howest
hogeschool

Module Web Backend

howest
hogeschool

Wat doen we deze week?

H1:.Net Core MVC inleiding
H2:Anatomie van een .Net Core MVC app
H2:Wat zijn Controllers binnen MVC
H2:Controllers gebruiken
H2:Wat zijn views
H2:Controllers en views gebruiken
H2:startup.cs, pipeline en statische files
H2:Wat is routing
H2:Routing gebruiken
Oefeningen

howest
hogeschool

Web Backend (.Net Core MVC)

Een inleiding

Inleiding: wat

- Het ontwikkelen van een databasegestuurde webapplicatie
- Gebruik maken van ASP.Net Core (.NET) en C#
- Focus ligt op server side programmatie
- HTML, CSS en C# taal en syntax komen minder aan bod
- Als **Software Design Pattern** gebruiken we het MVC model
- => Model View Controller

Inleiding: Hoe

- Naslagwerk voor de lessen is de cursus
- Tijdens contactmomenten
  - Theorie en voorbeelden
  - Veel oefenen (Veel!)
- Zelfstandig oefenen is zeer aan te raden!

Inleiding: Rules!

- Aanwezigheid in lessen is niet verplicht
- Maar als je aanwezig bent mee werk je mee!
- => geen Facebook, messenger, discord, instagram, reddit, 4chan,…
- PE opdrachten worden zelfstandig uitgevoerd!
- Respect voor mekaar en voor de lesgever
- Af en toe lachen kan en mag ☺

LES	Inhouden
Les1	Intro
Les2	Controllers + intro views+static files met duiding rond ingebouwde middleware
Les3	Routing
Les4	View models Views
Les5	View models en Views
Les6	Views+ razor syntax
Les7	Layouts, Tag helpers(razor syntax)
Les8	Partial views en view components
Les9	Partial views en view components
Les10	Formulieren
Les11	Formulieren
Les12	Formulieren
Les13	EF Core
Les14	EF Core
Les15	EF Core
Les16	EF Core
Les17	Dependency injection
Les18	State management
Les19	State management
Les20	Proefexamen

.NET

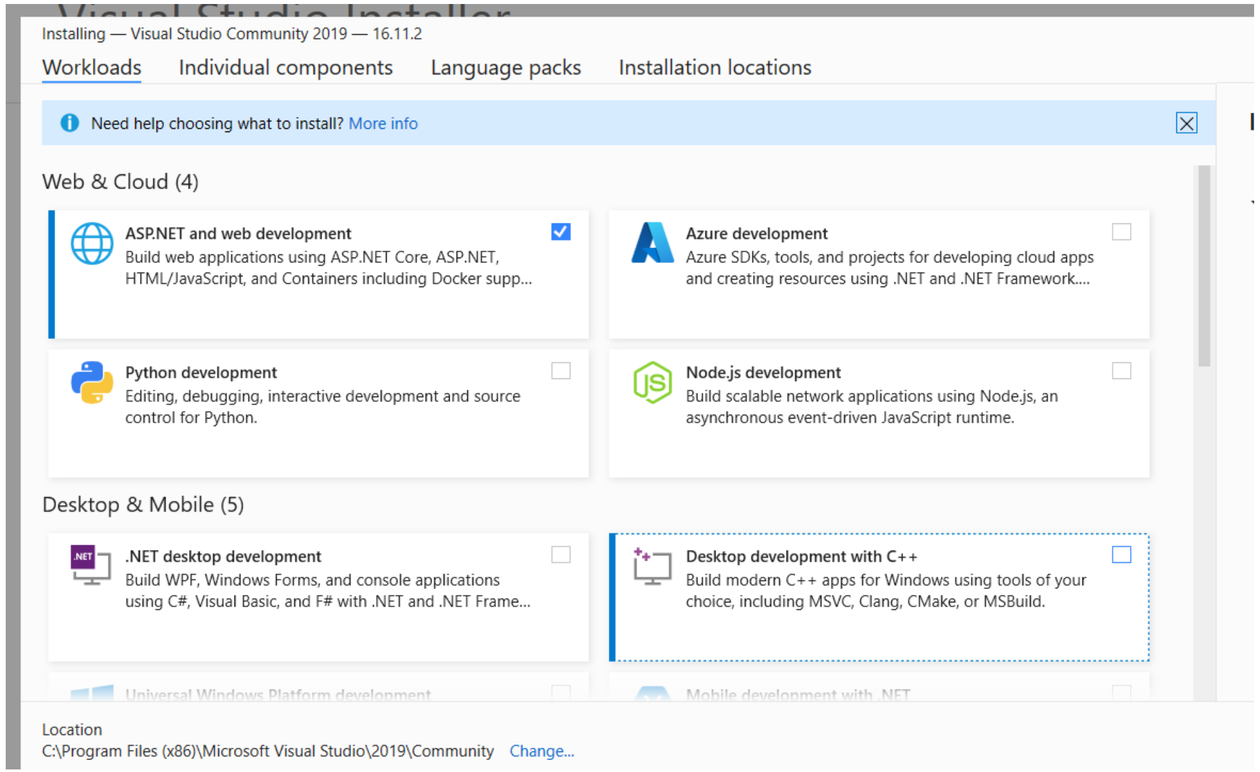

Build anything with a unified platform
CLOUD
WEB
DESKTOP
MOBILE
GAMING
IoT
AI
.NET
Visual Studio Family
Visual Studio Code
CLI
Windows
Linux
macOS
NuGet
GitHub
Components, tools, library vendors
TOOLS
OPERATING SYSTEM
ECOSYSTEM
.NET

.NET

- .NET bundelt verschillende omgevingen tot een **UNIFIED DEVELOPMENT PLATFORM**
- Wanneer je opzoekt zoek dan op *.NET Core MVC*
- Voordelen:
  - Cross Platform, Open Source
  - HTML zoals het hoort
  - HTTP by the book
  - Uitbreidbaar, testbaar
  - Verschillende web projecten mogelijk

Wat hebben we nodig?

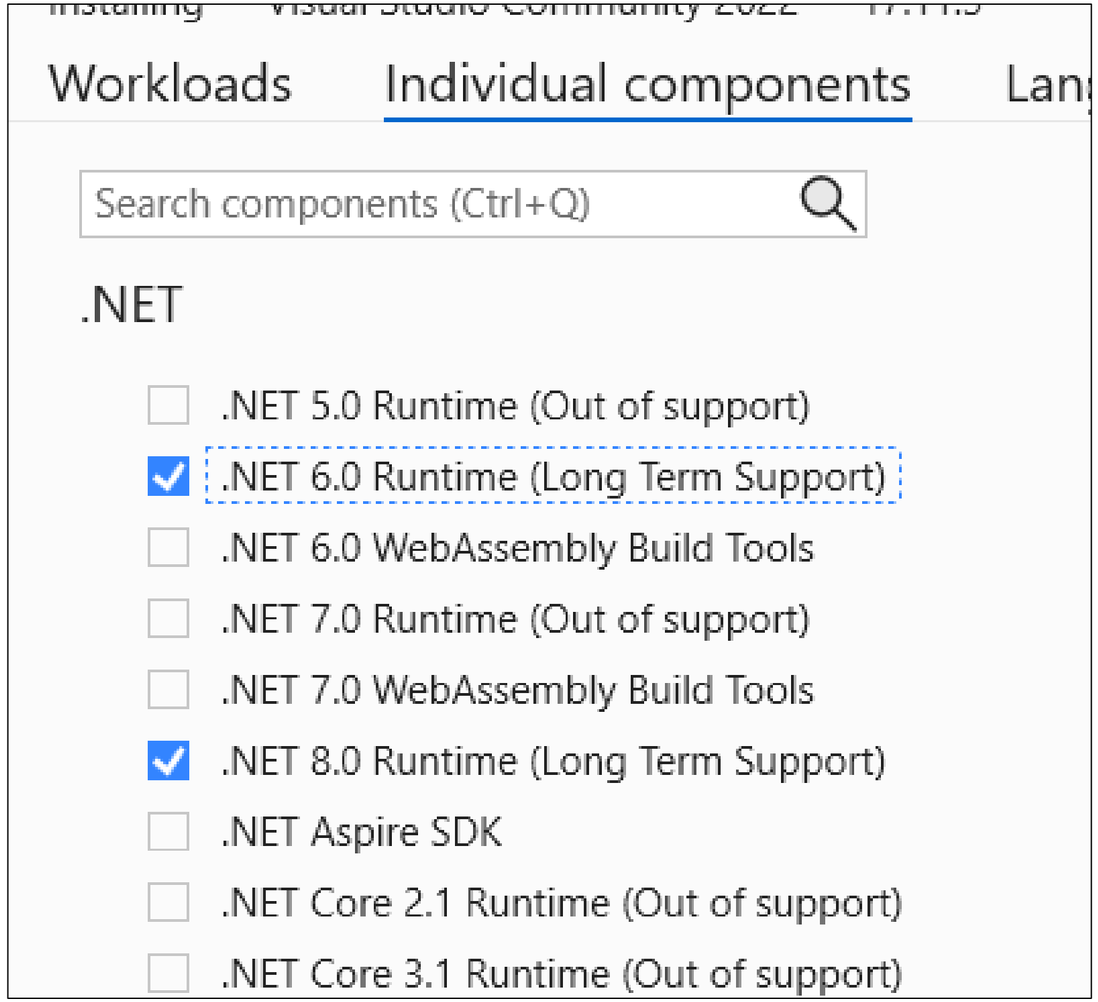
- Vs2022
- De ASP.Net development Workload
- Bij de installatie selecteren
- Of: Nadien toevoegen mbv de installer
- Versie: .Net LTS



howest hogeschool

Wat hebben we nodig?

- Vs 2022
- De ASP.Net development Workload
- Bij de installatie van VS 2022 selecteren
- Of: Nadien toevoegen mbv de installer
- **Versie: .Net LTS**



ASP.NET Core

- Uitbreidbaar en testbaar
  - C# OOP zorgt voor uitbreidbaarheid en structuur
  - We kunnen onafhankelijke componenten schrijven
  - We kunnen eenvoudige unit tests schrijven(module CIA)
- Keuze tussen verschillende **projecten/architectural patterns**
  - **MVC**: zorgt voor ***separation of control***. Makkelijk opdelen van de applicatie in afzonderlijke componenten
  - **Web API:** om REST-full webservices te bouwen die andere applicaties van data voorzien
  - **Razor Pages:** om een klassieke webapp te maken die opgebouwd is uit pagina's
  - **Razor Class Libraries:** om volledige stukjes UI en Business logic te verpakken in kleine, herbruikbare onderdelen

ASP.NET Core en WebServers

- ASP.Net applicaties zijn ***self hosted****:* ze beschikken over een eigen HTTP-server implementatie
- ASP.Net bevat 2 implementaties van een HTTP-server:
  - **Kestrel**:
  - ➢ Een snelle, eenvoudige, cross-platform HTTP-server. Voor een intranet applicatie is dit meestal voldoende
  - ➢ IIS

    Kan gebruikt worden voor zowel intranet als internet

Deze module focust op Web Development, niet op deployment

ASP.NET Core Ontwikkelomgeving

- Uiteraard gebruiken we Visual studio om onze WebApps te ontwikkelen
- => WebApps kunnen we binnen Visual studio op 2 manieren starten:

1. **Rechtstreeks**

Via de rechtstreekse uitvoering wordt in de achtergrond het **dotnet run** commando uitgevoerd, je ziet een command line venster waarin de kestrel server opgestart wordt

2. **Via IIS express**

Dit start een instantie van IIS express op die als reverse proxy fungeert voor de kestrel webserver, nuttig voor ssl configuratie en windows authenticatie te implementeren

howest
hogeschool

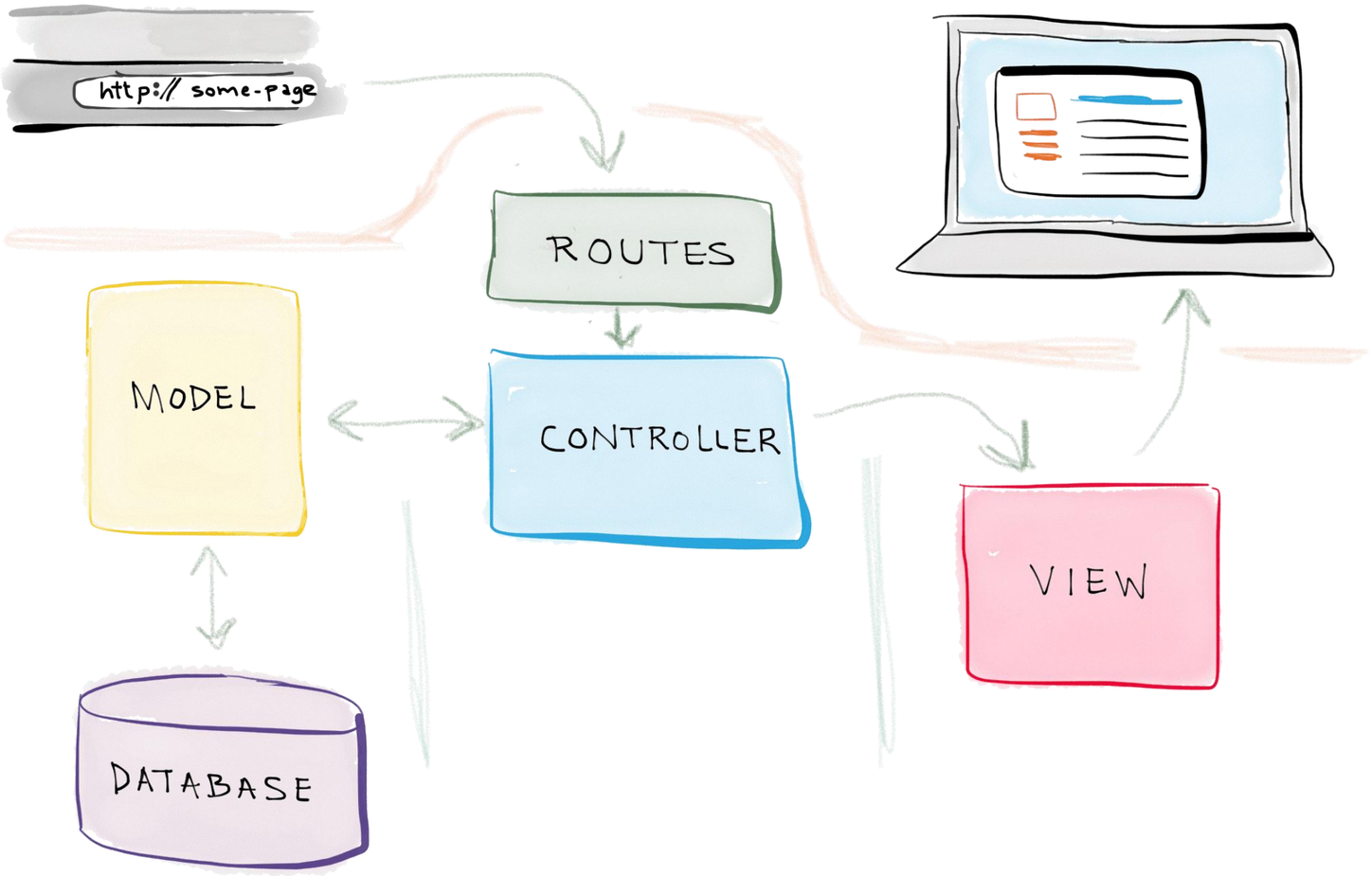
MVC

Nog een klein beetje theorie…

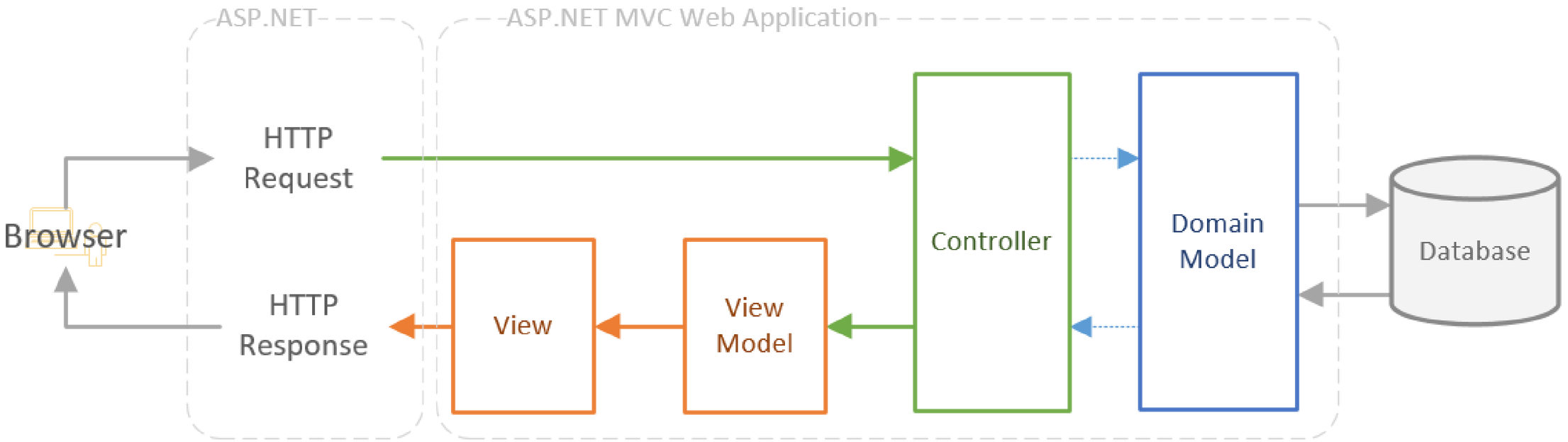
Wat is MVC?

- **MVC = Model – View – Controller**
- **= Design pattern**
- **Model** => Modelleert data, verzorgt de interacties met de data(database of API)
- **View** => bevat de front end files van de applicatie en gaat in interactie met de gebruiker
- **Controller** => Capteert de **requests** die via de views door een gebruiker gemaakt worden en bevraagt de **models** om vervolgens het resultaat door te sturen naar de correcte **views**.
- De verbinding tussen de gebruiker en de models, views en controllers wordt gemaakt door middel van **Routes**

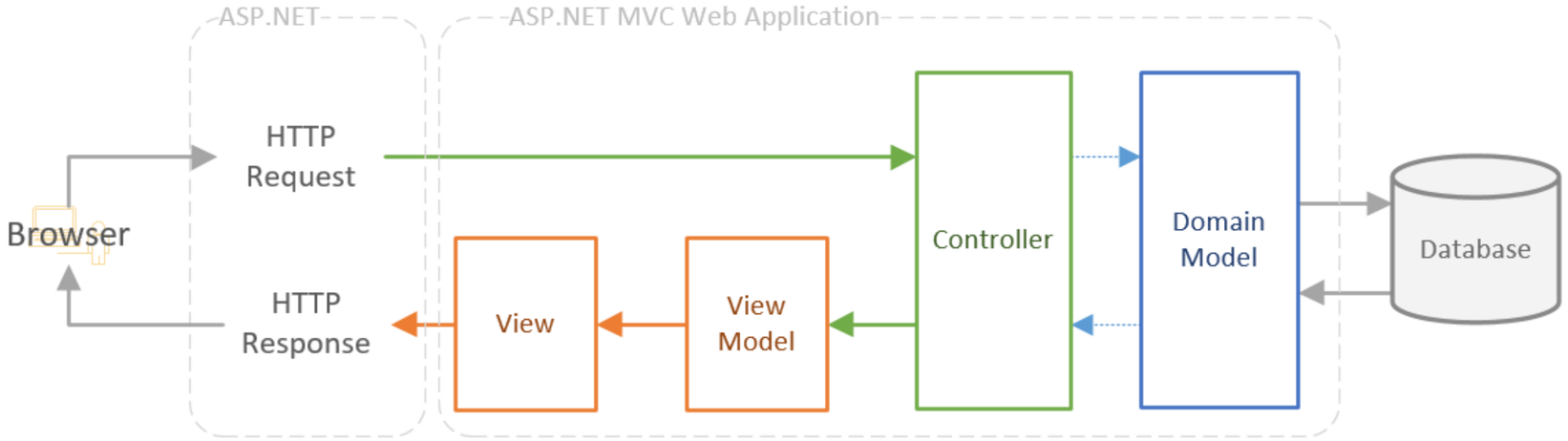
MVC Generic schema



MVC .Net Core schema



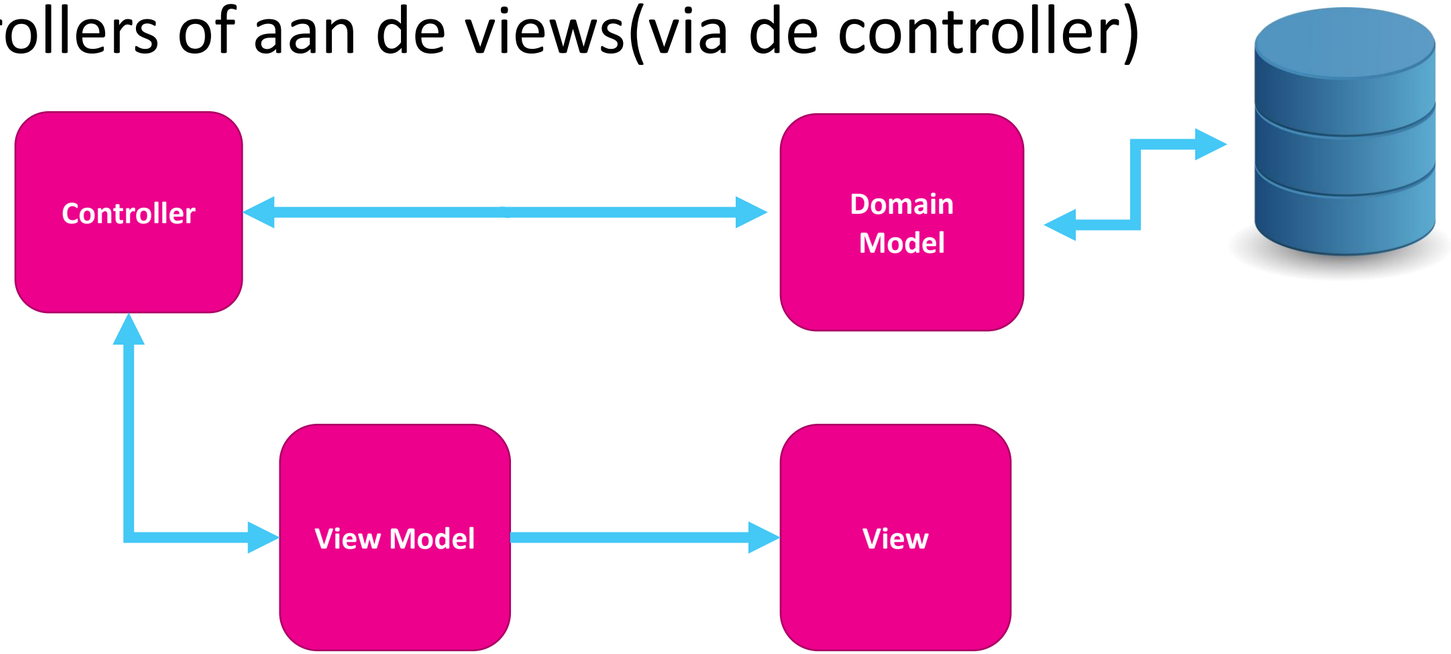
MVC .Net Core werking



- De **Controller** staat centraal
- De gebruiker tikt een url in die naar een **Route** verwijst
- Adhv de **Route** wordt bepaald welke **Controller** er opgeroepen wordt
- De **Controller** zal indien nodig een **Domain Model** aanspreken om data op te halen
- De Controller zal de data modelleren in een **View Model** en doorgeven aan een **View**
- De **View** zal de data displayen aan de gebruiker

MVC .Net Core werking: **Models**

- Een Model bevat gegevens die door de applicatie verwerkt worden
  - Vb, een lijst schoenen op bol.com, info over een game op steam, …
- Ze leveren data aan de controllers of aan de views(via de controller)
- Het zijn gewone C# klassen
- Er bestaan 2 soorten:
  - **Domain Models**
  - **View Models**

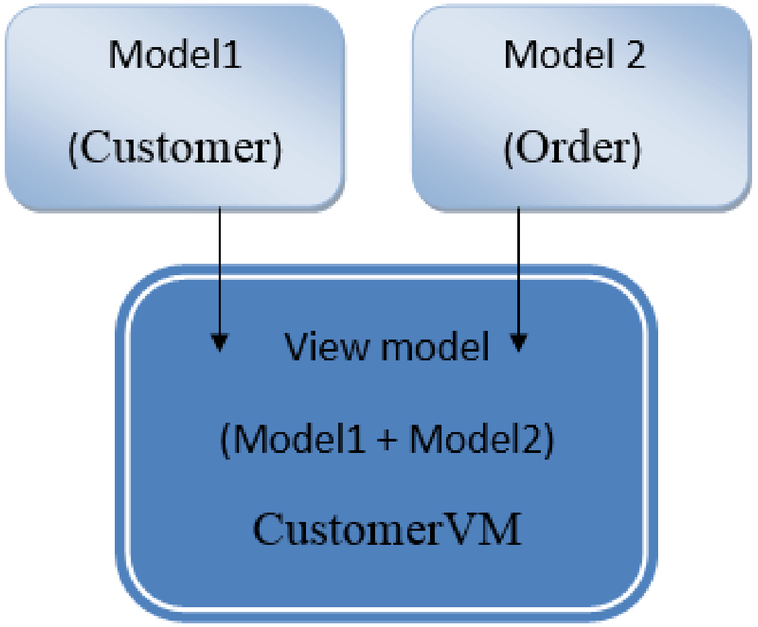


MVC .Net Core werking: **Models**

- **Domain Models**
  - Stellen de Business Logic voor
  - Benaderen gegevens uit externe bronnen(DB, API,…)
  - Vb bol.com heeft een database met producten en categoriëen
    - => een MVC Domain model zou een klasse product en een klasse categorie bevatten
  - Via deze klassen kunnen we de db aanspreken
- **View Models**
  - Bevatten data die de view nodig heeft om html te genereren voor de gebruiker
  - Data wordt aangeleverd via de controller
  - Als data uit een DB komt dan haalt de controller dit uit een Domain Model
  - De view gebruikt het View Model vervolgens om de gegevens te displayen

MVC .Net Core werking: **View Model vs Domain Model**

- Een **Domain Model** bevat gegevens die door de applicatie verwerkt worden
  - Vb, een lijst schoenen op bol.com, info over een game op steam, …
  - Vb klantgegevens
- De **Controller** haalt deze data op en plaatst die in 1 **View Model** (CustomerVM)
- De **Controller** geeft **CustomerViewModel** door aan de **View**
- De **View** toont de data aan de gebruiker dmv HTML



MVC .Net Core werking: **Controllers**

- Staan **Centraal** in een MVC structuur
- Bepalen welke operaties moeten worden opgeroepen bij de **Domain Models**
- Bepalen welke informatie daaruit nodig is voor de view en plaatsen die in **View Models**
- Speelt de data door naar de juiste **View** die de data omvormt naar HTML formaat
- **Vb:** je surft naar bol.com en vraag een lijst van Nike schoenen op link
  - **Controller** roept operaties op bij de **Domain Models** om een lijst van Nike schoenen op te halen
  - **Controller** plaatst de lijst in een **View Model** object(of een List van **View Model**)
  - **Controller** roept de correcte **View** aan en geeft het **View Model** door aan die view
  - **View** genereert HTML lijst op basis van data in **View Model**
  - **Gebruiker** klikt op gewenst schoen en alles wordt herhaald ☺

MVC .Net Core werking: **Views**

- Meest eenvoudige onderdeel van een **MVC** app
- De **View** genereert HTML voor de browser op basis van de gegevens in een **View Model**
- Dit is dus het gedeelte waarmee de gebruiker rechtstreeks in contact komt!
- Later meer hierover

MVC .Net Core werking: Wie mag wat doen?

- Er zijn **restricties** op wat elk onderdeel van **MVC** mag doen
- **Domain Model:**
  - Enkel de Business Logica bevatten en
  - operaties op die gegevens, aanmaken, manipuleren, verwijderen
- **View Model:**
  - Mag enkel de gegevens bevatten die voor de view bestemd zijn
  - Geen operaties op die gegevens!
- **Controller:**
  - Data uit de **Domain Models** oproepen, of doorgeven
  - Transformaties uitvoeren op de verkregen data en omzetten naar een **View Model**
  - De gewenste **View** selecteren en de **View Model** doorgeven aan die **View**
- **View:**
  - **Mag enkel de data uit een View Model presenteren via HTML aan de gebruiker**

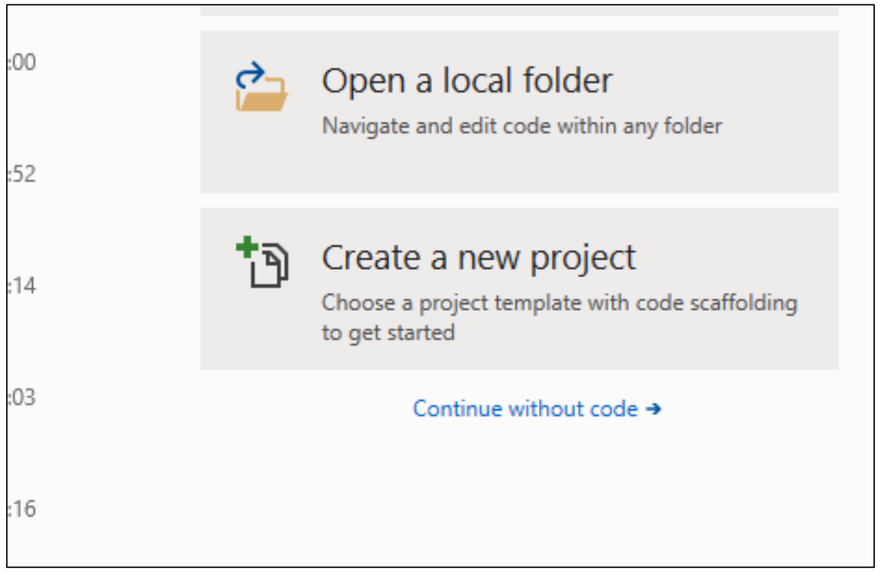
howest
hogeschool

Anatomie van .Net Core MVC

Een eerste MVC app

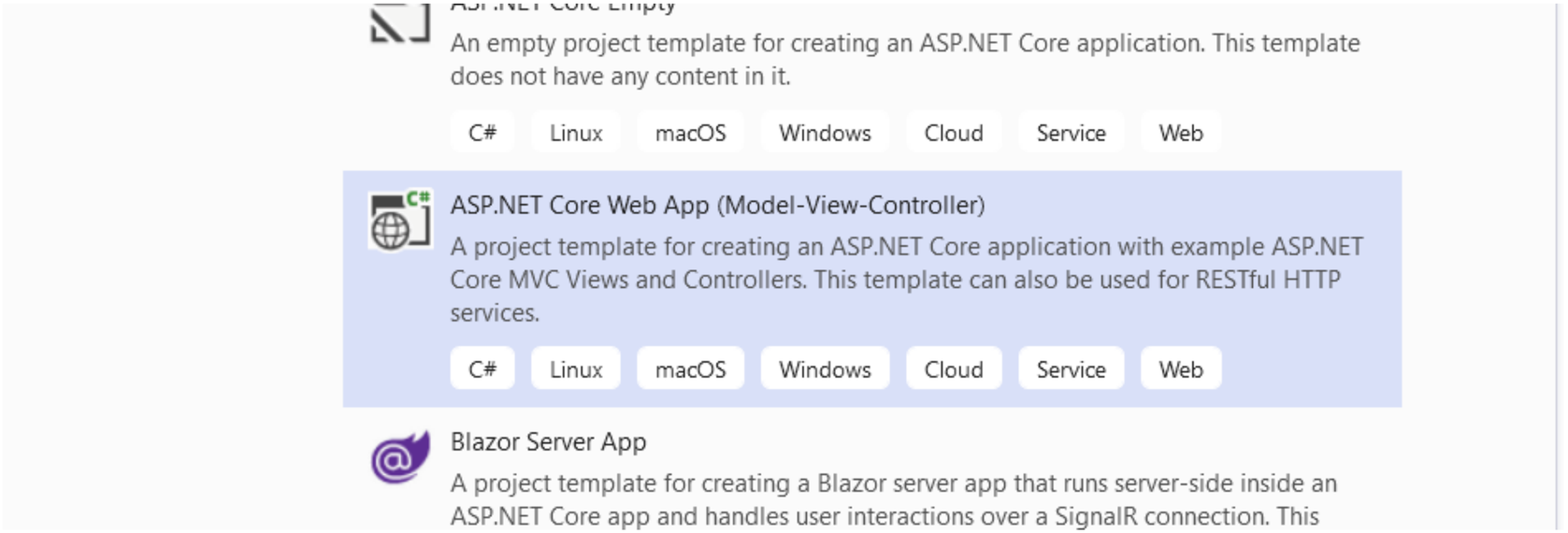
Anatomie van .Net Core MVC: Een eerste MVC app

- Maak nu zelf een eerste .Net Core MVC app aan:
- Start Visual Studio op
- Maak een nieuw project aan:



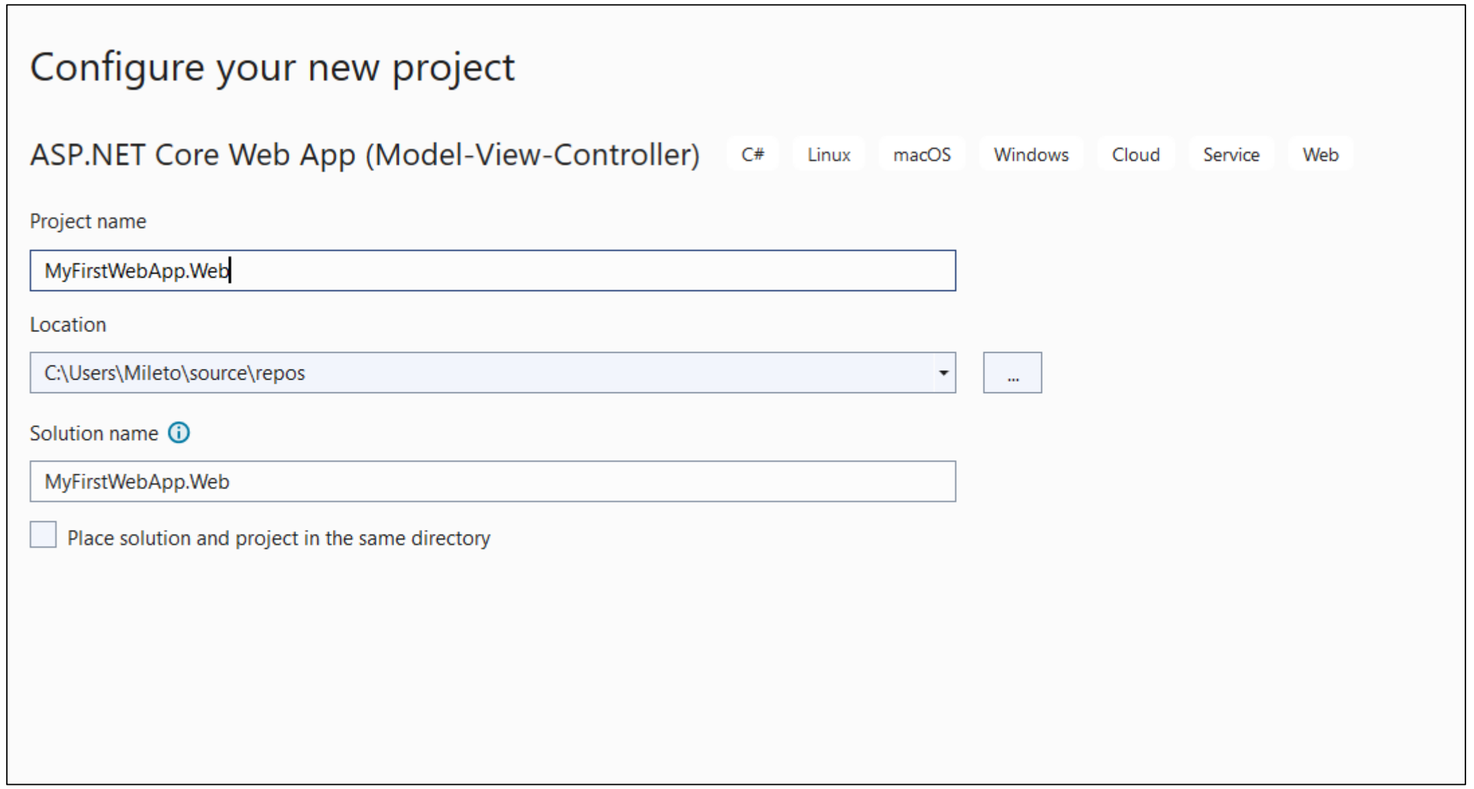
Anatomie van .Net Core MVC: Een eerste MVC app

- Selecteer **ASP.NET Core Web App (Model-View-Controller)** en klik **Next**



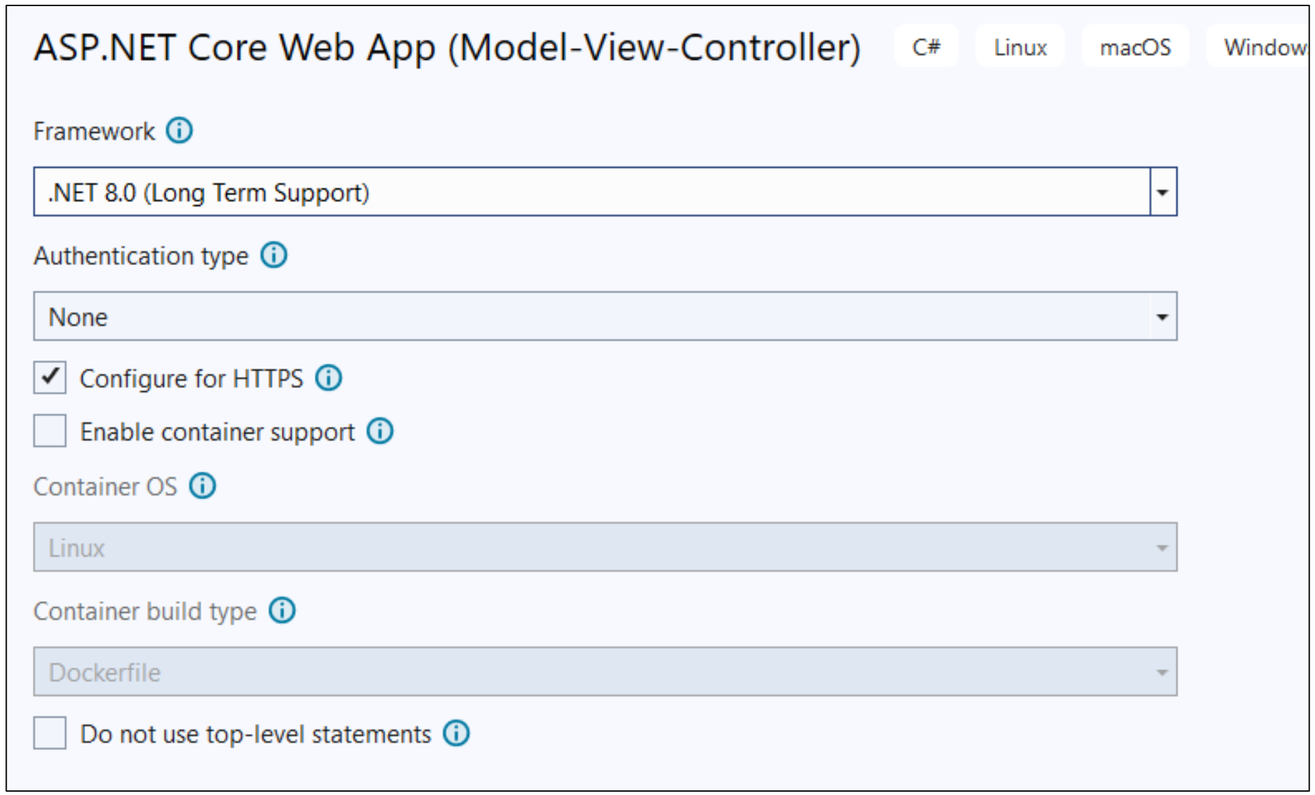
Anatomie van .Net Core MVC: Een eerste MVC app

- Geef een naam in voor jouw project vb **MyfirstApp.Web** en klik op **Next**



Anatomie van .Net Core MVC: Een eerste MVC app

- Selecteer **.NET … (Long-term support versie)**
- Klik op **Create**

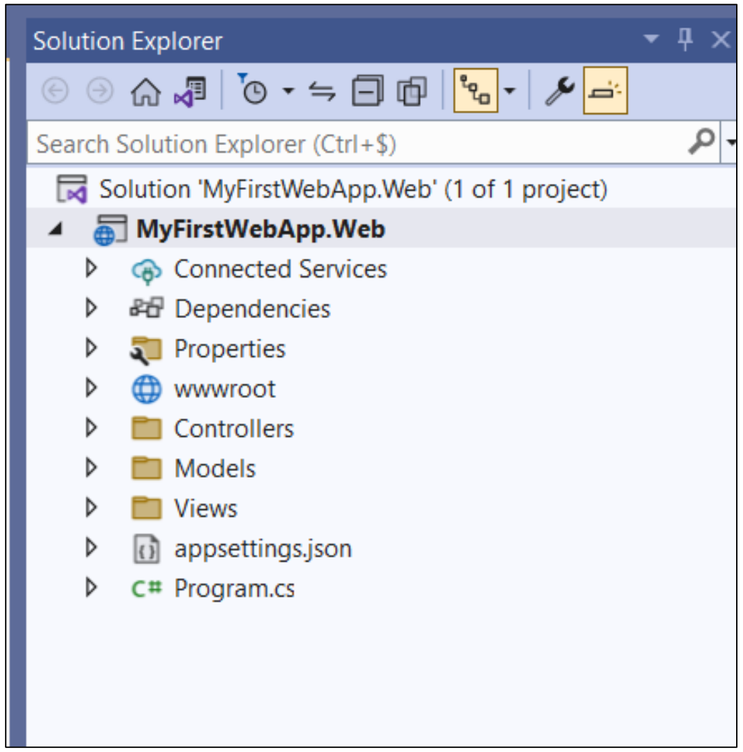


Anatomie van .Net Core MVC: Een eerste MVC app

- Nullable or not?
- Standaard staat de Nullable Context op **enabled**
- Kan soms vervelend zijn, gezien Web apps onvoorspelbaar zijn en vaak Null waarden kunnen bevatten
- We zetten dit af door onderstaande code in de project file te verwijderen

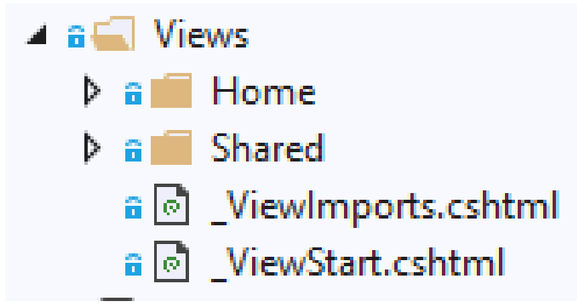
```
RsộřêstƴỳǦsộụř
    ĮắsĝệtƴGsắṇêxộsl ŋêtƴ`   ĮắsĝệtƴGsắṇêxộsl
    Nụľľắčľê êŋắčľê  Nụľľắčľê
    ÍṇřľîçîtƴÛṣîŋĝṣ êŋắčľê  Íṇřľîçîtƴ Ûṣîŋĝṣ
 RsộřêstƴỳǦsộụř
```

Anatomie van .Net Core MVC: mappenstructuur



Anatomie van .Net Core MVC: mappenstructuur

- Elk onderdeel heeft een vaste plaats binnen MVC
- **wwwroot**: alle statische bestanden (afbeeldingen, javascripts, css, …) die door de views gebruikt worden
- **Controllers**: is de **standaardlocatie** voor al de controllers
- **Views:** views **moeten** in deze map geplaatst worden, de **ViewEngine** zoekt naar views in deze map
  - 2 zoekmechanismen in deze map:
    - Een map met **dezelfde naam** als de controller
    - Een map met de naam **Shared**



howest
hogeschool

Anatomie van .Net Core MVC

Controllers

Anatomie van .Net Core MVC: Controllers

- **Controllers** zijn klassen die de browser requests afhandelen
- Verzamelen gegevens via de **Domain Models**
- Roepen de html sjablonen**(Views)** op voor de browser
- Bevatten 1 of meerdere public methoden die opgeroepen worden door een **request**(lees: url)
- **Vb:**
  - Wanneer we surfen naar https://localhost:57872/Home/Index wordt de index methode van onze **HomeController** aangeroepen
  - Het gedeelte **Home/Index** noemen we een **Route**

Anatomie van .Net Core MVC: Controllers

```
public class HomeController : Controller
{
        public IActionResult Index()
        {
            return View();
        }
}
```

- **= http://localhost:57872/Home/Index**

Anatomie van .Net Core MVC: Controllers

Let op!

- Een controller eindigt altijd op **controller => vb. ProductsController**
- Een controller erft (bijna) altijd over van de klasse **Controller**
- public IActionResult Index() = een **ActionMethode => retourneert een object van het type IActionResult**
- return view(): geeft de opdracht om de standaard **View** te laden in de map views/Product/
- Een **return** in een controller betekent zoveel als *geef iets terug aan de browser*
- Een overzicht van andere methoden die een **IActionResult** object teruggeven vind je op de volgende slide

Anatomie van .Net Core MVC: Controllers

Methode	**De browser krijgt…**
Content()	De inhoud van de string die meegegeven wordt als parameter.
Redirect()	Een 302 Redirect, dat de browser instrueert om naar een andere URL te navigeren.
RedirectPermanent()	Een 301 Redirect, dat vertelt de browser deze URL te vergeten en steeds de opgegeven alternatieve URL te gebruiken. Raadzaamheid is geboden bij gebruik hiervan!
Json()	Een object terug in JSON-formaat. Ideaal voor asynchrone javascript requests.
File()	De inhoud van een bestand aan in binaire vorm. Dit initieert een download in de browser.
NotFound()	Een HTTP 404 resultaat dat de browser vertelt dat gevraagde bron niet gevonden werd.
BadRequest()	Een HTTP 400 resultaat, bv. wanneer er een vereiste URL-parameter ontbreekt.

howest
hogeschool

Anatomie van .Net Core MVC

Views en Layouts

Anatomie van .Net Core MVC: **Views** en Layouts

- **Views** behandelen de data die je aan de gebruiker wilt tonen
- Ze zijn opgebouwd als **sjablonen** die door controllers aangeroepen worden
- **.cshtml** => c sharp html
- Ze worden in een map geplaatst met als naam de corresponderende **Controller**
- Ze hebben dezelfde naam als hun corresponderende **action methode**

Anatomie van .Net Core MVC: **Views** en Layouts

- De inhoud van een view bevat nochtans zeer weinig HTML??

```
@{
    ViewData["Title"] = "Privacy Policy";
}
<h1>@ViewData["Title"]</h1>

<p>Use this page to detail your site's privacy policy.</p>
```

- De view bevat enkel de specifieke data voor die functionaliteit => in dit geval de privacy policy

Anatomie van .Net Core MVC: **Views** en Layouts

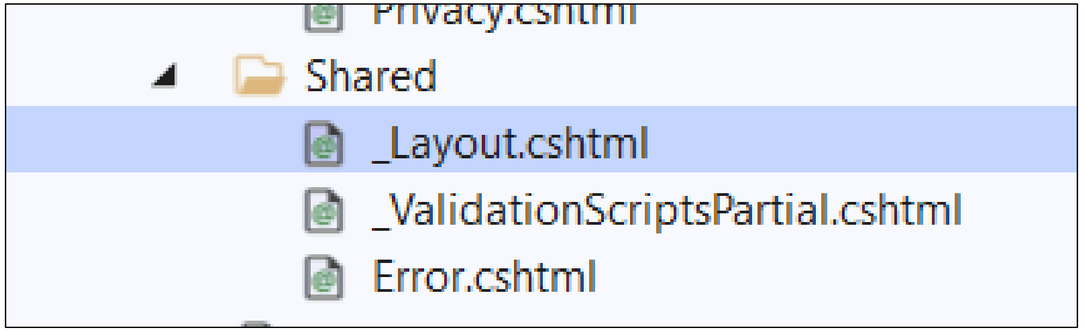
- Start de applicatie met F5
- Surf naar de link **Privacy**
- Bekijk de paginabron Ctrl-U
- Wat merk je?

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Privacy Policy - MyFirstApp</title>


        <link rel="stylesheet" href="/lib/bootstrap/dist/css/bootstrap.css" />


    <link rel="stylesheet" href="/css/site.css" />
</head>
<body>
    <header>
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">
            <div class="container">
                <a class="navbar-brand" href="/">MyFirstApp</a>
                <button class="navbar-toggler" type="button" data-toggle="collapse" data-target=".navbar-collapse" aria-controls="navbarSupportedContent"
                        aria-expanded="false" aria-label="Toggle navigation">
                    <span class="navbar-toggler-icon"></span>
                </button>
                <div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">
                    <ul class="navbar-nav flex-grow-1">
                        <li class="nav-item">
                            <a class="nav-link text-dark" href="/">Home</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" href="/Home/Privacy">Privacy</a>
                        </li>
                    </ul>
```

howest hogeschool

Anatomie van .Net Core MVC: Views en **Layouts**

- **Views** halen hun generische HTML uit **Layout** files
- Een **Layout** is een gemeenschappelijk HTML sjabloon
- Wordt gebruikt in alle **View** files
- Belangrijke files:
  - **_Layout.cshtml**
  - **_ViewStart.cshtml**

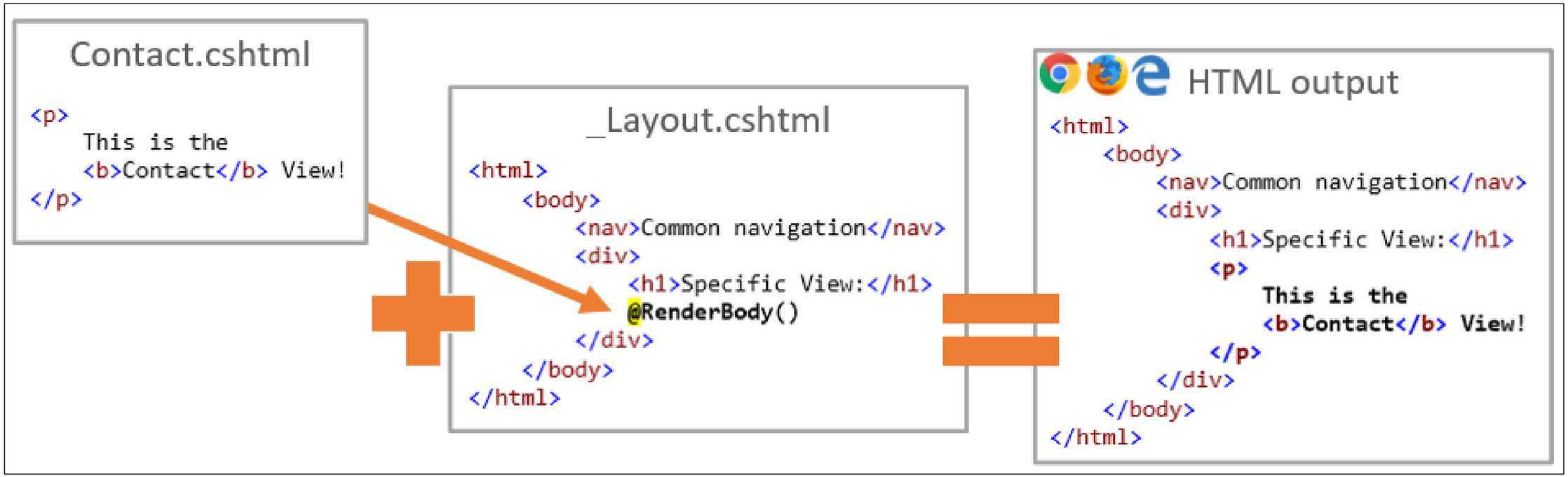


Anatomie van .Net Core MVC: Views en **Layouts**

- **_Layout.cshtml**
- Bevat de **gemeenschappelijke** HTML en opmaak die de look en feel van je app bepalen
- Ergens middenin vinden we de methode **@RenderBody()**
- Die zorgt ervoor dat de **View** (bv Privacy) geïnjecteerd wordt in de gemeenschappelijke HTML
- **_ViewStart.cshtml**
- Bevat een verwijzing naar het default gemeenschappelijke sjabloon

Anatomie van .Net Core MVC: Views en **Layouts**

- **Schematisch:**



howest hogeschool

Anatomie van .Net Core MVC: Oefening Controllers

- Maak de oefening over Controllers en Action Results
  - h02-1-1.controllers_actionresults

- De opgave vind je op Leho

howest
hogeschool

Anatomie van .Net Core MVC

Program.cs en de Pipeline

Anatomie van .Net Core MVC: **Program.cs**

- Zeer belangrijke file in de MVC applicatie
- Bepaalt de functionaliteiten van je applicatie
- Bevat twee belangrijke classes/instances:
- ***WebApplicationBuilder => builder***
  - Initialiseert ingebouwde of zelfgeschreven services zodat ze overal in je applicatie beschikbaar zijn(later meer)
- ***WebApplication => app***
  - Definieert de *pipeline* van je applicatie
  - Alle HTTP requests passeren doorheen deze methode

Anatomie van .Net Core MVC: Startup.cs

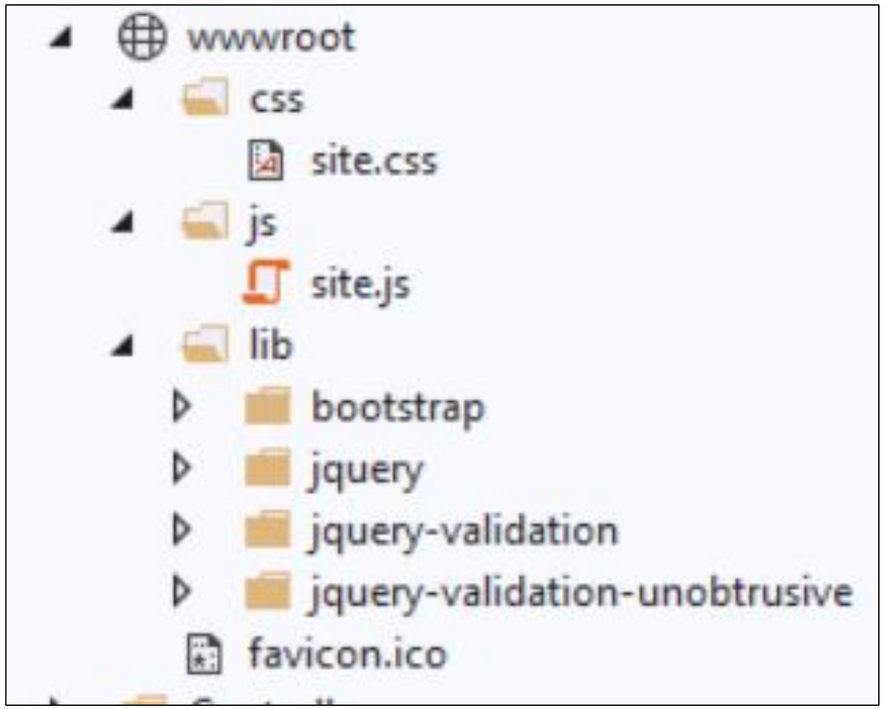
- In de ***pipeline*** wordt beslist wat er met een request moet gebeuren
- De componenten binnen deze request noemen we **middleware**
- Wanneer we een standaard MVC app openen vinden we de volgende middleware in de **Program.cs** class

Anatomie van .Net Core MVC: Startup.cs

```
ŵắs ắřř   čụîľđês Bụîľđ

   Cộŋğîĝụsê ţhê ȞŢŢR sêřụêşţ řîřêľîŋê
îğ   ắřř Éŋŵîsộŋṇêŋţ ÍşDêŵêľộřṇêŋţ

    ắřř ÛşêÉỵçêřţîộŋȞắŋđľês   Ȟộṇê Éssộs
       Ţȟê đêğắụľţ ȞŞŢŞ ŵắľụê îş ˛˳ đắỳş  Ÿộụ ṇắỳ xắŋţ ţộ çȟắŋĝê ţȟîş ğộs řsộđụçţîộŋ şçêŋắsîộş  şêê
ȟţţřş   ắľắ ṇş ắşřŋêţçộsê ȟşţş
    ắřř ÛşêȞşţş


ắřř ÛşêȞţţřşŖêđîsêçţîộŋ
ắřř ÛşêŞţắţîçGîľêş

ắřř ÛşêŖộụţîŋĝ

ắřř ÛşêAụţȟộsîċắţîộŋ

ắřř ŇắřCộŋţsộľľêsŖộụţê
    ŋắṇê   đêğắụľţ
    řắţţêsŋ    çộŋţsộľľês Ȟộṇê   ắçţîộŋ Íŋđêỵ   îđ

ắřř Ŗụŋ
```

Anatomie van .Net Core MVC: statische files

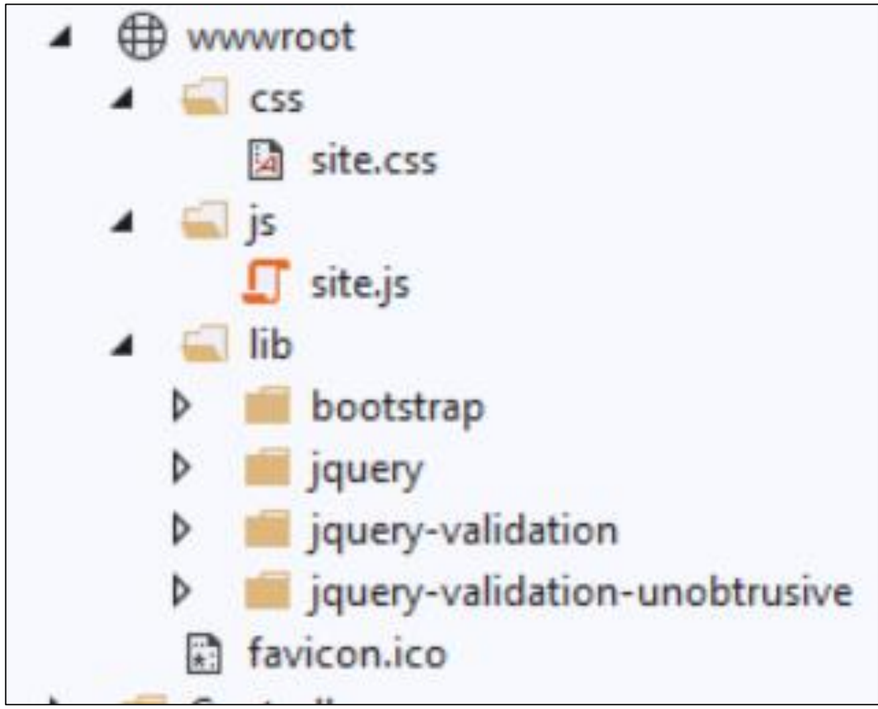
- Statische files = afbeeldingen, opmaak(css) en client side code(javascript)
- Deze worden in de map **wwwroot** geplaatst onder de corresponderende map



howest hogeschool

Anatomie van .Net Core MVC: statische files

- We merken reeds bestaande files in deze map
- Je kan daar je eigen opmaak toevoegen/vervangen
- Je kan er ook je eigen JavaScript code plaatsen
- Of bootstrap
- De statische files worden geleverd door de app.**UseStaticFiles()** methode



Anatomie van .Net Core MVC: statische files

- Klasoefening:
  - plaats je de **UseStaticFiles()** methode in commentaar
  - Start de app en refresh de pagina cache let Ctrl-F5
  - Wat merk je?

howest
hogeschool

Anatomie van .Net Core MVC

Routing

Anatomie van .Net Core MVC: Routing

- HTPP Requests passeren dus langs middleware in onze **Program.cs class** om aan de juiste acties te voldoen
- **Routing** wordt door een overload van de middleware **MapControllerRoute()** geregeld
- Een **Route** wordt **gemappet** naar een overeenkomende **Controller** en **action methode**
- **Vb:** https://localhost:5001/Home/Privacy
- Wordt door de **pipeline** doorgestuurd naar de **HomeController** en naar de methode `public IActionResult Privacy()`

Anatomie van .Net Core MVC: Routing

- **Route templates**
- **=** een patroon/sjabloon om routes te mappen aan controllers en actionmethodes
- Er moet **minstens 1 routetemplate** gedefinieerd zijn
- MVC apps definiëren hun **standaardroute** als volgt:

```
ǎřř ŇǎřCộŋƫsộľľês�collectionRộụƫê
    ŋǎṇê   đêğǎụľƫ
    řǎƫƫêsŋ    çộŋƫsộľľês Ňộṇê   ǎçƫîộŋ Íŋđêɏ   îđ
```

Anatomie van .Net Core MVC: Routing

- `"{controller=Home}/{action=Index}/{id?}`
- `Template = controller/action/optionele id`
- `=>` https://localhost:44371/Home/Privacy/1
- Het derde deel *{id?}* gebruiken we om een parameter door te geven aan een controllermethode
- Wanneer een controller en/of actionmethod niet bestaan krijgen we een 404 error
- De standaardwaarden zorgen ervoor dat er bij een request zonder action standaard de methode *Index()* wordt geladen.

Anatomie van .Net Core MVC: Eigen routes maken

- Vaak kan het nuttig zijn om eigen routes te definiëren
- Dit kan perfect binnen de **app.MapControllerRoute()** methode
- De eigen routes komen vaak **voor** de algemene default route te staan.
- Stel: we willen 2 zoekmethodes implementeren
  - **searchByIndex** met een parameter Id, type int (default)
  - **searchByString** met een parameter searchString, type string

Anatomie van .Net Core MVC: Eigen routes maken

```
public IActionResult SearchById(int id)
{
   var result = $"<b>You asked for Id:{id}</b>";
   return Content(result, "text/html");
}

public IActionResult SearchByString(string needle)
{
   var result = $"<b>You looked for:{needle}</b>";
   return Content(result, "text/html");
}
```

- We testen onderstaande routes uit met onze default route template
- **/home/searchByID/5**
- **/home/searchByString/bob**
- **/home/searchByID?id=5**
- **/home/searchByString?needle=bob**

Anatomie van .Net Core MVC: Eigen routes maken

- Wat merken we?
  - Enkel de lelijke querystring versie en de klassieke default met Id lijken te werken
- Oplossing: een nieuwe route definiëren die rekening houdt met onze parameter **needle**
- **De defaults is hier nodig!**
- Nu kunnen we de route met searchString parameter wel gebruiken

```
app.MapControllerRoute(
   name: "SearchByString",
   pattern: "Home/SearchByString/{needle}",
   defaults: new {Controller="Home",Action="SearchByString"}
   );
```

Anatomie van .Net Core MVC: Eigen routes maken

- **Route constraints:**
  - De parameters van een route kunnen we ook verplichten om van een bepaald type te zijn
  - Op deze manier kunnen we gelijke routes **differentiëren**
  - **Bv.** we willen één enkele search Url die kan zoeken op Id (int) of op Needle (string), naargelang het type
  - Naargelang het type zullen we doorgestuurd worden naar de juiste methode
    - **searchByIndex** of **searchByString**
  - We passen hiervoor onze **routemapping** aan als volgt:

Anatomie van .Net Core MVC: Eigen routes maken

- **Route constraints:**
  - We beginnen met de specifieke **SearchById** waar we opleggen dat Id een int moet zijn
  - Is dit niet het geval dan zal de string versie in werking treden

```
app.MapControllerRoute(
   name: "SearchById",
   pattern: "Home/Search/{id:int}",
   defaults: new {Controller="Home",Action="SearchById"}
);
app.MapControllerRoute(
   name: "SearchByString",
   pattern: "Home/Search/{needle}",
   defaults: new { Controller = "Home", Action = "SearchByString" }
);
```

Anatomie van .Net Core MVC: Eigen routes maken

- **Complexere Routes:**
  - Routes kunnen soms wat complexer opgebouwd worden:
  - Vb: een route **Products/{id}/ShowInfo**
  - Merk op dat de parameter in het midden van de route komt te staan

```
app.MapControllerRoute(
   name: "ShowProductInfo",
   pattern: "Products/{id:int}/ShowInfo",
   defaults: new {Controller="Home",Action="ShowProductInfo" });
```

Anatomie van .Net Core MVC: Eigen routes maken

- **Complexere Routes:**
  - Routes kunnen vaak ook uit meerdere parameters bestaan:
  - Vb: een route **Products/{category}/{price}**
  - Bijv. een search functie met meerdere parameters

```
endpoints.MapControllerRoute(
   name: "ShowProductInfo",
   pattern: "Products/{id:int}/{Category}/{Price}",
   defaults: new {Controller="Home",Action="SearchProduct"});
```

Anatomie van .Net Core MVC: Eigen routes maken

- **Route Attributes**
- Routeparameters kunnen ook voor een controller gedefinieerd worden
- Elke action method moet dan wel een attribute bevatten die **alle mogelijke routes** specifieert voor een bepaalde action method
- De middleware **moet dan wel aangepast worden**

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

- De routes worden door het attribute **Route()** gedefinieerd
- **Vb.** stel de index method van de homecontroller in als root: `[Route("/")]`
- Stel ook de route **home/index** in: `[Route("home/index")]`
- Stel ook de route **/home** in: `[Route("home")]`

Anatomie van .Net Core MVC: Eigen routes maken

- **Route Attributes**
- De route voor de **search** methodes kunnen we als volgt instellen:

```
[Route("home/search/{id:int}")]
public IActionResult SearchByID(int Id)
{
    return Content(Id.ToString());
}
[Route("home/search/{needle}")]
public IActionResult SearchByString(string needle)
{
    return Content(needle);
}
```

Anatomie van .Net Core MVC: Eigen routes maken

- **Conventional routing gebruiken VS Route Attributes gebruiken?**
  - Voor een MVC applicatie geven we de voorkeur aan **Conventional Routing**
  - **Attribute Routing gebruiken we bij voorkeur voor Web API's => *Programming Integration***

Anatomie van .Net Core MVC: Eigen routes maken

- **Oefeningen op leho ☺**