

# ASP.NET Core (MVC)

Elektronica – ICT

Sven Mariën

(sven.marien01@ap.be)

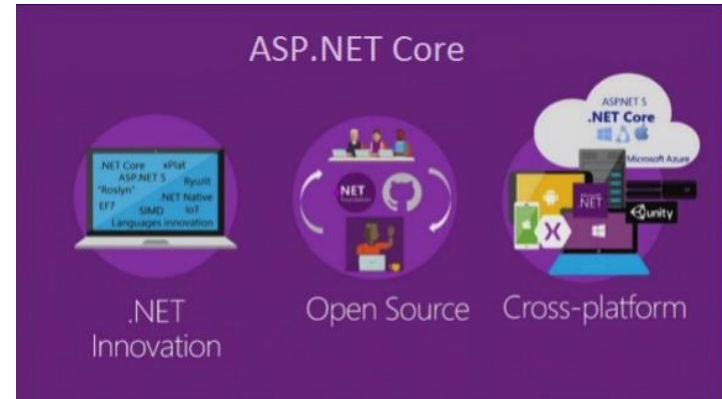


AP HOGESCHOOL  
ANTWERPEN

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...?

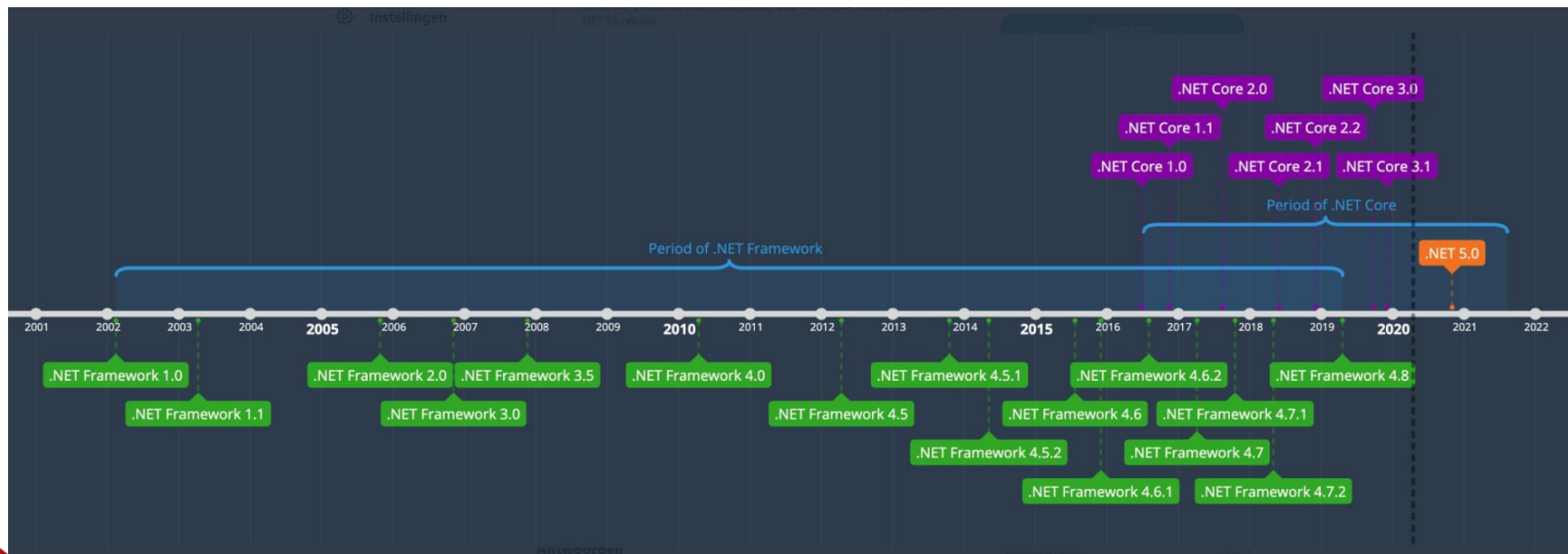
Even terug in de tijd:

- **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 (vbscript, jscript of perl) 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. adhv. 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. 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 (planned)

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

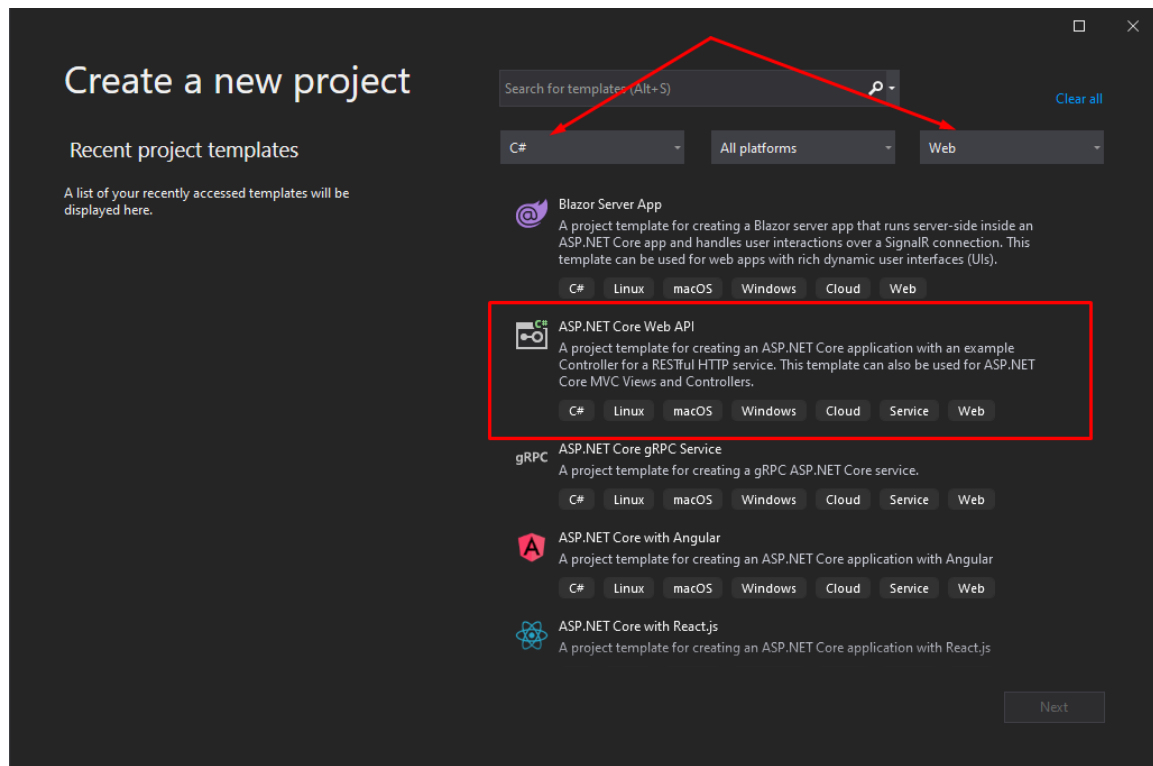


# De toekomstplannen...

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



# VS 2019: Aanmaken van een nieuw project



# VS 2019: Aanmaken van een nieuw Web API project

**Additional information**

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

Target Framework ⓘ  
.NET 5.0 (Current)

Authentication Type ⓘ  
None

☒ Configure for HTTPS ⓘ

☐ Enable Docker ⓘ

Docker OS ⓘ  
Linux

☒ Enable OpenAPI support ⓘ

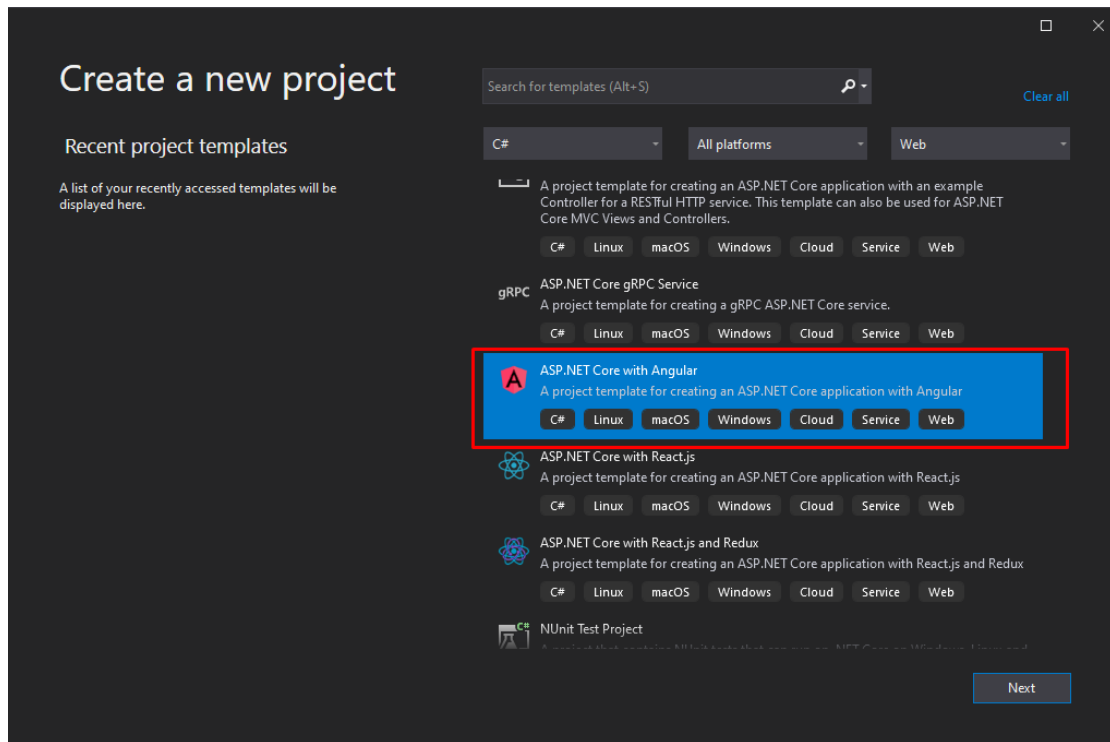
Target Framework ⓘ  
.NET 5.0 (Current)  
.NET Core 3.1 (Long-term support)  
.NET 5.0 (Current)

Back Create



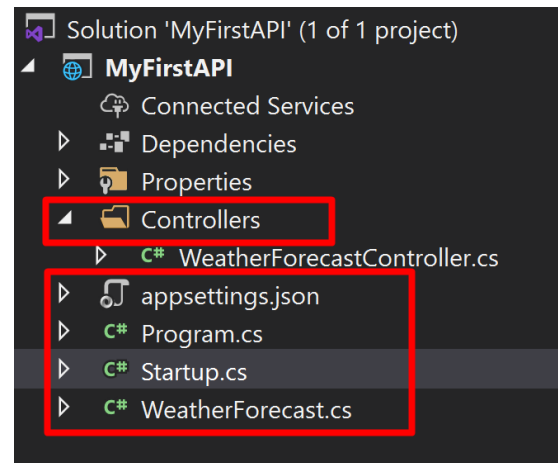
# VS 2019: Aanmaken van een nieuw project (**alternatief**)

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



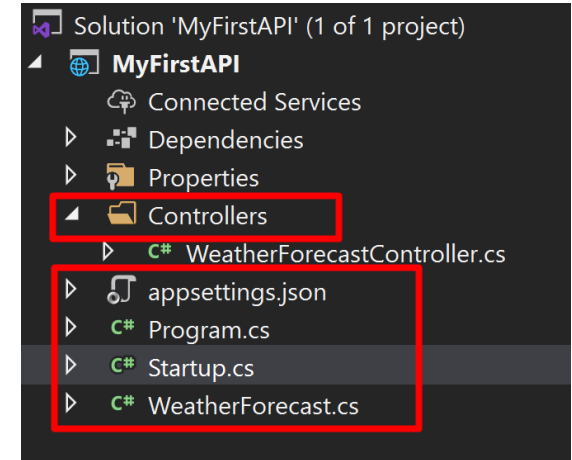
# Structuur van het (lege) project

- **xxxxx.csproj** -> project file
  - 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
  - De **Startup** class als opstart parameter
- **Startup.cs**
  - Bevat een class waar allerlei configuratie kan gebeuren
  - Dependency injection instellen
  - Middleware pipeline instellen (bv. Authenticatie,..)
  - ...



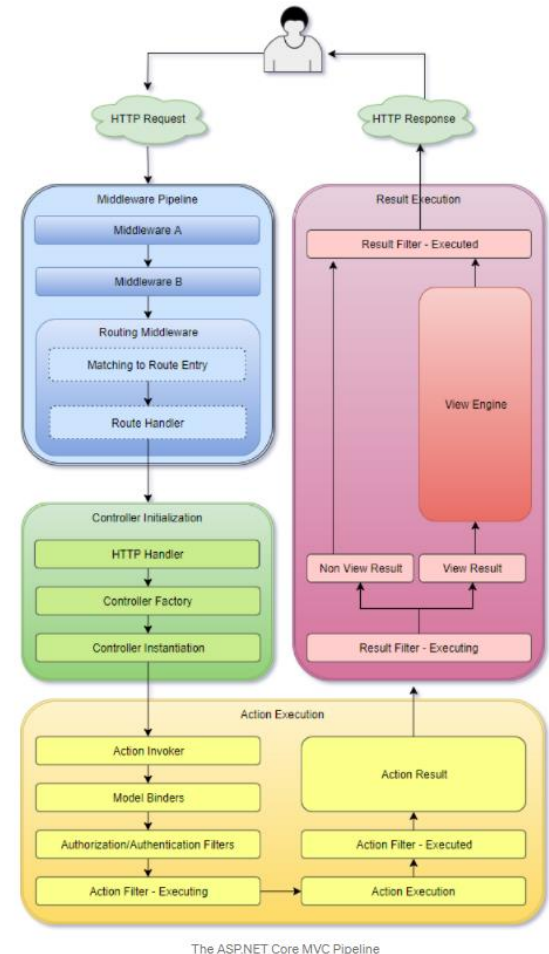
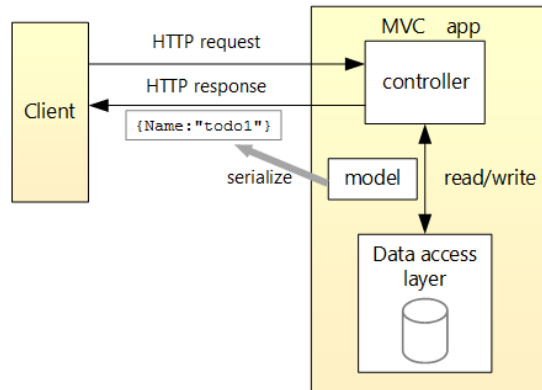
# Structuur van het (lege) project (2)

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

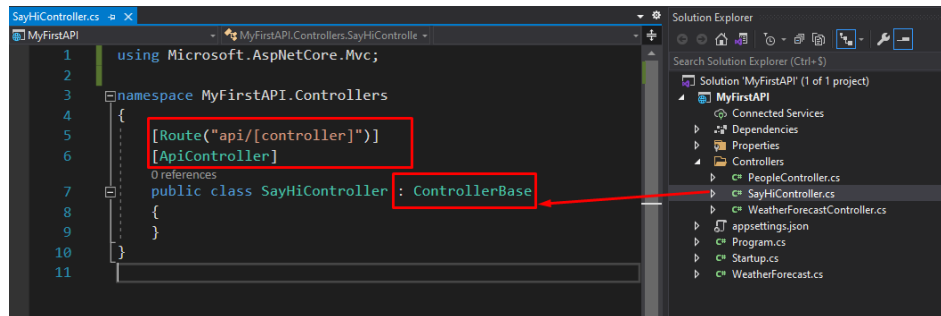
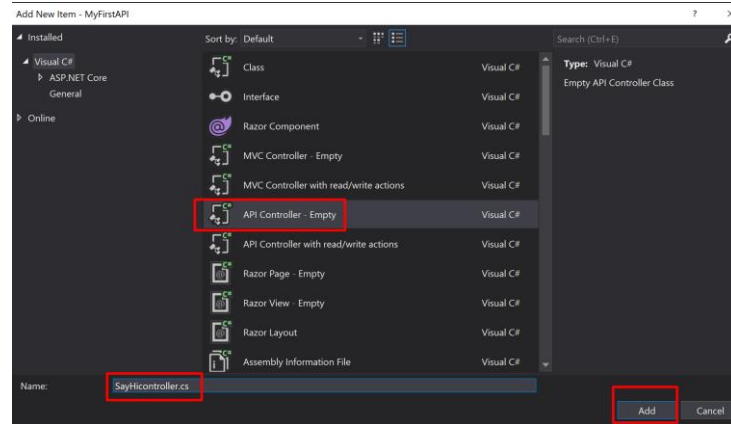
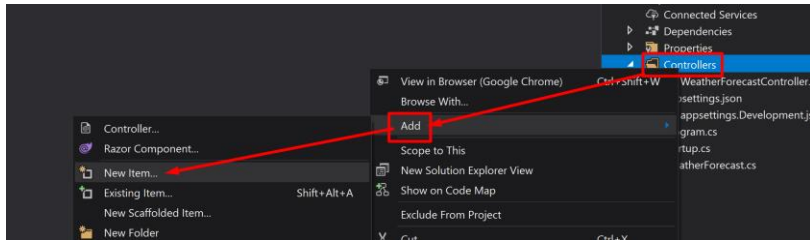


# Wat is een Controller ?

- De ASP.NET routing “middleware” zorgt ervoor dat een request bij de juiste controller terecht komt.
- De controller:
  - Handelt de **Http Request** af
  - Stuurt een **Http Response** terug
  - Gebeurt met behulp van “Controller Actions”
- Een project kan/mag 1 of meerdere controllers bevatten.



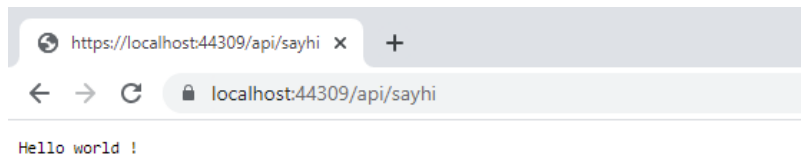
# Controller aanmaken



# Your first controller

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

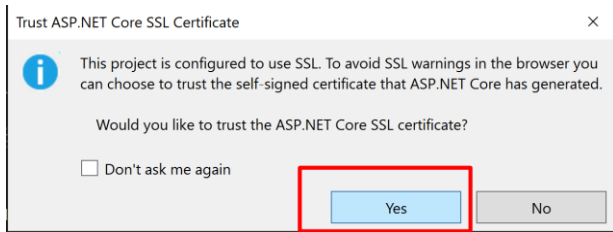
```
[Route("api/[controller]")]
[ApiController]
0 references
public class SayHiController : ControllerBase
{
    [HttpGet]
    0 references
    public IActionResult Hello()
    {
        return Ok("Hello world !");
    }
}
```



- En start vervolgens de API 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.

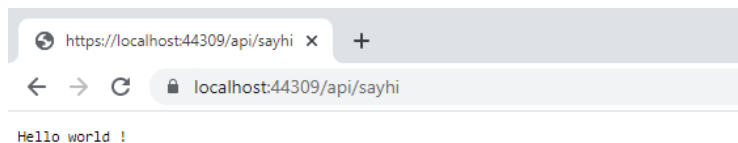
Wilt u dit certificaat installeren?

Ja

Nee

# Your first controller: explained

- Elke controller
  - **erft** van de **ControllerBase** klasse !
  - Is voorzien van het **[ApiController]** attribute
- 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
{
    [HttpGet]
    0 references
    public IActionResult Hello()
    {
        return Ok("Hello world !");
    }
}
```



# Controllers en routing

## HTTP (GET) Request

`http://localhost:xxxxx/api/sayhi`

`http://localhost:xxxxx/api/students`

`http://localhost:xxxxx/api/courses`

ASP Middleware  
pipeline

`[Route("api/sayhi")]`

SayHiController

`[Route("api/students")]`

StudentsController

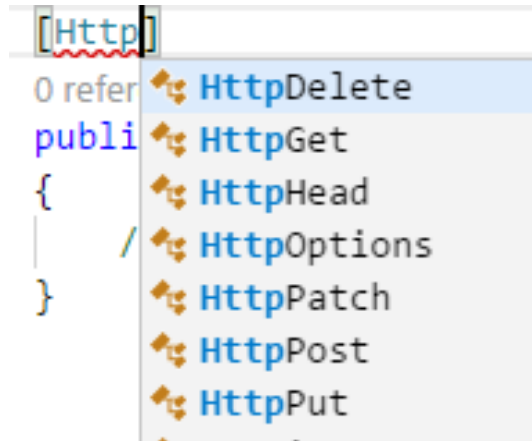
`[Route("api/courses")]`

CoursesController

*\* Als je de standaard naamgeving volgt kan je als route gebruiken: `[Route("api/[controller]")]`*

# HTTP “Verbs”

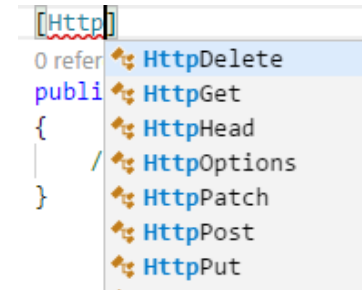
- Het HTTP protocol bevat verschillende “Verbs”
- Deze kunnen we instellen bij de controller Action(s)
- Wat is de betekenis hiervan in onze API ?



# 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 kunnen nu dus voor elke CRUD actie een Action voorzien in de controller.
- De ASP router zal ervoor zorgen dat de juiste actie wordt aangeroepen in functie van:
  - De Route
  - De Verb

```
[Route("api/[controller]")]
[ApiController]
1 reference
public class StudentsController : ControllerBase
{
    0 references
    public StudentsController()...

    [HttpGet] Read
    0 references
    public IActionResult GetStudents()...

    [HttpPost] Create
    0 references
    public IActionResult CreateStudent()...

    [HttpDelete] Delete
    0 references
    public IActionResult DeleteStudent()...

    [HttpPut] Update
    0 references
    public IActionResult UpdateStudent()...
```

# Route instellen op een Action

- De **route** kan ingesteld worden
  - op niveau van de **controller**
  - op het niveau van een **action**
  - of een combinatie van beide (zie verder)

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

    [HttpGet]
    0 references
    public IActionResult GetStudents()...

    [HttpPost]
    0 references
    public IActionResult CreateStudent()...

    [HttpDelete]
    0 references
    public IActionResult DeleteStudent()...

    [HttpPut]
    0 references
    public IActionResult UpdateStudent()...
```

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

    [Route("api/[controller]")]
    [HttpGet]
    0 references
    public IActionResult GetStudents()...

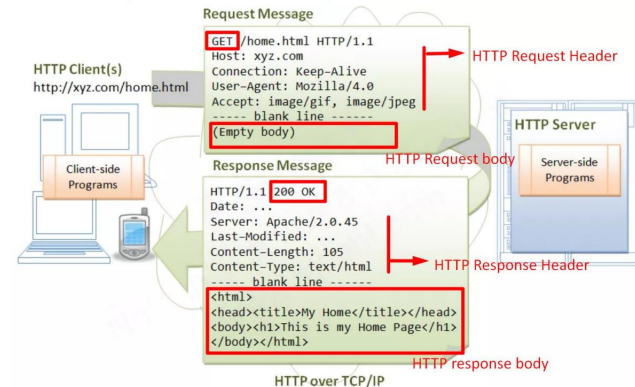
    [Route("api/[controller]")]
    [HttpPost]
    0 references
    public IActionResult CreateStudent()...

    [Route("api/[controller]")]
    [HttpDelete]
    0 references
    public IActionResult DeleteStudent()...

    [Route("api/[controller]")]
    [HttpPut]
    0 references
    public IActionResult UpdateStudent()...
```

# 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 de 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:

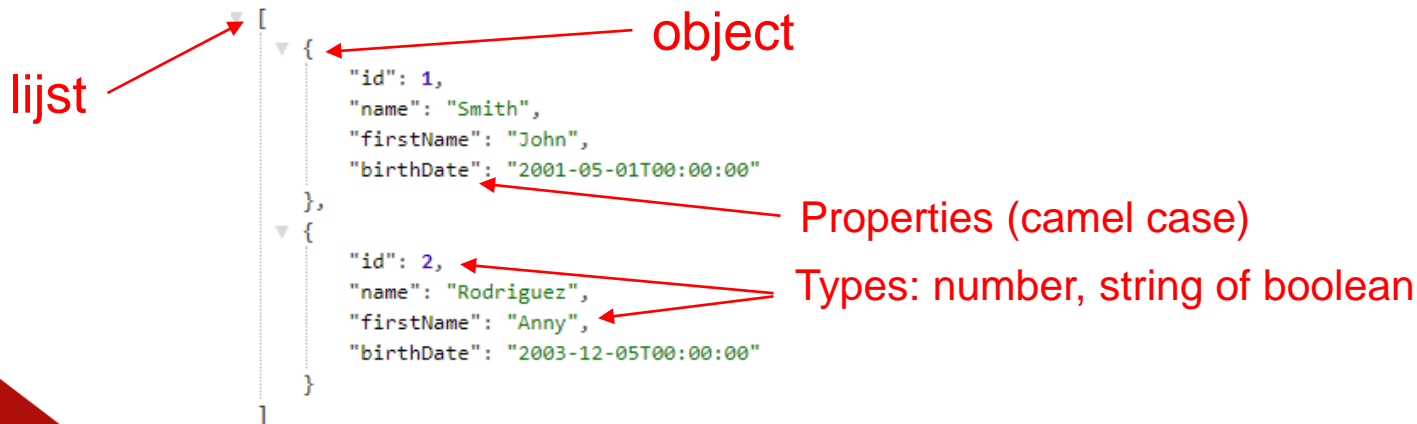
```
[
  {
    "id": 1,
    "name": "Smith",
    "firstName": "John",
    "birthDate": "2001-05-01T00:00:00"
  },
  {
    "id": 2,
    "name": "Rodriguez",
    "firstName": "Anny",
    "birthDate": "2003-12-05T00:00:00"
  }
]
```

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`
- Request Headers (17)**

# JSON formaat

- **JSON** = Javascript **O**bject **N**otation
- Vandaag de dag het meest gebruikte formaat in web API's om gegevens uit te wisselen.
- Het ASP framework zal alle lijsten of objecten die we terug geven vanuit een action 'by default' omzetten naar JSON formaat.
  - Zal deze in de **body** van de **response** plaatsen
  - Zal het **content-type** in de response header 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's opening curly brace `{`.
- Properties (camel case)** points to the `birthDate` property in the first object.
- Types: number, string of boolean** points to the values `1` (number) and `"Anny"` (string) in the second object.

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



# Controller: status code

- Als we vanuit een Action een resultaat willen terugsturen moeten we daarnaast ook de **status code** kunnen instellen.
- Hiervoor zijn er op de base class (ControllerBase) een aantal **helper** methodes aanwezig
  - **Ok ()** => status code 200
  - **NotFound ()** = status code 404
  - **BadRequest ()** = status code 400
  - ...

```
[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

# Zelf status code instellen in het “ActionResult”

- Als je geen helper methode terugvindt kan je ook zelf de status code instellen, gebruik hiervoor dan het **ContentResult**

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

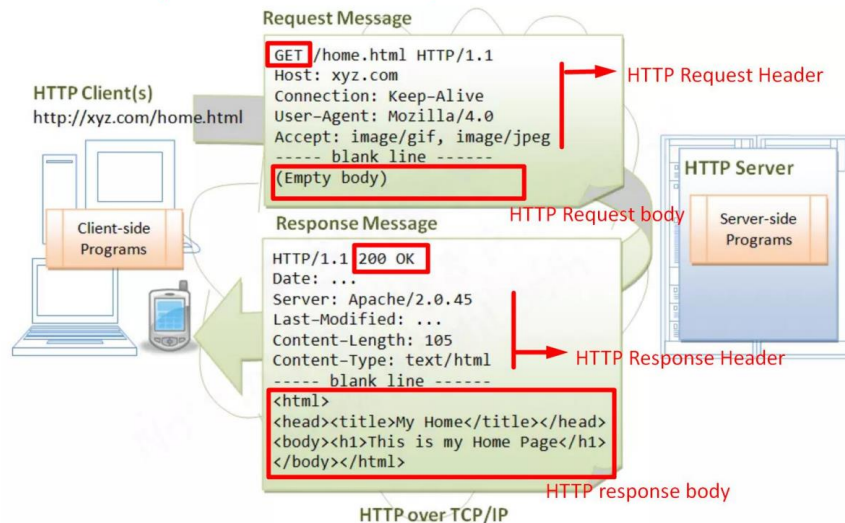
    //
    // Summary:
    //     Gets or sets the content representing the body of the response.
    public string Content { get; set; }
    //
    // Summary:
    //     Gets or sets the Content-Type header for the response.
    public string ContentType { get; set; }
    //
    // Summary:
    //     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.

# Hoe kunnen we 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
  - **Key/value** pairs
- Via de body:
  - **Object**
  - **lijst** van objecten



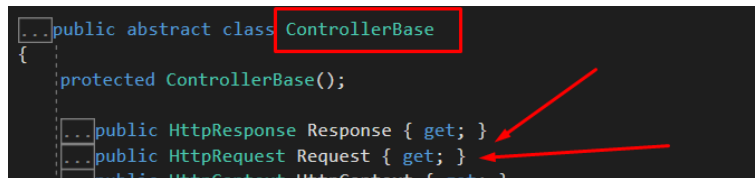
# Gebruik van de query parameters

- 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 de query parameters uitlezen in de controller ?

# Parameters uit de request halen

- ControllerBase bevat eveneens de properties:

- Request
- Response



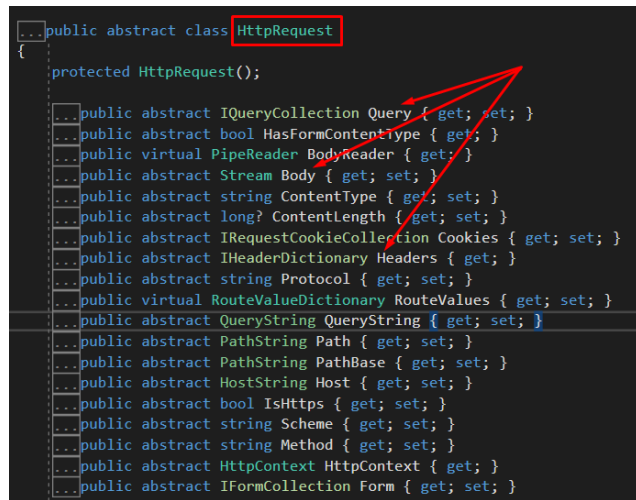
```
...public abstract class ControllerBase
{
    protected ControllerBase();

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

A code snippet showing the `ControllerBase` class. The class name is highlighted with a red box. Two red arrows point to the `Response` and `Request` properties, which are marked with `get; set;`.

- Via het **HttpRequest** object kunnen we vervolgens alle info uit de request halen:

- Query**: een lijst van alle query parameters
- Headers**: een lijst van alle header velden
- Body**: de inhoud van de body



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

    ...public abstract ICollection 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 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 Form { get; set; }
```

A code snippet showing the `HttpRequest` class. The class name is highlighted with a red box. Multiple red arrows point to various properties including `Query`, `BodyReader`, `Body`, `ContentType`, `ContentLength`, `Cookies`, `Headers`, `Protocol`, `RouteValues`, `QueryString`, `Path`, `PathBase`, `Host`, `IsHttps`, `Scheme`, `Method`, `HttpContext`, and `Form`, all of which are marked with `get; set;`.

# Uitlezen van 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 meer. We kunnen eenvoudigweg onze actie als volgt declareren.

- Optie 2:

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



# ASP.NET request handling

- Heel gelijkaardig kunnen we ook de header velden en de body uitlezen. Maar het ASP framework kan dit ook voor ons doen.
- We kunnen bovendien met **extra attributen** aangeven waar de info zich zou moeten bevinden.
  - [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

- En deze ?:
  - <http://localhost:xxxxx/api/students/1>
  - Hiermee vragen we enkel de student op met **Id = 1**
  - (zie later bij REST voor meer info hieromtrent)
- Zoals reeds aangegeven is het ook mogelijk om op 2 niveau's een route in te stellen.
- Dit kan van pas komen als we dergelijke URL wensen af te handelen.
- We gaan dus de route uitbreiden met een **Id**

# Route op controller **en** Action (2)

- Er zijn nu **2 actions** die via een **GET** 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 and Delete

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

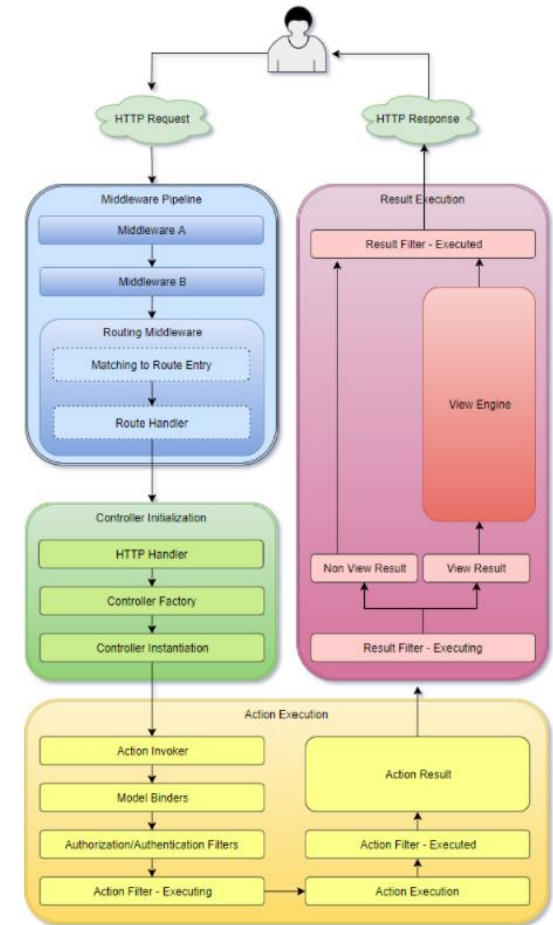
```
[HttpPost]
0 references
public IActionResult CreateStudent([FromBody] Person student)
{
    //TODO: add the student
    return Ok(people);
}

[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);
}
```

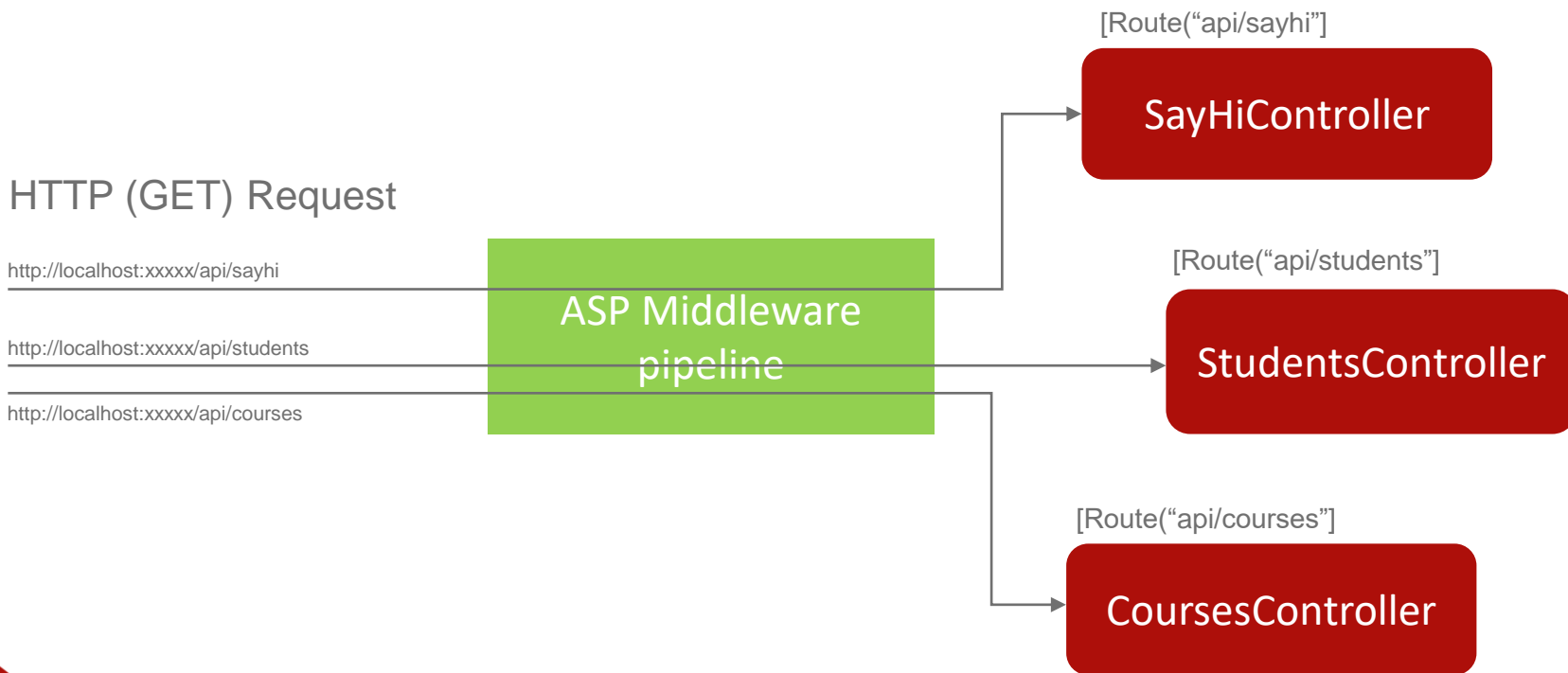
# Lifecycle van een controller

- Merk op dat we zelf nergens een “new” controller hoeven aan te maken.
- Dit wordt gedaan door het ASP framework
- Wanneer gebeurt dit nu echter juist ? En hoelang blijft een controller object bestaan ?



The ASP.NET Core MVC Pipeline

# Lifecycle van een controller (2)



\* Als je de standaard naamgeving volgt kan je als route gebruiken: `[Route("api/[controller]")]`

