

ASP.NET Core (MVC)

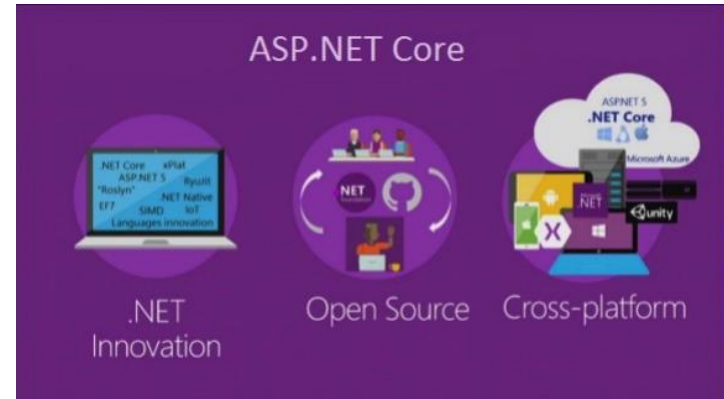
Bachelor IT

Sven Mariën

(sven.marien01@ap.be)

ASP.NET Core

- Server side framework
- = **Open-source, cross-platform framework for building modern internet connected applications (website, webapps, web api's,...)**
- => Microsoft's tegenhanger van **Express / Node.JS**



ASP.NET Core MVC What...?

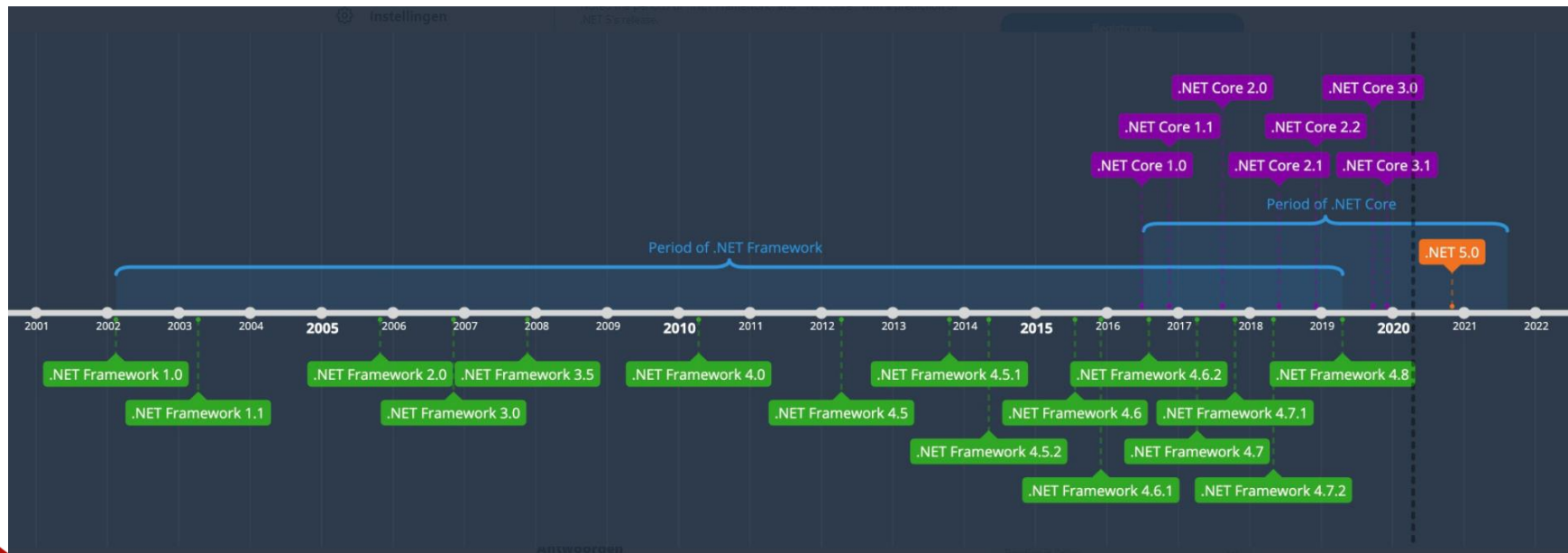
Even terug in de tijd:

- **ASP** = “Active Server Pages”. (1996)
 - Deze term dateert nog van 1996 waarbij Microsoft een aantal zaken lanceerde onder de naam “Active” (**Active** directory, **Active** X componenten, **Active** Template Library,...).
 - Active duidde op het feit dat de pagina in kwestie een combinatie van HTML en “server side script” bevatte. Dat script werd uitgevoerd aan de server zijde vooraleer de pagina werd teruggestuurd naar de client (browser). Dmv. Het script werd de inhoud van de pagina dynamisch ingevuld, bv. Aan de hand van gegevens uit een databank.
- **ASP.NET** (2002)
 - Enkele jaren later werd het mogelijk om de server side code te maken in c#, VB.NET of een andere .NET programmeertaal en werd het framework omgedoopt tot **ASP.NET**
- **ASP.NET MVC** (2009)
 - Eerste versie dateert van 2009, versie 6 kwam uit in 2015.
 - Werkt volgens het “Model View Controller” design patroon (-> Software Engineering)
 - Hierbij werd er meer structuur voorzien vanuit het framework om beter scheiding te krijgen tussen de html (View), de data (Model) en de code (Controller)

ASP.NET Core MVC What...? (2)

- Het ASP.NET MVC framework was/is heel populair bij de MS community.
- In 2016 bracht MS echter een eerste open source / cross-platform versie uit
- => **.NET core** 1.0 en **ASP.NET Core MVC** 1.0
 - Cross-platform: Draait zowel op Windows/ MacOS / Linux
 - Open source: source code staat op github
- “Build from the ground-up” (dus geen volgende versie van ASP.NET)
- Betere performantie, minder onderhoud, strakkere security,...
- Volledig modulair opgebouwd (nuGet packages), je kan dus enkel installeren wat je nodig hebt in je applicatie
- => sinds 2017: .NET Core 2.0 en ASP.NET Core MVC v2.0
- => sinds 2019: .NET Core 3.0 en ASP.NET Core MVC v3.0 (ondertussen **v3.1**) (+ WPF)
- => sinds december 2020: **.NET 5**
- => november 2021: .NET 6
- => november 2022: .NET 7
- => november 2023: .NET 8
- => november 2024: .NET 9 (planned)
- => november 2025: .NET10 (planned)

Evolutie van .NET 1 (via .NET core) naar .NET 5

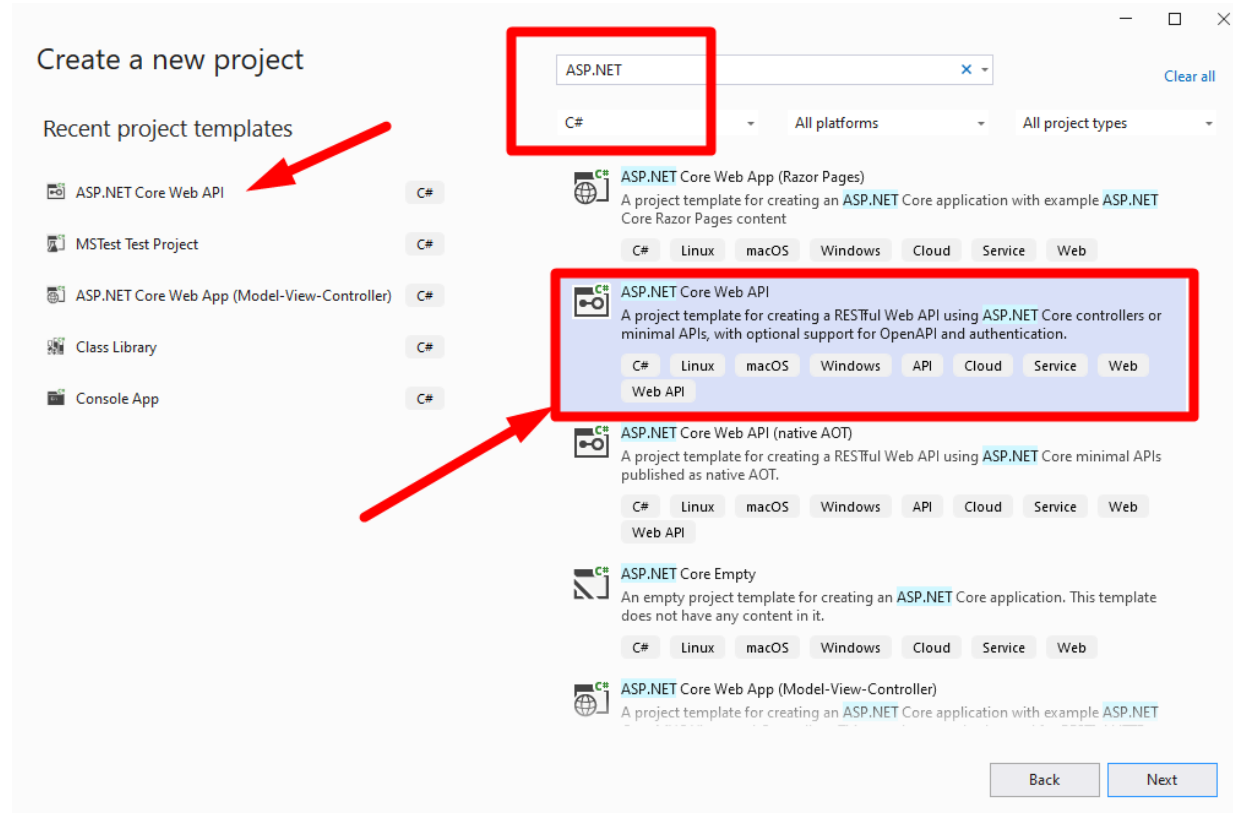


De toekomstplannen...

- Slechts 1 .NET platform blijft over: “A unified platform”



Aanmaken van een nieuw API project



Aanmaken van een nieuw Web API project

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 ⓘ

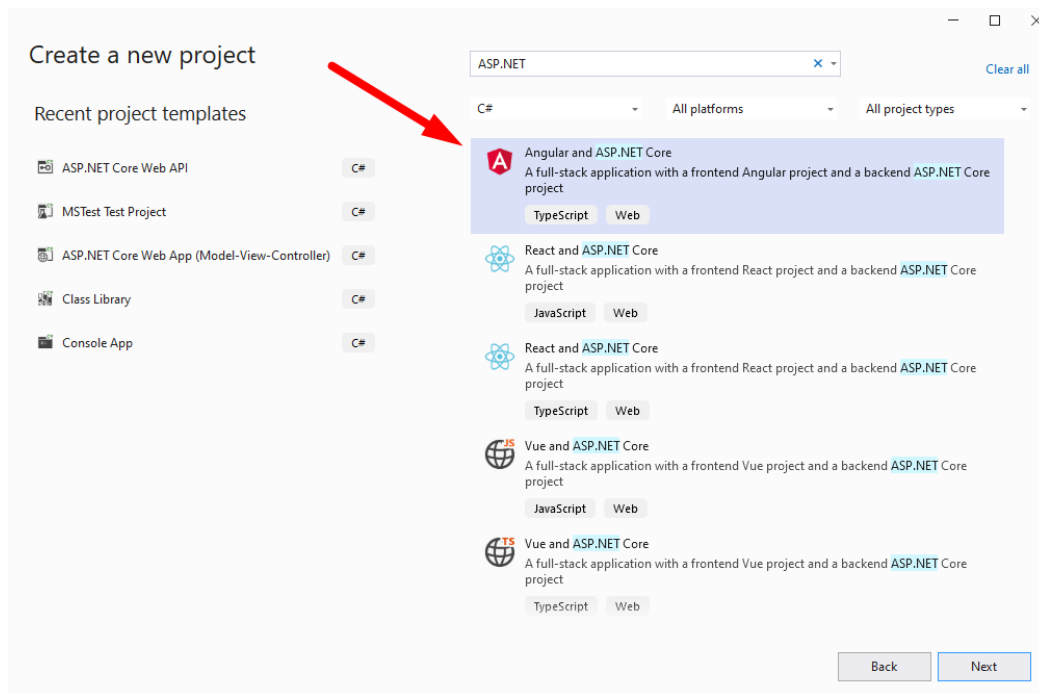
Docker OS ⓘ
[Linux ▼]

☒ Enable OpenAPI support ⓘ
☒ Do not use top-level statements ⓘ
☒ Use controllers ⓘ

Back Create

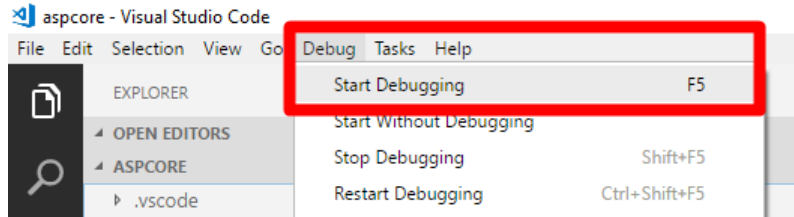
Aanmaken van een nieuw project (alternatief)

- Angular SPA client + ASP.NET Core Web API server in 1 VS solution

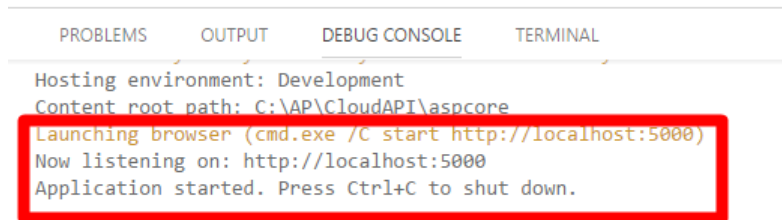


Project opstarten en debuggen

- Via F5 of the Debug menu:



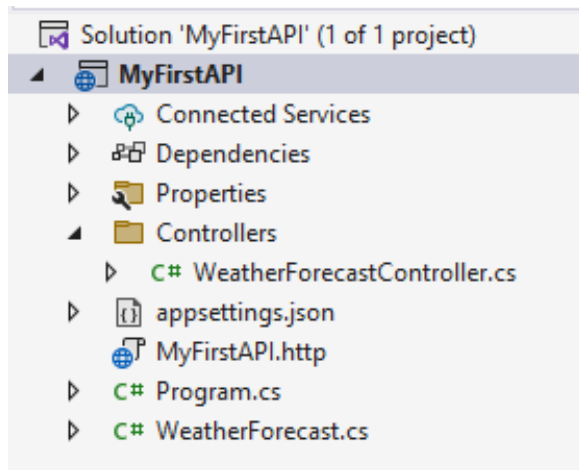
- Browser wordt automatisch opgestart, URL wordt weergegeven



Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\2.0.3

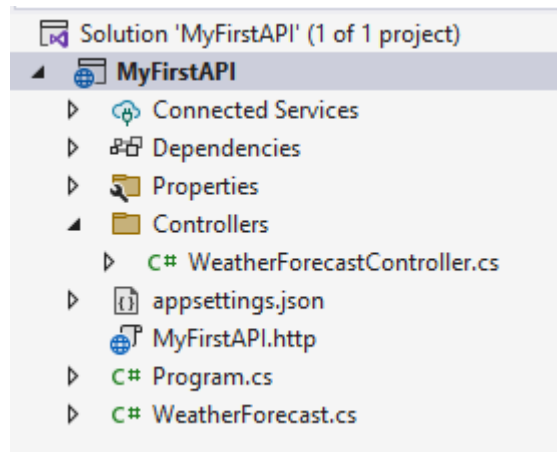
Structuur van het (lege) project

- **xxxxx.csproj** -> project file (= onzichtbaar in Visual Studio)
 - Bevat de verwijzing naar externe (nuget) packages
 - Gebruik de plug-in om packages toe te voegen
- **Program.cs**
 - Bevat de **main** functie (zoals een console app)
 - Er wordt een default webserver opgestart
 - Plaats waar allerlei configuratie kan gebeuren
 - Dependency injection instellen
 - Middleware toevoegen
 - ...



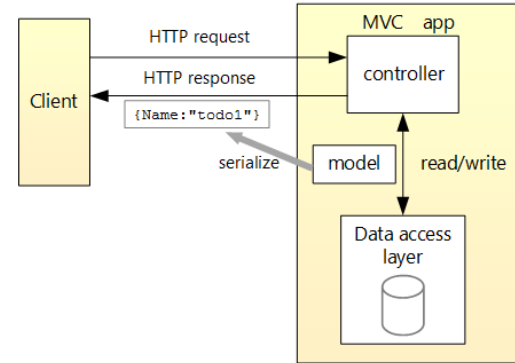
Structuur van het (lege) project (2)

- **Appsettings.json**
 - Logging configuratie, ...
- **Controllers** map
 - Hieronder komen alle “controllers”

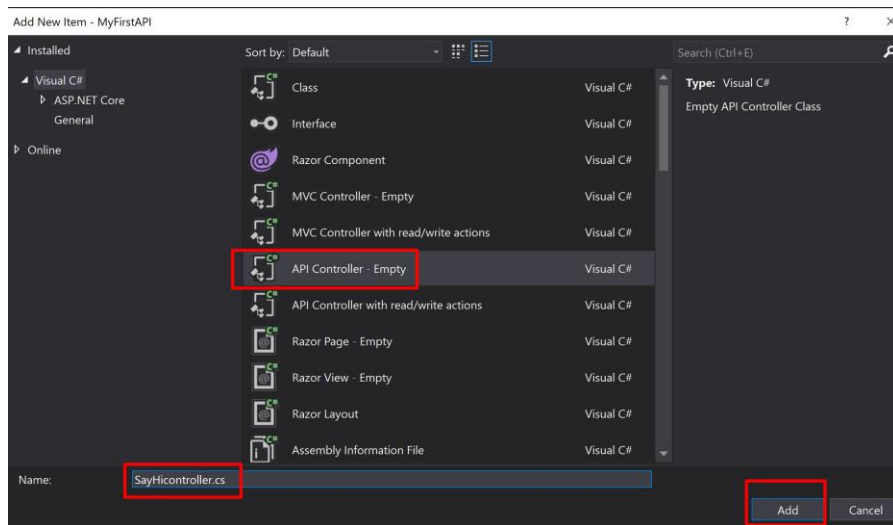
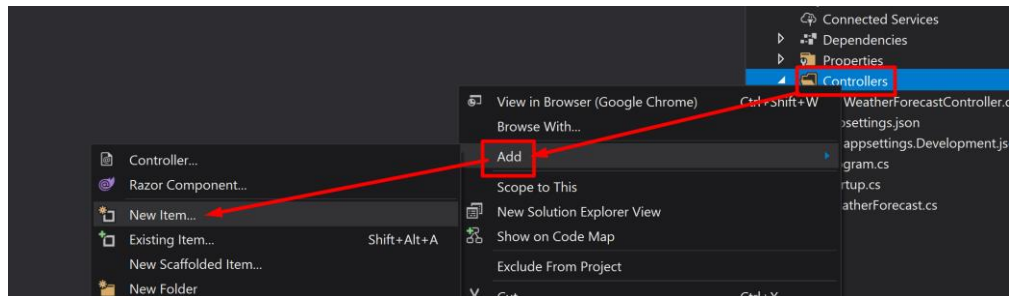


Wat is een Controller ?

- Controllers
 - Handelt een bepaalde **Http Request** af
 - Stuurt een **Http Response** terug
 - Gebeurt mbv. “Controller Actions”
- Een project kan 1 of meerdere controllers bevatten
 - Typisch 1 controller per “onderwerp”
- Het ASP framework is verantwoordelijk voor:
 - het aanmaken van controller objecten
 - het aanroepen van de juiste controller + action



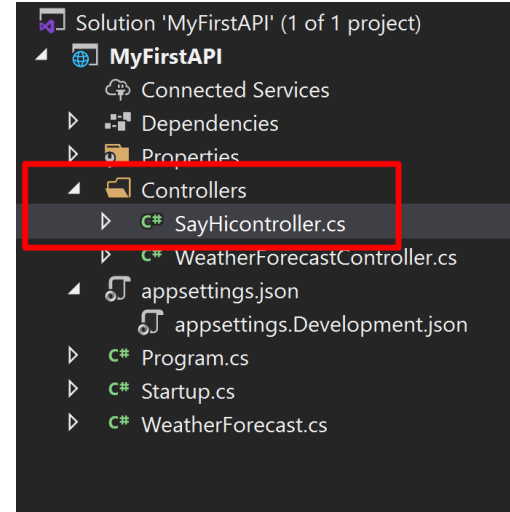
Controller aanmaken



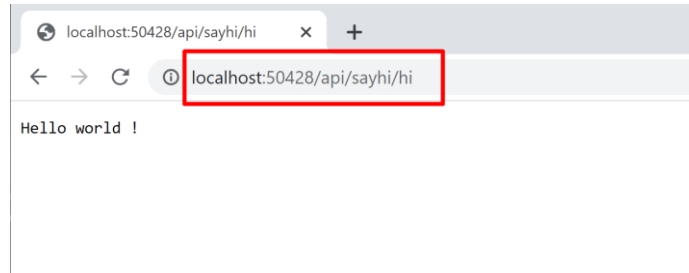
Your first controller

- Voeg een Action (=methode) toe in je controller:

```
[Route("api/[controller]")]
[ApiController]
0 references
public class SayHiController : ControllerBase
{
    //This action is accessible via http://xxx/api/sayhi/hi
    [Route("hi")]
    [HttpGet]
    0 references
    public IActionResult Hello()
    {
        return Content("Hello world !");
    }
}
```

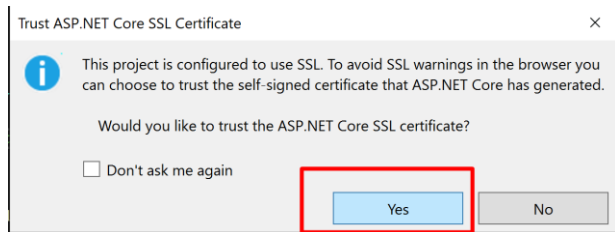


- En start vervolgens het project op:



Met https...

- Indien je was vergeten om https uit te vinken bij de aanmaak van het project zal er de eerste maal een certificaat worden geïnstalleerd:



Beveiligingswaarschuwing



U staat op het punt om een certificaat van een certificeringsinstantie (CA) te installeren die als vertegenwoordiging optreedt van:

localhost

Kan niet valideren of het certificaat daadwerkelijk afkomstig is van localhost. Neem contact met localhost op om de verleners te laten bevestigen. Gebruik het volgende nummer voor deze procedure:

Vingerafdruk (sha1): 0D5CCE86 99C8E913 FD4D32A3
C3AC6C8D DDAD2190

Waarschuwing:

Als u dit basiscertificaat installeert worden automatisch alle certificaten vertrouwd die door deze certificeringsinstantie zijn verleend. Installatie van een certificaat met een niet-geverifieerde vingerafdruk is een beveiligingsrisico. Als u op Ja klikt, gaat u akkoord met dit risico.

Wilt u dit certificaat installeren?

Ja

Nee

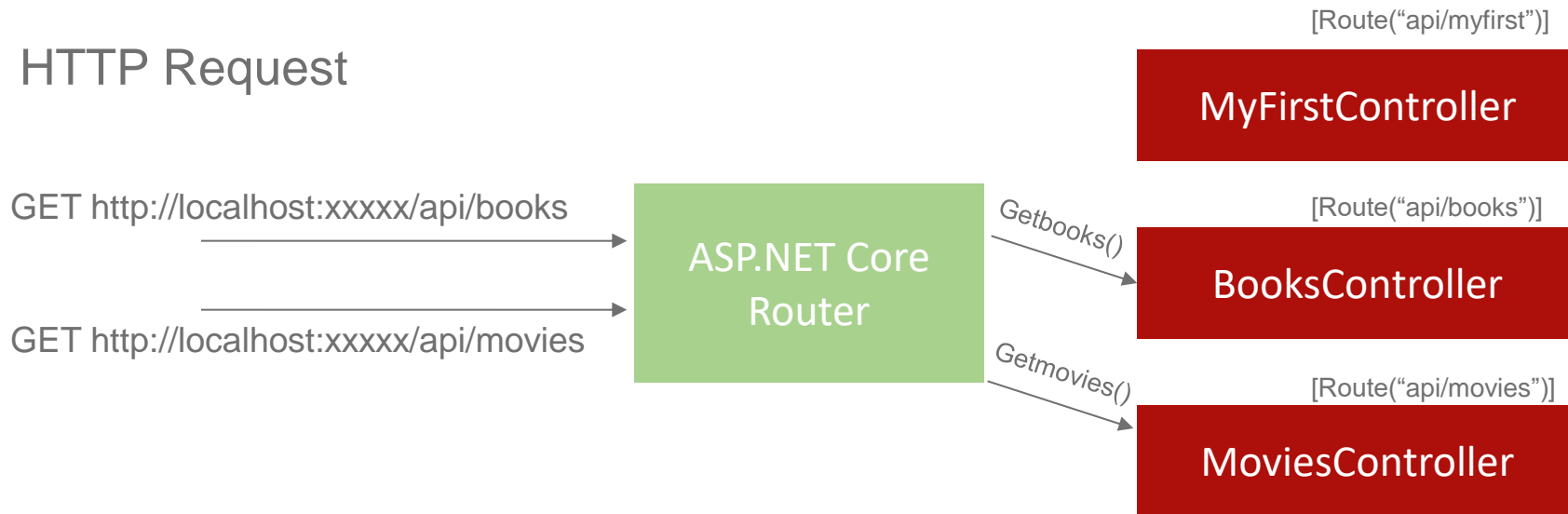
Your first controller: explained

- Elke controller
 - **erft** van de **ControllerBase** klasse !
 - Is voorzien van het **[ApiController]** attribuut
- Een controller bevat controller **Actions** (methoden van de klasse)
 - Elke actie moet worden ingesteld voor een bepaalde “**route**” + **VERB**
 - Elke actie zal steeds een **ActionResult** teruggeven (wordt omgezet naar een **Http response**)

```
{  
    [Route("api/[controller]")]  
    [ApiController]  
    0 references  
    public class SayHiController : ControllerBase  
    {  
        //This action is accessible via http://xxx/api/sayhi/hi  
        [Route("hi")]  
        [HttpGet]  
        0 references  
        public IActionResult Hello()  
        {  
            return Content("Hello world !");  
        }  
    }  
}
```

Controllers & routing

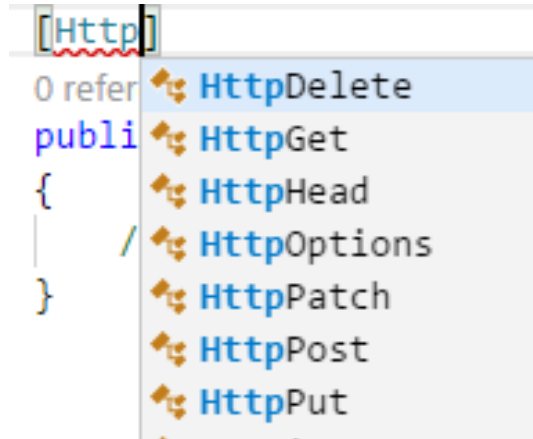
HTTP Request



Als je de standaard naamgeving gebruikt kan je als route instellen bij elke controller: `[Route("api/[controller]")]`

HTTP “verbs”

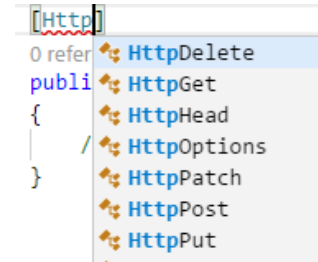
- Het HTTP protocol bevat verschillende “Verbs”
- Deze moeten we instellen bij elke controller Action
- Wat is de betekenis hiervan in een API ?



Verbs en CRUD

- CRUD = Create, Read, Update & Delete
- Om alle operaties te kunnen doen moeten we de 4 CRUD acties voorzien in onze controller

Actie	Verb
Read	GET
Create	POST
Delete	DELETE
Update	PUT en/of PATCH



Controller met de 4 CRUD acties

- We moeten dus 4 Actions voorzien in elke controller
- De ASP router zal ervoor zorgen dat de juiste actie wordt aangeroepen in functie van:
 - De Route
 - De Verb

```
[Route("api/[controller]")]
[ApiController]
0 references
public class StudentsController : ControllerBase
{
    [HttpGet] Read
    0 references
    public IActionResult GetStudents()...
    [HttpPost] Create
    0 references
    public IActionResult CreateStudent()...
    [HttpPut] Update
    0 references
    public IActionResult UpdateStudent()...
    [HttpDelete] Delete
    0 references
    public IActionResult DeleteStudent()...
}
```

Instellen van de routes

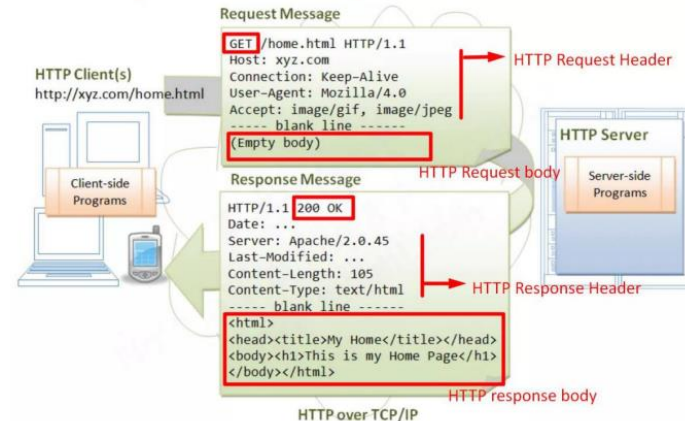
- De route kan ingesteld worden
 - Op niveau van de **controller** (standaard)
 - Op niveau van de **action**
 - Of een combinatie van **beide** (zie verder)

```
[Route("api/[controller]")] ←  
[ApiController]  
0 references  
public class StudentsController : ControllerBase  
{  
    [HttpGet]  
    0 references  
    public IActionResult GetStudents()...  
  
    [HttpPost]  
    0 references  
    public IActionResult CreateStudent()...  
  
    [HttpPut]  
    0 references  
    public IActionResult UpdateStudent()...  
  
    [HttpDelete]  
    0 references  
    public IActionResult DeleteStudent()...  
}
```

```
[ApiController]  
0 references  
public class StudentsController : ControllerBase  
{  
    [Route("api/[controller]")] ←  
    [HttpGet]  
    0 references  
    public IActionResult GetStudents()...  
  
    [Route("api/[controller]")] ←  
    [HttpPost]  
    0 references  
    public IActionResult CreateStudent()...  
  
    [Route("api/[controller]")] ←  
    [HttpPut]  
    0 references  
    public IActionResult UpdateStudent()...  
  
    [Route("api/[controller]")] ←  
    [HttpDelete]  
    0 references  
    public IActionResult DeleteStudent()...  
}
```

Resultaat van een action (ActionResult)

- De opgevraagde gegevens worden in de **body** van de **response** teruggestuurd.
- Het formaat dat hierbij gebruikt wordt is veelal **JSON**, maar ook andere formaten zijn mogelijk (bv. XML)
- Daarnaast heeft een **HTTP response** ook nog:
 - **Header** velden
 - **Statuscode** (200=OK)



Voorbeeld van een GET /api/students

```
[Route("api/[controller]")]
[ApiController]
1 reference
public class StudentsController : ControllerBase
{
    private List<Person> people = new List<Person>();

    [HttpGet]
    0 references
    public IActionResult GetStudents()
    {
        return Ok(people);
    }
}
```

```
public class Person
{
    2 references
    public int Id { get; set; }
    0 references
    public string Name { get; set; }
    0 references
    public string FirstName { get; set; }
    2 references
    public DateTime BirthDate { get; set; }
}
```

The screenshot shows a web browser at the URL `https://localhost:44309/api/students`. The response is a JSON array of two student objects, highlighted with a red box:

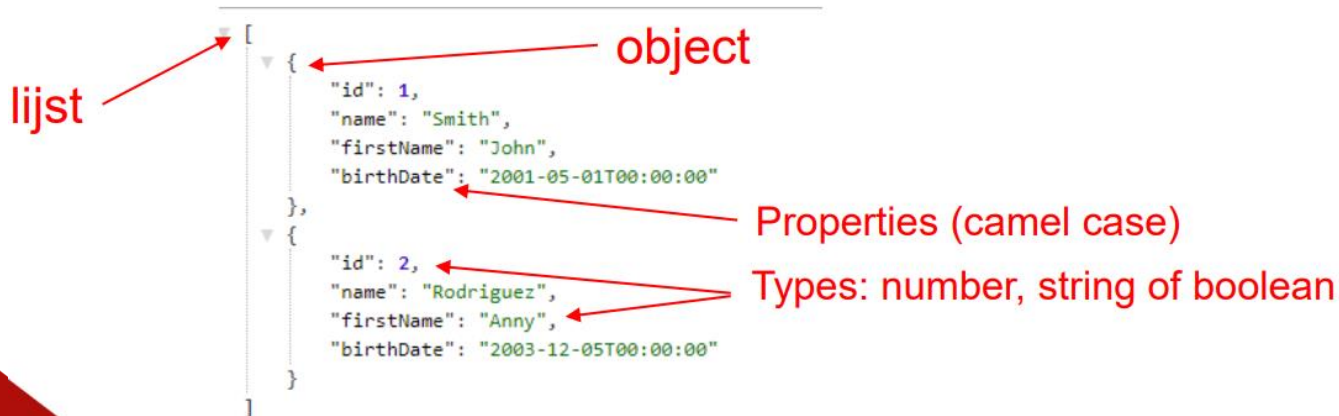
```
[
  {
    "id": 1,
    "name": "Smith",
    "firstName": "John",
    "birthDate": "2001-05-01T00:00:00"
  },
  {
    "id": 2,
    "name": "Rodriguez",
    "firstName": "Anny",
    "birthDate": "2003-12-05T00:00:00"
  }
]
```

The browser's developer tools are open, showing the Network tab. The request details are as follows:

- Request URL: `https://localhost:44309/api/students`
- Request Method: GET
- Status Code: 200
- Remote Address: `::1:44309`
- Referrer Policy: `strict-origin-when-cross-origin`
- Response Headers:
 - `content-type: application/json; charset=utf-8`
 - `date: Fri, 17 Sep 2021 12:16:10 GMT`
 - `server: Microsoft-IIS/10.0`
 - `x-powered-by: ASP.NET`

JSON Formaat

- JSON = **J**avascript **O**bject **N**otation
- Vandaag de dag het meeste gebruikte formaat om gegevens uit te wisselen.
- Het ASP framework zal alle objecten / lijsten die we terug geven vanuit een action omzetten naar JSON formaat
 - Zal deze in de **body** van de response plaatsen
 - Zal het content-type instellen op **application/json**
- Dit noemen we JSON “serialization”



The diagram shows a JSON array structure with two objects. Red arrows point from text labels to specific parts of the JSON code:

- lijst** points to the opening square bracket `[`.
- object** points to the first object `{`.
- Properties (camel case)** points to the property names `"id"`, `"name"`, `"firstName"`, and `"birthDate"`.
- Types: number, string of boolean** points to the values `1`, `"Smith"`, `"John"`, and `"2001-05-01T00:00:00"`.

```
[
  {
    "id": 1,
    "name": "Smith",
    "firstName": "John",
    "birthDate": "2001-05-01T00:00:00"
  },
  {
    "id": 2,
    "name": "Rodriguez",
    "firstName": "Anny",
    "birthDate": "2003-12-05T00:00:00"
  }
]
```

Status code instellen

- We willen ook de **status code** kunnen instellen in de response wanneer we een resultaat teruggeven.
- Hiervoor zijn er **helper** methodes aanwezig op de base class (ControllerBase)
 - Ok() => code 200
 - NotFound() => code 404
 - BadRequest() => code 400
 - enz...

```
[Route("api/[controller]")]
[ApiController]
1 reference
public class StudentsController : ControllerBase
{
    private List<Person> people = new List<Person>();

    [HttpGet]
    0 references
    public IActionResult GetStudents()
    {
        return Ok(people);
    }
}
```

Status code in de response

- Enkele veelgebruikte codes:
- 2xx (actie is gelukt)
 - 200 bij opvragen van gegevens
 - 201 bij aanmaken (post)
 - 204 bv. bij delete
- 4xx (probleem aan de client zijde)
 - 400 de request is niet juist geformuleerd
 - 401 niet aangemeld
 - 403 wel aangemeld, maar geen toegang
 - 404 de gevraagde resource werd niet gevonden
- 5xx (probleem aan de server zijde)
 - 500: als er “een” fout is opgetreden aan de server zijde

Level 200 Success

200 - OK
201 - Created
204 - No Content

Level 400 Client Error

400 - Bad Request
401 - Unauthorized
403 - Forbidden
404 - Not Found
409 - Conflict

Level 500 Server Error

500 - Internal
Server Error

Status code zelf instellen in het “ActionResult”

- Indien je geen helper methode terugvindt kan je ook zelf de status code instellen, gebruik hiervoor dan de **Content()** methode.
- Stel je geen status code in, dan zal 200 worden teruggestuurd.

```
public class ContentResult : ActionResult
{
    public ContentResult();

    //
    // Summary:
    //     Gets or set the content representing the body of the response.
    public string Content { get; set; }
    //
    // Summary:
    //     Gets or sets the Content-Type header for the response.
    public string ContentType { get; set; }
    //
    // Summary:
    //     Gets or sets the HTTP status code.
    public int? StatusCode { get; set; }
}
```


```
public class SayHiController : ControllerBase
{
    //This action is accessible via http://xxxx/api/sayhi/hi
    [Route("hi")]
    [HttpGet]
    0 references
    public IActionResult Hello()
    {
        var result = Content("Hello world !");
        result.StatusCode = 204;
        return result;
    }
}
```

Indien je geen status code instelt zal default **200** worden teruggestuurd.

Content Type zelf instellen in de response


- Op het ContentResult object kan je het type instellen
- Als je geen content Type opgeeft zal default: text/plain worden teruggestuurd.

```
// This action is accessible via the url (route): http://localhost:5000/hi
[Route("hi")]
[HttpGet]
0 references
public IActionResult Hello()
{
    var result = Content("Hello world !");
    result.ContentType = "text/plain";
    result.StatusCode = 200;
    return result;
}
```



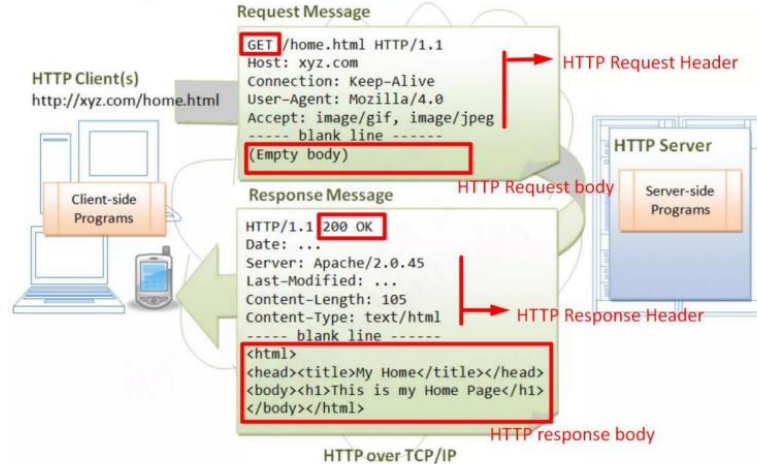
```
public class ContentResult : ActionResult
{
    public ContentResult();

    //
    // Summary:
    //     Gets or set the content representing the body of the response.
    public string Content { get; set; }
    //
    // Summary:
    //     Gets or sets the Content-Type header for the response.
    public string ContentType { get; set; }
    //
    // Summary:
    //     Gets or sets the HTTP status code.
    public int? StatusCode { get; set; }
}
```



Hoe parameters meegeven in de request

- Via de URL:
 - <http://localhost:xxxxx/api/students?name=smith&age=22>
 - <http://localhost:xxxxx/api/students/1>
- Via de header velden
 - **Key/value** pairs
- Via de body
 - **Object**
 - Lijst van objecten



Gebruik van query parameters

- Bv met deze URL:
 - <http://localhost:xxxxx/api/students?name=smith&age=22>
 - Wensen we alle studenten te bekomen met
 - Naam = “smith”
 - En Leeftijd = 22 jaar
 - Hoe kunnen we de query parameters nu uitlezen in de controller ?

Parameters uit de request halen

- ControllerBase bevat eveneens de properties:
 - Request
 - Response
- Via het HttpRequest object kunnen we vervolgens alle info uit de request halen:
 - Query: een lijst van query parameters
 - Headers: een lijst van alle header velden
 - Body: de inhoud van de body

```
...public abstract class ControllerBase
{
    protected ControllerBase();

    ...public HttpResponseMessage Response { get; }
    ...public HttpRequest Request { get; }
    ...public HttpContext HttpContext { get; }
```

```
...public abstract class HttpRequest
{
    protected HttpRequest();

    ...public abstract ICollection<string> Query { get; set; }
    ...public abstract bool HasFormContentType { get; }
    ...public virtual PipeReader BodyReader { get; }
    ...public abstract Stream Body { get; set; }
    ...public abstract string ContentType { get; set; }
    ...public abstract long? ContentLength { get; set; }
    ...public abstract IRequestCookieCollection Cookies { get; set; }
    ...public abstract IDictionary<string, string> Headers { get; }
    ...public abstract string Protocol { get; set; }
    ...public virtual RouteValueDictionary RouteValues { get; set; }
    ...public abstract QueryString QueryString { get; set; }
    ...public abstract PathString Path { get; set; }
    ...public abstract PathString PathBase { get; set; }
    ...public abstract HostString Host { get; set; }
    ...public abstract bool IsHttps { get; set; }
    ...public abstract string Scheme { get; set; }
    ...public abstract string Method { get; set; }
    ...public abstract HttpContext HttpContext { get; }
    ...public abstract ICollection<string> Form { get; set; }
```


Uitlezen van de query parameters

- Optie 1:

```
[HttpGet]
0 references
public IActionResult GetStudents()
{
    string name = null;
    string age = null;


    if (Request.Query.ContainsKey("name"))
        name = Request.Query["name"];
    if (Request.Query.ContainsKey("age"))
        age = Request.Query["age"];    //TODO: check if valid number and convert to int

    //TODO: filter these students from the list.
    return Ok(people);
}
```

- Maar het ASP framework doet voor ons nog veel meer,
We kunnen ook als volgt tewerk gaan.

- Optie 2:

```
[HttpGet]
0 references
public IActionResult GetStudents(string name, int age)
{
    //TODO: filter these students from the list.
    return Ok(people);
}
```



Uitlezen van de header velden / body

- Heel gelijkaardig kunnen we ook de header velden en de body uitlezen via het request object, of we gebruiken de eenvoudigere manier waarbij het ASP framework deze zaken voor ons gaat opzoeken in de request
- We kunnen daarbij met extra attributen aangeven waar de info moet gehaald worden.
 - **[FromQuery]**
 - **[FromHeader]**
 - **[FromBody]**
 - ...

```
[HttpGet]
0 references
public IActionResult GetStudents([FromHeader] string licenseKey, [FromQuery] string name, [FromQuery] int age)
{
    //TODO: filter these students from the list.
    return Ok(people);
}
```

```
[HttpPost]
0 references
public IActionResult CreateStudent([FromBody] Person student)
{
    return Ok(people);
}
```

Route op controller **en** action

- Hoe kunnen we deze afhandelen ?
 - <http://localhost:xxxxx/api/students/1>
 - Hiermee willen we enkel de student met Id=1 opvragen (zie later bij REST voor meer info hierover)
- Zoals reeds eerder aangegeven is het ook mogelijk om op 2 niveau's een route in te stellen
- Dit kunnen we nu gebruiken voor dergelijke URL af te handelen

Route op controller **en** action (2)

- Er zijn nu **2 actions** die via een **GET** request kunnen worden aangeroepen
- Dit mag aangezien de **route** voor de actions **verschillend** is !
- Ook hier wordt de **Id** door het framework uitgelezen uit de URL

```
[Route("api/[controller]")]
[ApiController]
1 reference
public class StudentsController : ControllerBase
{
    private List<Person> people = new List<Person>();

    /// <summary>
    /// GET naar api/students
    /// </summary>
    /// <param name="name"></param>
    /// <param name="age"></param>
    /// <returns></returns>
    [HttpGet]
    0 references
    public IActionResult GetStudents([FromQuery] string name, [FromQuery] int age)
    {
        //TODO: filter these students from the list.
        return Ok(people);
    }

    /// <summary>
    /// GET naar api/students/1
    /// </summary>
    /// <param name="Id"></param>
    /// <returns></returns>
    [Route("{Id}")]
    [HttpGet]
    0 references
    public IActionResult GetStudentsById(int Id)
    {
        //TODO: filter the student with Id == 1 from the list.
        return Ok(people);
    }
}
```

Create / Update en Delete

- Gelijkaardig kunnen we ook de andere actions nu uitbreiden zodat de nodige parameters kunnen worden doorgegeven aan de actions:

```
[HttpPost]
0 references
public IActionResult CreateStudent([FromBody] Person student)
{
    //TODO: add the student
    return Ok(people);
}

[Route("{Id}")]
[HttpDelete]
0 references
public IActionResult DeleteStudent(int Id)
{
    //TODO: lookup the student and remove
    return Ok(people);
}

[Route("{Id}")]
[HttpPut]
0 references
public IActionResult UpdateStudent(int Id, [FromBody] Person student)
{
    //TODO: lookup the student and update
    return Ok(people);
}
```

Lifecycle van een controller

- We hadden reeds gezien dat we zelf geen nieuwe controllers moeten aanmaken.
- Dit wordt gedaan door het ASP framework
 - Wanneer gebeurt dit nu juist ?
 - En hoelang blijft een controller object bestaan ?

Lifecycle van een controller (2)

