevraagde `beginnaam`. Je geeft het resultaat aan de method `Ok`. De Web API stuurt dit resultaat in JSON of XML formaat naar de browser.

Je kan met Fiddler brouwers opzoeken (`/api/brouwers?beginnaam=a`)



Opmerking: een request kan meerdere query parameters bevatten

Voorbeeld: `/api/brouwers?vanpostcode=8000&totpostcode=8999`

Je geeft de controller method die deze request verwerkt dan meerdere parameters

```
public IHttpActionResult GetByPostcodeBetween(int vanpostcode,
int totpostcode)
```

## 4.6 Namen van XML elementen en namespace

Bij een request doet naar `/brouwers/3`, met als request Accept header `application/xml`, krijg je volgende response:

```
<?xml version="1.0" encoding="UTF-8"?>
<Brouwer xmlns="http://schemas.datacontract.org/2004/07/BrouwersService"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <Gemeente>Bavikhove</Gemeente>
  <ID>3</ID>
  <Naam>Bavik</Naam>
  <Postcode>8531</Postcode>
</Brouwer>
```

- De namen van de elementen zijn gelijk aan de namen van de bijbehorende class (bvb. `Brouwer`) of de bijbehorende property (bvb. `ID`).
- De Web API heeft zelf een namespace verzonnen (bij `xmlns`)

Je kan beide aspecten aanpassen in de entity class.

Je voegt eerst aan je project een reference toe naar `System.Runtime.Serialization`

Je wijzigt de class `Brouwer`:

```
[DataContract(Name="brouwer", Namespace="")] (1)
```

```
public class Brouwer
```

```
{ (2)
```

```
    [DataMember(Name="id")]
    public int ID { get; set; }
```

```
    [DataMember(Name="naam")]
    public string Naam { get; set; }
```



```
[DataMember(Name="postcode")]
public int Postcode { get; set; }
[DataMember(Name="gemeente")]
public string Gemeente { get; set; }
...
```

- (1) De naam van het XML element dat een brouwer voorstelt, wordt brouwer.  
XML element namen bevatten meestal kleine letters.  
REST houdt de XML in requests en responses kort om de leesbaarheid te bevorderen en gebruikt daarom zelden XML namespaces.
- (2) De naam van het XML element dat ID voorstelt wordt id.

Bij een request naar /brouwers/3 krijg je volgende response:  
(ietwat jammer is dat je het attribuut xmlns:i niet kan weglaten)

```
<?xml version="1.0" encoding="UTF-8"?>
<brouwer xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <gemeente>Bavikhove</gemeente>
  <id>3</id>
  <naam>Bavik</naam>
  <postcode>8531</postcode>
</brouwer>
```

De child elementen van brouwer (gemeente, naam, ...) komen nu in alfabetische volgorde.  
Je kan de volgorde zelf definiëren in de class Brouwer.

Je wijzigt een deel van de class Brouwer:

```
[DataMember(Name="id", Order=1)]
public int ID { get; set; }
[DataMember(Name="naam", Order=2)]
public string Naam { get; set; }
[DataMember(Name="postcode", Order=3)]
public int Postcode { get; set; }
[DataMember(Name="gemeente", Order=4)]
public string Gemeente { get; set; }
```

(1)

- (1) Het element id zal nu het eerste child element zijn (Order:=1)

Bij een request naar /brouwers/3, krijg je nu volgende response

```
<?xml version="1.0" encoding="UTF-8"?>
<brouwer xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <id>3</id>
  <naam>Bavik</naam>
  <postcode>8531</postcode>
  <gemeente>Bavikhove</gemeente>
</brouwer>
```

## 4.7 Namen van XML elementen die verzameling voorstellen

Als de client een GET request doet naar /brouwers,  
is de naam van het eerste XML element in de response ArrayOfbrouwer.

Dit element stelt een verzameling brouwers voor.

Om de naam van dit element te wijzigen, maak je in het onderdeel Models een class Brouwers.  
Deze class stelt niet één brouwer voor, maar een verzameling brouwers:

```
using System.Collections.Generic;
using System.Runtime.Serialization;
namespace BrouwersService.Models
{
    [CollectionDataContract(Name="brouwers", Namespace= "")]
    public class Brouwers:List<Brouwer>
    {
    }
}
```

(1)  
(2)

- (1) Je gebruikt bij een class die een verzameling objecten voorstelt het attribuut `CollectionDataContract` in plaats van het attribuut `DataContract`.  
Je geeft bij `Name` de naam mee van het bijbehorende XML element.  
Je geeft bij `Namespace` de bijbehorende XML namespace mee.
- (2) Deze class erft van `List<Brouwer>` en stelt dus een lijst van brouwers voor.

Je wijzigt in de class `BrouwersController` de code in de method `GetAll`

```
var brouwers = new Brouwers();
brouwers.AddRange(InMemoryDataBase.Brouwers.Values);
return this.Ok(brouwers);
```

en de code in de method `GetByBeginNaam`

```
beginNaam = beginNaam.ToLower();
var brouwers = new Brouwers();
brouwers.AddRange(from brouwer in InMemoryDataBase.Brouwers.Values
                  where brouwer.Naam.ToLower().StartsWith(beginNaam)
                  orderby brouwer.Naam
                  select brouwer);
return Ok(brouwers);
```

Bij een GET request naar `/brouwers`, begint de response nu met een element `brouwers`.



Opmerking: met `DataContract`, `DataElement` en `CollectionDataContract` wijzig je niet alleen de namen van een response uitgedrukt in XML, maar ook van een response uitgedrukt in JSON

## 4.8 Responses met hyperlinks

Sommige responses bevatten niet één brouwer, maar een verzameling brouwers.

Deze response bevat per brouwer alle properties.

Als een class veel properties heeft, krijg je op die manier omvangrijke responses.

Dit is niet goed voor de performantie van de service.

Een veel gebruikte oplossing is per brouwer enkel de essentiële properties op te nemen

(bvb. id en naam). De response bevat daarnaast per brouwer een hyperlink met de naam `Detail`.

Deze hyperlink bevat de URI van de brouwer. Als de client alle properties van die brouwer wenst te lezen, doet hij een request naar die URI.

Voorbeeld:

```
<brouwers xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <brouwer>
    <nummer>3</nummer>
    <naam>Bavik</naam>
    <detail>http://localhost:52001/brouwers/3</detail>
  </brouwer>
  ...
</brouwers>
```

Je voegt aan het onderdeel `Models` een class `BrouwerBeknopt` toe

```
using System.Runtime.Serialization;
namespace BrouwersService.Models
{
    [DataContract(Name = "brouwer", Namespace = "")]
    public class BrouwerBeknopt
    {
        [DataMember(Name = "id", Order = 1)]
        public int ID { get; set; }
        [DataMember(Name = "naam", Order = 2)]
        public string Naam { get; set; }
        [DataMember(Name = "detail", Order = 3)]
        public string Detail { get; set; }
    }
}
```

Je wijzigt in de class Brouwers `List<Brouwer>` naar `List<BrouwerBeknopt>`

Je wijzigt methods in de class BrouwersController:

Je wijzigt de tweede opdracht in de method GetAll

```
var detail = this.Request.RequestUri.AbsoluteUri + "/"; (1)
```

```
brouwers.AddRange(from brouwer in InMemoryDataBase.Brouwers.Values
    orderby brouwer.Naam
    select new BrouwerBeknopt
    {
        ID = brouwer.ID,
        Naam = brouwer.Naam,
        Detail = detail + brouwer.ID (2)
    });
```

- (1) Je erft van je base class een property Request.  
Deze bevat gegevens over de huidige request.
- (2) De expressie request.RequestUri.AbsoluteUri geeft je de URL van de huidige request  
(bijvoorbeeld `http://localhost:59198/api/brouwers`)
- (3) Je vult de property Detail met de URI van de huidige brouwer  
(bij brouwer 1 is dit bijvoorbeeld `http://localhost:59198/api/brouwers/1`)

Je wijzigt ook de derde opdracht in method GetBrouwersByBeginNaam

```
var detail = this.Request.RequestUri.AbsoluteUri; (1)
```

```
detail = detail.Substring(0, detail.IndexOf('?')); (2)
```

```
detail += "/";
```

```
brouwers.AddRange(from brouwer in InMemoryDataBase.Brouwers.Values
    where brouwer.Naam.ToLower().StartsWith(beginNaam)
    orderby brouwer.Naam
    select new BrouwerBeknopt
    {
        ID = brouwer.ID,
        Naam = brouwer.Naam,
        Detail = detail + brouwer.ID
    });
```

- (1) Je vraagt de URL van de huidige request.  
Die is bijvoorbeeld `http://localhost:59198/api/brouwers?beginnaam=a`
- (2) Je verwijdert uit deze URL de tekens vanaf het teken ?

Bij een request naar `/brouwers` krijg je nu een response met per brouwer beknopte informatie (nummer en naam) én een URI waarop je een nieuwe request kan doen om detailinformatie van die brouwer te weten.

Je kan dit uitproberen met Fiddler.



Kazen service lees operaties: zie takenbundel

## 5 REST SERVICES: ANDERE OPERATIES

### 5.1 DELETE requests: data verwijderen

De client doet een request met de method DELETE naar een URI om de resource te verwijderen die de URI voorstelt.

Je voegt aan de class BrouwersController een method toe die de request verwerkt.

```
public IHttpActionResult Delete(int id) (1)
{
    if (InMemoryDataBase.Brouwers.ContainsKey(id))
    {
        InMemoryDataBase.Brouwers.Remove(id); (2)
        return this.Ok();
    }
    return this.NotFound();
}
```

- (1) De Web API associeert een method, waarvan de naam begint met Delete en een parameter met de naam id heeft, met een DELETE request naar de URI die bij de huidige Controller hoort en één path variabele heeft (api/brouwers/1).
- (2) Je verwijdert de brouwer uit de database.

Je kan het verwijderen testen met Fiddler.

- Je kiest rechts het tabblad Composer
- Je wijzigt in de uitvallijst GET naar DELETE
- Je doet een request naar /brouwers/1

### 5.2 POST request: data toevoegen

De client doet een request met de method POST naar een URI die een verzameling resources voorstelt om een resource aan die verzameling toe te voegen.

Een voorbeeld is een POST request naar /brouwers om een brouwer toe te voegen.

De request body bevat de properties van de toe te voegen resource.

- Als deze properties zijn uitgedrukt in XML, plaatst de client de request header Content-type op application/xml.
- Als deze properties zijn uitgedrukt in JSON, plaatst de client de request header Content-type op application/json.

De service stuurt een response terug.

- Als de service de resource met succes heeft toegevoegd, bevat de status code de waarde 201 (Created).  
De response header Location bevat de URI van de nieuwe resource.
- Als properties van de toe te voegen resource verkeerde waarden bevatten, bevat de status code de waarde 400 (Bad request)  
De response body kan meer informatie bevatten over de fout.

Je voegt aan de class BrouwersController een method toe die de request verwerkt.

```
public IHttpActionResult Post(Brouwer brouwer) (1)
{
    var id = InMemoryDataBase.Brouwers.Keys.Max() + 1; (2)
    brouwer.ID = id; (3)
    InMemoryDataBase.Brouwers[id] = brouwer; (4)
    return this.Created(this.Request.RequestUri.AbsoluteUri + "/" + id, brouwer); (5)
}
```

- (1) De Web API associeert een method, waarvan de naam begint met Post, met een POST request naar de URI die bij de huidige Controller hoort.  
Je voegt aan de method die een POST request verwerkt, een parameter toe die de toe te voegen entity voorstelt. De Web API vult de properties van deze parameter met de data die de client in de body van de request plaatste.
- (2) Je bepaalt het nummer voor de nieuwe brouwer:  
het hoogste nummer van de bestaande brouwers plus één.
- (3) Je vult dit nummer in bij de nieuwe brouwer.
- (4) Je voegt de nieuwe brouwer toe aan de verzameling brouwers.
- (5) Je roept de method Created op. Je erft die method van je base class ApiController.  
De method geeft een object van het type CreatedNegotiatedContentResult terug.  
Zoals eerder vermeld in de cursus implementeert de class CreatedNegotiatedContentResult de interface IHttpActionResult en stelt een response voor met status code 201 (Created).  
Je geeft als eerste parameter de URI van de nieuwe brouwer mee.  
De method Created vult hiermee de response header Location.  
Je geeft als tweede parameter de toegevoegde brouwer mee.  
De method Created vult hiermee de response body

Je kan het toevoegen testen met Fiddler.

- Je kiest rechts het tabblad Composer.
- Je wijzigt in de uitvallijst GET naar POST.
- Je tikt bij Request Headers een extra header: *Content-type:application/xml*
- Je tikt bij Request Body (onder Request Header) volgende request body:

```
<brouwer>
<naam>Bios</naam>
<postcode>9940</postcode>
<gemeente>Ertvelde</gemeente>
</brouwer>
```

- Je tikt `http://localhost:52001/api/brouwers` naast POST en je kiest Execute
- Je dubbelklikt deze request in het linker deel.
- Je ziet in het rechter deel in het tabblad Headers
  - dat de response de Status 201 (Created) heeft.
  - dat de response een header Location bevat met de URI van de nieuwe brouwer: `localhost:52001/brouwers/4`.
- De response body bevat de toegevoegde brouwer.

### 5.3 PUT request: data wijzigen todo tot hier

De client doet een request met de method PUT naar een URI om de resource te wijzigen die de URI voorstelt.

Een voorbeeld is een PUT request naar `/brouwers/2` om brouwer 2 te wijzigen

De request body bevat de properties van de te wijzigen resource.

- Als deze properties zijn uitgedrukt in XML, plaatst de client de request header *Content-type* op *application/xml*.
- Als deze properties zijn uitgedrukt in JSON, plaatst de client de request header *Content-type* op *application/json*.

De service stuurt een response terug.

- Als de resource met succes gewijzigd is, bevat de status code 204 (No Content).
- Als de te wijzigen resource niet bestaat bevat de status code 404 (Not Found)
- Als properties van de toe te voegen resource verkeerde waarden bevatten, bevat de status code de waarde 400 (Bad request)  
De response body kan meer informatie bevatten over de fout.

Je voegt aan de class BrouwersController een method toe die de request verwerkt.

```
public IActionResult Put(int id, Brouwer brouwer)           (1)
{
    if (InMemoryDataBase.Brouwers.ContainsKey(id))          (2)
    {
        InMemoryDataBase.Brouwers[id] = brouwer;           (3)
        return this.Ok();
    }
    return this.NotFound();                                   (4)
}
```

- (1) De Web API associeert een method, waarvan de naam begint met Put en een parameter met de naam id heeft, met een Put request naar de URI die bij de huidige Controller hoort en één path variabele heeft (api/brouwers/1).  
De Web API vult de parameter id met de inhoud van die path variabele.  
Je voegt aan de method, een parameter toe die de te wijzigen entity voorstelt.  
De Web API vult de properties van deze parameter met de data die de client in de body van de request plaatste.
- (2) Je zoekt de te wijzigen brouwer
- (3) Je wijzigt de brouwer
- (4) Je hebt de brouwer niet gevonden in de database.  
Je maakt response met status code 404 (Not Found).

Je kan het wijzigen testen met Fiddler:

- Je kiest rechts het tabblad Composer.
- Je wijzigt in de uitvallijst GET naar PUT.
- Je tikt daar onder een extra header: Content-type:application/xml
- Je tikt daar onder (bij Request Body) volgende request body:

```
<brouwer>
<id>2</id>
<naam>Alken</naam>
<postcode>1000</postcode>
<gemeente>Brussel</gemeente>
</brouwer>
```

- Je tikt http://localhost:52001/api/brouwers/2 naast PUT en je kiest Execute.
- Je ziet in het linker deel dat de response de status 204 (No Content) heeft.
- Als je daarna een GET request doet naar http://localhost:52001/api/brouwers/2, zie je dat de postcode en gemeente van brouwer 2 inderdaad gewijzigd zijn.

## 5.4 Validaties

### 5.4.1 Algemeen

Bij een POST request stuurt de client data over een nieuwe resource mee.

Bij een PUT request stuurt de client data over de te wijzigen resource mee.

De Web API bevat voorzieningen om deze data te valideren

(de naam van een brouwer is bijvoorbeeld verplicht mee te geven).

Het valideren gebeurt in drie stappen

- Met validatie attributen aangeven wat er moet gevalideerd worden
- De validatie zelf
- Een response met foutinformatie naar de browser sturen

### 5.4.2 Met validatie attributen aangeven wat er moet gevalideerd worden

Je tikt voor iedere te controleren property van een entity een validatie attribuut. Je geeft met dit validatie attribuut aan hoe je die property wil valideren.

Je wijzigt zo de class Brouwer

```
using System.Runtime.Serialization;
using System.ComponentModel.DataAnnotations;
namespace BrouwersService.Models
{
    [DataContract(Name="brouwer", Namespace="")]
    public class Brouwer
    {
        [DataMember(Name = "id", Order = 1)]
        public int ID { get; set; }
        [Required] (1)
        [StringLength(255, MinimumLength=1)] (2)
        [DataMember(Name = "naam", Order = 2)]
        public string Naam { get; set; }
        [Range(1000,9999)]
        [DataMember(Name = "postcode", Order = 3)] (3)
        public int Postcode { get; set; }
        [Required]
        [StringLength(255, MinimumLength=1)]
        [DataMember(Name = "gemeente", Order = 4)]
        public string Gemeente { get; set; }
    }
}
```

- (1) Je tikt het attribuut Required voor de property Naam.  
Je geeft daarmee aan dat de property Naam verplicht in te vullen is.
- (2) Je tikt ook het attribuut StringLength voor de property Naam.  
De eerste parameter is het maximaal aantal tekens.  
De named parameter MinimumLength is het minimaal aantal tekens.  
Je geeft aan dat de inhoud van de property tussen 1 en 255 tekens moet bevatten.
- (3) Je tikt het attribuut Range voor de property Postcode.  
De eerste parameter is een ondergrens. De tweede parameter is een bovengrens.  
Je geeft daarmee aan dat de inhoud van de property Postcode moet liggen tussen 1000 en 9999.

Naast Required, StringLength en Range bestaat nog het validatie attribuut RegularExpression. Je geeft daarmee aan dat de inhoud van een property moet passen bij het tekstpatroon dat je meegeeft als parameter van RegularExpression.

Voorbeeld:

```
[RegularExpression(@"\d[3]-\d[7]-\d[2]")]
public string BankRekeningNummer { get; set; }
```

Je geeft hiermee aan dat de inhoud van de property BankRekeningNummer moet bestaan uit 3 cijfers (\d[3]), het min teken, 7 cijfers, het min teken en 2 cijfers.

### 5.4.3 De validatie zelf

Bij een PUT of POST request bevat de request body data.

De Web API controleert automatisch iedere property in de data ten opzichte van de bijbehorende validatie attributen.

De Web API plaatst de property ModelState.IsValid van je Controller op false als er validatiefouten zijn. Anders is deze property gelijk aan true.

#### 5.4.4 Een response met foutinformatie naar de browser sturen

Als er validatiefouten zijn, stuur je een response met status code 400 (Bad Request) naar de browser.

Je tikt in de class BrouwersController volgende regels als eerste in de method PostBrouwer en de method PutBrouwer

```
if (! this.ModelState.IsValid)
{
    return this.BadRequest();
}
```

(1)

- (1) Je roept de method BadRequest op. Je erft die method van je base class ApiController. De method geeft een object van het type BadRequestResult terug. Zoals eerder vermeld in de cursus implementeert de class BadRequestResult de interface IActionResult en stelt een response voor met status code 400 (Bad Request).

Je kan dit uitproberen met Fiddler: je wijzigt filiaal 2 met een negatieve postcode

#### 5.4.5 Uitgebreidere foutinformatie aanbieden aan de client

De foutinformatie is tot nu minimaal: een status code 400.

Je zal nu code toevoegen zodat de response body meer detailinformatie over de verkeerde properties bevat.

Je vervangt in de methods PostBrouwer en PutBrouwer

```
return this.BadRequest();
```

door

```
return this.BadRequest(this.ModelState);
```

(1)

- (1) Je geeft de ModelState mee als parameter. De ModelState bevat alle informatie over de validatiefouten. De method BadRequest plaatst die informatie in XML of JSON formaat in de response body.

Je kan dit uitproberen met Fiddler: je wijzigt filiaal 2 met een negatieve postcode



## 5.5 Automatische online documentatie van je REST service

Je REST service bevat automatisch online documentatie

Je start de website en je kiest in de menu rechts boven API

Application name Home API

Je ziet een pagina met een onderdeel Brouwers

### Brouwers

API	Description
<a href="#">GET api/Brouwers</a>	No documentation available.
<a href="#">GET api/Brouwers/{id}</a>	No documentation available.
<a href="#">DELETE api/Brouwers/{id}</a>	No documentation available.
<a href="#">POST api/Brouwers</a>	No documentation available.
<a href="#">GET api/Brouwers?beginNaam={beginNaam}</a>	No documentation available.
<a href="#">PUT api/Brouwers/{id}</a>	No documentation available.

Je kan de zin "No documentation available" wijzigen met speciale commentaar voor methods van je Controllers

Je tikt voor de method GetAll

```
/// <summary>  
/// Alle brouwers lezen  
/// </summary>
```

Je tikt voor de method Get

```
/// <summary>  
/// Een brouwer lezen  
/// </summary>  
/// <param name="id">De id van de te lezen brouwer</param>  
/// <returns>De brouwer</returns>
```

Je moet voor het ganse project nog volgende stappen doen

- Je klikt met de rechtermuisknop op je project in de Solution Explorer en je kiest Properties
- Je kiest links Build
- Je plaatst een vinkje bij XML documentation file en je tikt daarnaast App\_Data/XmlDocument.xml
- Je opent in het project onderdeel Areas, HelpPage, App\_Start de source HelpPageConfig.cs
- Je haalt volgende regel uit commentaar  
`config.SetDocumentationProvider(  
new XmlDocumentationProvider(  
HttpContext.Current.Server.MapPath("~/App_Data/XmlDocument.xml")));`

Je start de website en je kiest in de menu rechts boven API. Je ziet nu

## Brouwers

API	Description
<a href="#">GET api/Brouwers</a>	Alle brouwers lezen
<a href="#">GET api/Brouwers/{id}</a>	Een brouwer lezen

Je klikt op [GET api/Brouwers/{id}](#) en je ziet

## GET api/Brouwers/{id}

Een brouwer lezen

### Request Information

#### URI Parameters

Name	Description	Type	Additional information
id	De id van de te lezen brouwer	integer	Required



Kazen service andere operaties: zie takenbundel

## 6 AFWIJKEN VAN CONFIGURATION BY CONVENTION

### 6.1 Algemeen

De REST service is nu gebaseerd op configuration by convention

- Het begin van de naam van de controller class bepaalt een stuk van de URL waarop die controller requests verwerkt (**BrouwersController** naar de URL **/api/brouwers**)
- Het begin van de naam van een method in een controller bepaalt de HTTP method van de request die de controller method verwerkt (**Get()** verwerkt een GET request)

Je leert hier hoe je kan afwijken van deze conventies en toch een werkende REST service schrijft.

### 6.2 De naam van een method in een controller class

Een controller method waarvan de naam niet begint met Get, Post, Put of Delete kan toch een request verwerken. Je geeft dan met een C# attribuut, dat je voor de method tikt, de HTTP method aan die de controller method verwerkt.

Je wijzigt als eerste voorbeeld het begin van de method `public IActionResult Get()`

```
[HttpGet]                                     (1)
public IActionResult Allen()                  (2)
...
```

- (1) Je geeft met het attribuut `HttpGet` expliciet aan dat de method die er op volgt GET requests verwerkt, zelfs al begint de methodnaam niet met Get.
- (2) Je kan de methodnaam nu vrij kiezen

Als je met Fiddler een GET request doet naar `://localhost:52001/api/brouwers` voert de Web de method `Allen()` uit en geeft een response met alle brouwers terug.

Je wijzigt op dezelfde manier het begin van de andere methods in de controller

oud begin van de method	nieuw begin van de method
<code>public IActionResult Get(int id)</code>	<code>[HttpGet]</code> <code>public IActionResult Een(int id)</code>
<code>public IActionResult Delete(int id)</code>	<code>[HttpDelete]</code> <code>public IActionResult Verwijder(int id)</code>
<code>public IActionResult Post(Brouwer brouwer)</code>	<code>[HttpPost]</code> <code>public IActionResult VoegToe(Brouwer brouwer)</code>
<code>public IActionResult GetByBeginNaam(string beginNaam)</code>	<code>[HttpGet]</code> <code>public IActionResult ByBeginNaam(string beginNaam)</code>
<code>public IActionResult Put(int id, Brouwer brouwer)</code>	<code>[HttpPut]</code> <code>public IActionResult Wijzig(int id, Brouwer brouwer)</code>

### 6.3 De naam van de controller class

De URL waarop een method van een controller class requests verwerkt, hoeft niet bepaald te worden door de naam van de controller class, maar kan je ook bepalen met het C# attribuut `Route`

Je tikt dit attribuut voor de method `Allen()`

```
[Route("brewers")]                             (1)
```

- (1) Je geeft aan dat de method `Allen()` requests verwerkt op de URL `brewers`, en niet op de standaard URL van de controller (`api/brouwers`)

Als je met Fiddler een GET request doet naar `://localhost:52001/brewers` voert de Web de method `Allen()` uit en geeft een response met alle brouwers terug.

Je tikt voor de method Een

```
[Route("brewers/{id}")] (1)
```

- (1) Dit is een URI template met een path variabele id. Een path variabele staat tussen accolades en heeft dezelfde naam als een parameter van de method waarvoor je Route tikt. Als een request binnenkomt naar brewers/7, bevat de parameter id de waarde 7.



Opmerking: een URI template kan meerdere path variabelen hebben, bijvoorbeeld statistieken/{jaar}/{maand}. Je geeft de method, waarvoor je Route met zo'n URI template schrijft, evenveel bijbehorende method parameters:

```
[Route("statistieken/{jaar}/{maand}")]  
public IHttpActionResult StatistiekPerJaarPerMaand(int jaar, int maand)
```

Je tikt voor de methods Verwijder, VoegToe, ByBeginNaam en Wijzig

```
[Route("brewers")] (1)
```

- (1) De method ByBeginNaam verwerkt een query parameter beginnaam (bvb. een request naar /brewers?beginnaam=a). Je moet een query parameter niet moet vermelden bij Route.

Je kan dit uittesten met Fiddler.

Het is vervelend dat je in ieder Route attribuut eenzelfde URL begin (namelijk brewers) herhaalt.

Je vermijdt dit met het attribuut RoutePrefix.

Je tikt juist voor de class BrouwersController

```
[RoutePrefix("brewers")] (1)
```

- (1) De Web API zal het woord hier vermeld (brewers) gebruiken als begin van de URI in de Route attributen die je bij elke method tikte.

Je kan bij de methods Allen, Verwijder, VoegToe, ByBeginNaam en Wijzig

```
[Route("brewers")]
```

nu vervangen door

```
[Route]
```

Je kan bij de methods Een

```
[Route("brewers/{id}")]
```

nu vervangen door

```
[Route("{id}")] (1)
```

- (1) Als bij een class RoutePrefix staat en bij een method Route, mag je de parameter van Route niet beginnen met /. De Web API voegt / zelf in tussen de parameter van RoutePrefix (brewers) en de parameter van Route ({id}), zodat de volledige URI template brewers/{id} wordt.

## 7 REST CLIENTS

### 7.1 Algemeen

Je leert in dit hoofdstuk een REST client maken.

Je voegt aan de solution BrouwersServiceSolution een Console Application project toe met de naam BrouwersClient. Je zal met die client de brouwer service die je al hebt oproepen.

Je hebt voor een REST client enkele classes nodig die niet behoren tot de standaard libraries.

Je installeert deze met de NuGet Package Manager. Je kan met deze tool extra libraries (bovenop de standaard libraries) van het internet downloaden en installeren.

- Je kiest in het menu TOOLS de opdracht NuGet Package Manager en daarna de opdracht Manage NuGet Packages for Solution
- Je tikt in het zoek tekstvak rechts boven microsoft.aspnet.webapi.client en je drukt Enter
- Je kiest Install bij de library Microsoft ASP.NET Web API 2.2 Client Libraries
- Je verwijdert het vinkje bij het project BrouwersService en je kiest OK
- Je kiest I Accept
- Je kiest Close

Je voegt aan het project references toe naar

- System.Runtime.Serialization.
- System.Net.Http

### 7.2 Data uit requests en responses

- Als een client een GET request doet, krijgt hij een response terug met data.
- Als een client een POST of PUT request doet, plaatst hij data in de request.

Je definieert deze data op dezelfde manier als in een REST service: als classes.

Je voegt aan het project de classes Brouwers en BrouwerBeknopt toe.

Deze classes stellen de data voor in de responses van de GET requests naar

- /brouwers/
- /brouwers?beginnaam={beginNaam}

De class BrouwerBeknopt:

```
using System.Runtime.Serialization;
namespace BrouwersClient
{
    [DataContract(Name = "brouwer", Namespace = "")]
    public class BrouwerBeknopt
    {
        [DataMember(Name = "id", Order = 1)]
        public int ID { get; set; }
        [DataMember(Name = "naam", Order = 2)]
        public string Naam { get; set; }
        [DataMember(Name = "detail", Order = 3)]
        public string Detail { get; set; }
    }
}
```

De class Brouwers:

```
using System.Runtime.Serialization;
using System.Collections.Generic;
namespace BrouwersClient
{
    [CollectionDataContract(Name = "brouwers", Namespace = "")]
    public class Brouwers : List<BrouwerBeknopt> {}
}
```

Je voegt aan het project de class Brouwer toe.

Deze class stelt de data voor in

- de responses van de GET requests naar /brouwers/{id}
- de POST requests naar /brouwers/
- de PUT requests naar /brouwers/{id}

```
using System.Runtime.Serialization;
namespace BrouwersClient
{
    [DataContract(Name = "brouwer", Namespace = "")]
    public class Brouwer
    {
        [DataMember(Name = "id", Order = 1)]
        public int ID { get; set; }
        [DataMember(Name = "naam", Order = 2)]
        public string Naam { get; set; }
        [DataMember(Name = "postcode", Order = 3)]
        public int Postcode { get; set; }
        [DataMember(Name = "gemeente", Order = 4)]
        public string Gemeente { get; set; }
    }
}
```

Je ziet dat deze classes gelijkaardig zijn aan dezelfde classes in de REST service.

Als je de client én de service programmeert van eenzelfde REST service, is het aan te raden

- Deze classes één keer te schrijven in een project van het type Class Library.
- Aan het REST service project een reference toe te voegen naar dit Class Library project.
- Aan het REST client project een reference toe te voegen naar dit Class Library project.

### 7.3 Data lezen met GET requests

Je wijzigt Program.cs:

```
using System;
using System.Net.Http;
namespace BrouwersClient
{
    class Program
    {
        static void Main(string[] args)
        {
            var client = new HttpClient(); // (1)
            var response =
                client.GetAsync("http://localhost:53810/brewers").Result; // (2)
            if (response.IsSuccessStatusCode) // (3)
            {
                var brouwers = response.Content.ReadAsAsync<Brouwers>().Result; // (4)
                brouwers.ForEach(
                    beknopt => Console.WriteLine(beknopt.ID + ":" + beknopt.Naam)); // (5)
                Console.Write("Kies een id:");
                int id = int.Parse(Console.ReadLine());
                foreach (var brouwerBeknopt in brouwers)
                {
                    if (brouwerBeknopt.ID == id)
                    {
                        response = client.GetAsync(brouwerBeknopt.Detail).Result; // (6)
                        if (response.IsSuccessStatusCode)
                        {
                            var brouwer = response.Content.ReadAsAsync<Brouwer>().Result; // (7)
                            Console.WriteLine(brouwer.Postcode + " " + brouwer.Gemeente);
                            break;
                        }
                    }
                }
            }
        }
    }
}
```

```

        else
        {
            Console.WriteLine("Brouwer niet meer gevonden");
        }
    }
}
else
{
    Console.WriteLine("Brouwers niet gevonden");
}
}
}
}
}

```

- (1) Je maakt een object van de class HttpClient.  
Je stuurt met zo'n object requests naar een REST service .
  - (2) Je doet een GET request met de method GetAsync.  
Je geeft als parameter de URL mee waarnaar je de GET request stuurt.  
De returnwaarde van de method GetAsync bevat een property result.  
Dit is de response die je terugkreeg van de REST service.
  - (3) De property IsSuccessStatusCode bevat true als de statuscode van de response gelijk is aan 200 (OK), 201 (Created) of 204 (No Content)
  - (4) De property Content bevat de response body.  
Je leest deze response body met de method ReadAsStringAsync.  
Je geeft aan deze method het type mee naar waar .net de response body (die XML of JSON) bevat moet converteren: Brouwers.  
De returnwaarde van de method ReadAsStringAsync bevat een property result.  
Dit is de response body, uitgedrukt als een object van het type Brouwers.
  - (5) Je itereert over de BrouwerBeknopt objecten in het Brouwers object.
  - (6) Je doet een GET request naar de URL van één brouwer.  
Je vond deze URL in de property Details van het BrouwerBeknopt object.
  - (7) De property Content bevat de response body.  
Je leest deze response body met de method ReadAsStringAsync.  
Je geeft aan deze method het type mee naar waar .net de response body (die XML of JSON) bevat moet converteren: Brouwer.
- Je klikt in de Solution Explorer met de rechtermuisknop op je Solution
  - Je kiest Set Startup Projects
  - Je kiest Multiple startup projects
  - Je kiest naast BrouwersClient en naast BrouwersService voor Start en je kiest OK

Je drukt Ctrl+F5. De client én de service starten en je kan de client uitproberen.

## 7.4 Data verwijderen met een DELETE request

Je wijzigt Program.cs:

```

using System;
using System.Net.Http;
namespace BrouwersClient
{
    class Program
    {
        static void Main(string[] args)
        {
            const string serviceURI = "http://localhost:53810/brewers";
            Console.WriteLine("Tik een id:");
            var id = int.Parse(Console.ReadLine());
            var client = new HttpClient();
            var response = client.DeleteAsync(serviceURI + "/" + id).Result;
        }
    }
}

```

(1)

```

        if (response.IsSuccessStatusCode)
        {
            response = client.GetAsync(serviceURI).Result;
            var brouwers = response.Content.ReadAsAsync<Brouwers>().Result;
            brouwers.ForEach(
                beknopt => Console.WriteLine(beknopt.ID + ":" + beknopt.Naam));
        }
        else
        {
            Console.WriteLine("Kan brouwer niet verwijderen");
        }
    }
}
}

```

(1) Je stuurt een DELETE request met de method DeleteAsync.

(2) Je leest de overgebleven brouwers.

Je kan de applicatie uitproberen.

## 7.5 Data toevoegen met een POST request

Je wijzigt Program.cs:

```

using System;
using System.Net;
using System.Net.Http;
namespace BrouwersClient
{
    class Program
    {
        static void Main(string[] args)
        {
            var brouwer = new Brouwer();
            Console.Write("Naam:");
            brouwer.Naam=Console.ReadLine();
            Console.Write("Postcode:");
            brouwer.Postcode = int.Parse(Console.ReadLine());
            Console.Write("Gemeente:");
            brouwer.Gemeente= Console.ReadLine();
            var client = new HttpClient();
            var response = client.PostAsJsonAsync<Brouwer>(
                "http://localhost:53810/brewers/", brouwer).Result;
            if (response.IsSuccessStatusCode)
            {
                Console.WriteLine("URL nieuwe brouwer:" + response.Headers.Location);
            }
            else
            {
                Console.WriteLine("Probleem bij toevoegen brouwer:"+response.StatusCode);
            }
        }
    }
}

```

(1) Je verstuurt een POST request met de method PostAsJsonAsync

Je vermeldt tussen <> het type object dat je in de request body meegeeft.

(2) De 1° parameter is de URL naar waar je de POST request stuurt. De 2° parameter is het object dat je in de request body wil plaatsen. De returnwaarde bevat een property Result. Dit is de response die je binnenkrijgt na het versturen van de request.

(3) De property IsSuccessStatusCode bevat true als de statuscode van de response gelijk is aan 200 (OK), 201 (Created) of 204 (No Content)

(4) Je toont de URL van de nieuwe brouwer. Je vindt die in de response header location.

Je kan de applicatie uitproberen.



## 7.6 Data wijzigen met een PUT request

Je wijzigt Program.cs:

```
using System;
using System.Net.Http;
namespace BrouwersClient
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Tik een id:");
            var id = int.Parse(Console.ReadLine());
            var client = new HttpClient();
            var url = "http://localhost:53810/brewers/" + id;
            var response = client.GetAsync(url).Result;
            if (response.IsSuccessStatusCode)
            {
                var brouwer = response.Content.ReadAsAsync<Brouwer>().Result;
                brouwer.Gemeente = brouwer.Gemeente.ToUpper();
                response = client.PutAsJsonAsync<Brouwer>(url, brouwer).Result;      (1)
                if (!response.IsSuccessStatusCode)
                {
                    Console.WriteLine("Er is een fout opgetreden:" + response.StatusCode);
                }
            }
        }
    }
}
```

- (1) Je verstuurt een PUT request met de method `PutAsJsonAsync`  
Je vermeldt tussen <> het type object dat je in de request body meegeeft.  
De eerste parameter is de URL naar waar je de POST request stuurt.  
De tweede parameter is het object dat je in de request body wil plaatsen.  
De returnwaarde bevat een property `Result`.  
Dit is de response die je binnenkrijgt na het versturen van de request.

Je kan de applicatie uitproberen en met een GET request in Fiddler nazien of de naam gewijzigd is.



Kaas client: zie takenbundel

## 8 COLOFON

**Domeinexpertisemanager:** Rita Van Damme

**Moduleverantwoordelijke:** Hans Desmet

**Medewerkers:** Hans Desmet

**Versie:** 27/11/2014

**Nummer dotatielijst:**