

HoGent

BEDRIJF
EN
ORGANISATIE

Hoofdstuk 6: Een eerste MVC applicatie

Hoofdstuk 6: Een eerste MVC applicatie

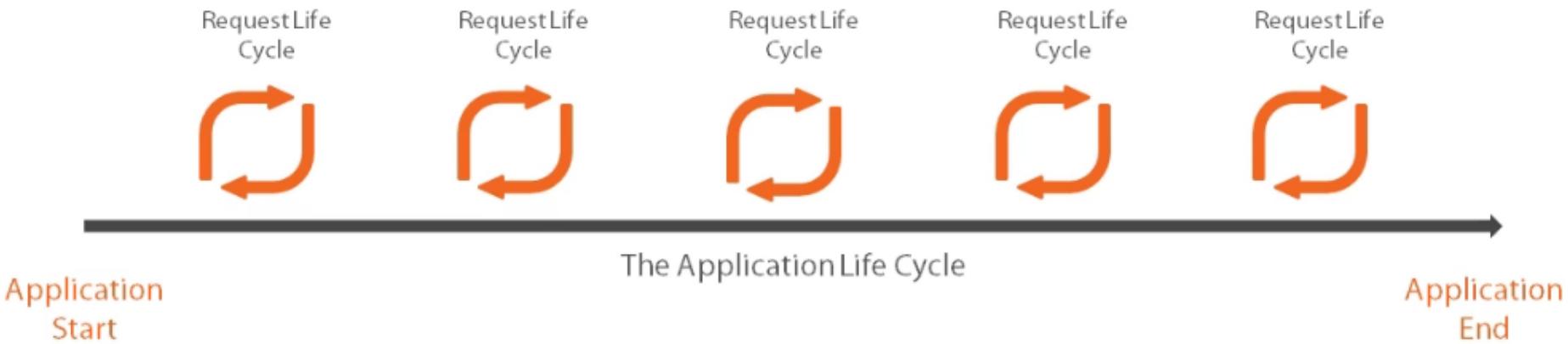
- ▶ Inleiding ASP.NET MVC
- ▶ Hello MVC
 - ✓ Routing
 - ✓ Controller
 - ✓ View
- ▶ SnakeEyes
- ▶ MVC Flow
- ▶ Oefening

Een eerste MVC applicatie

Inleiding

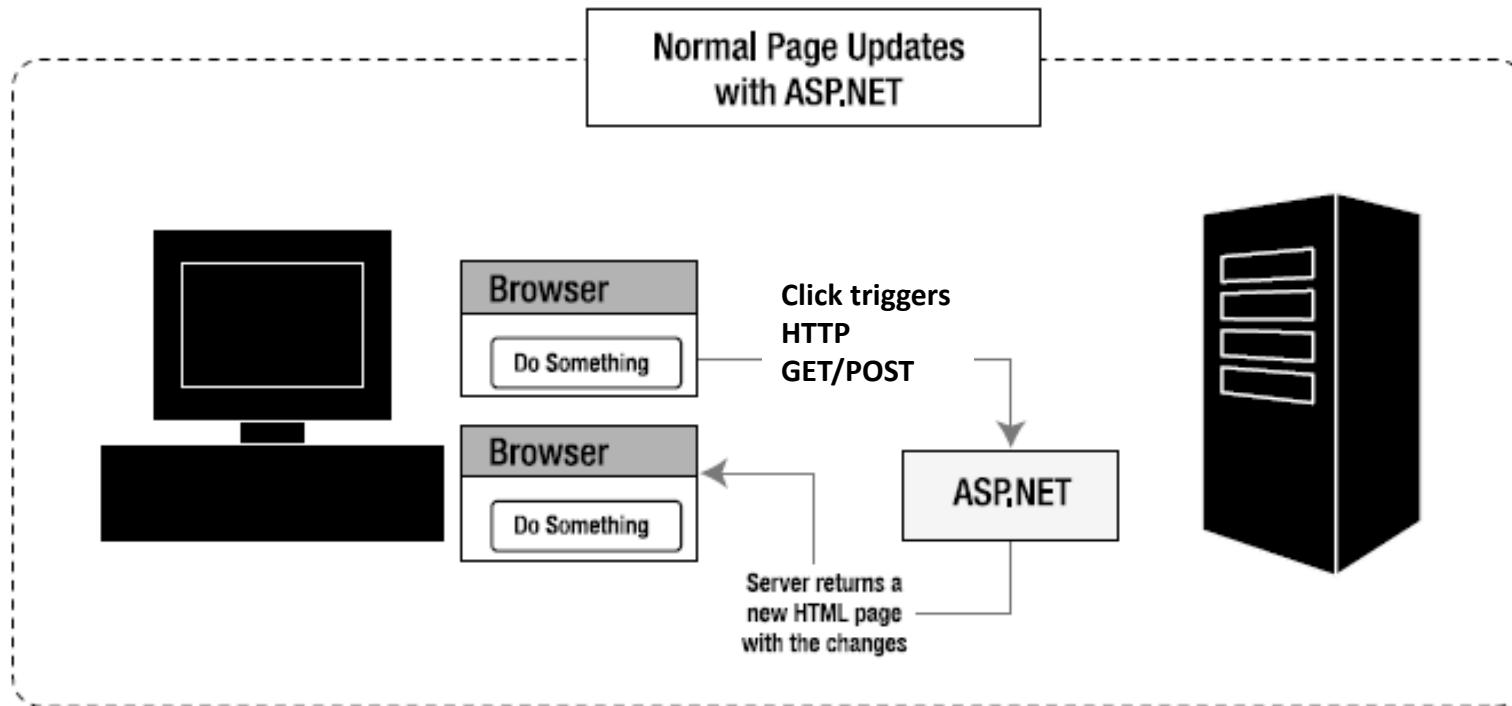
1. Inleiding ASP.NET MVC

▶ The life of an application



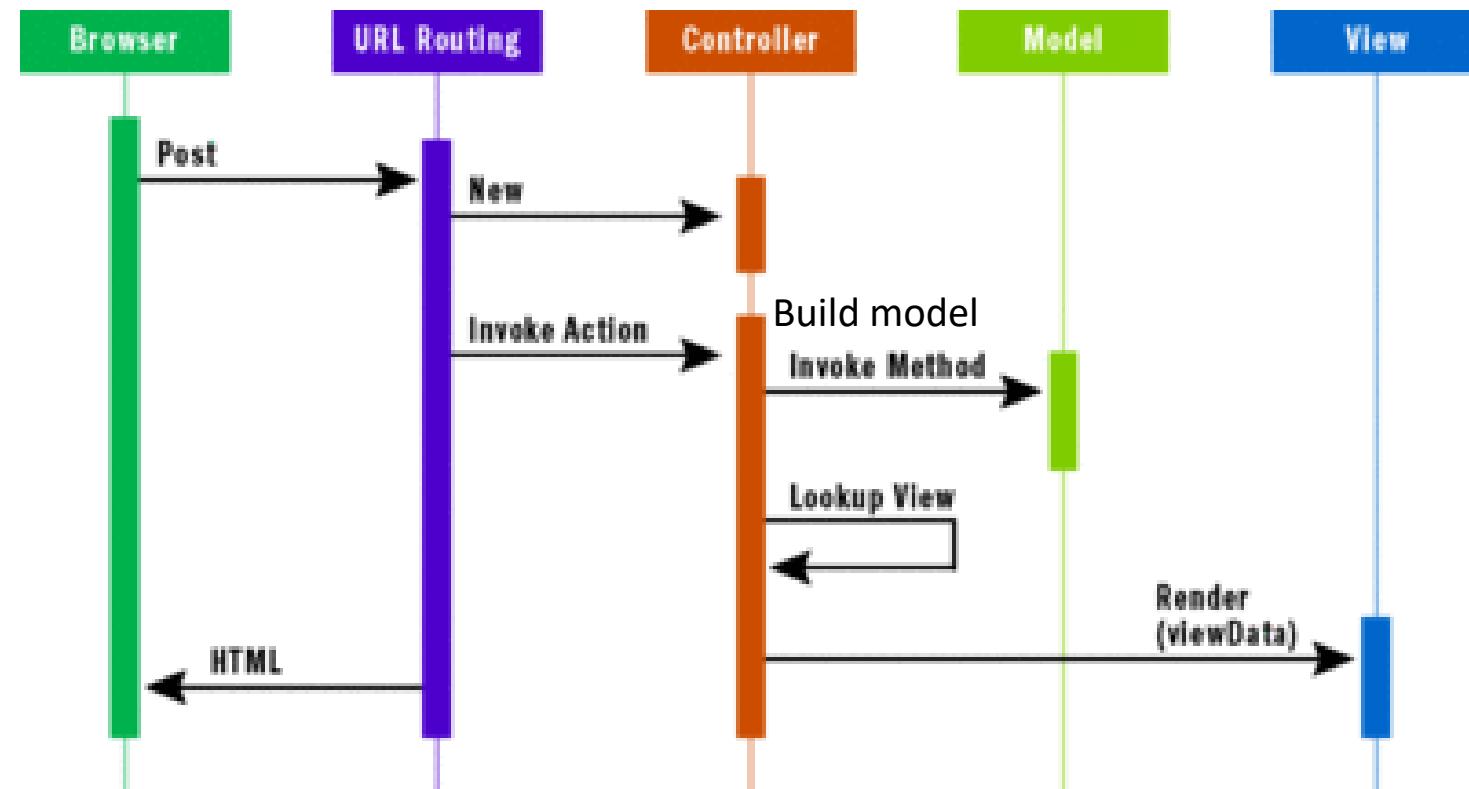
1. Inleiding ASP.NET MVC

- Request life cycle



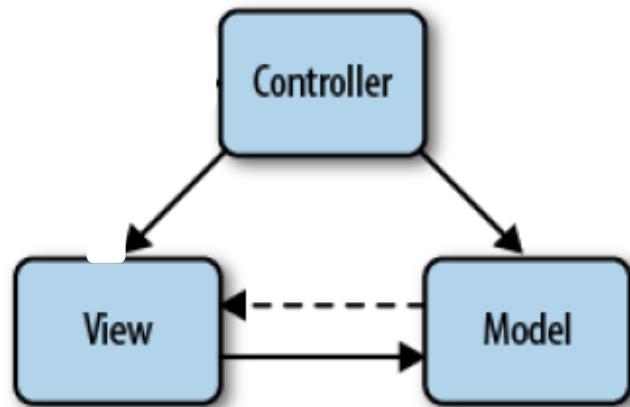
1. Inleiding ASP.NET MVC

- ▶ De **web** variant van MVC (Model2) : request life cycle



1. Inleiding ASP.NET MVC

<https://docs.asp.net/en/latest/mvc/overview.html>



Controller

- Intercepts user input
- Coordinates the view and model
- Handles communication between the model and data layer

Model

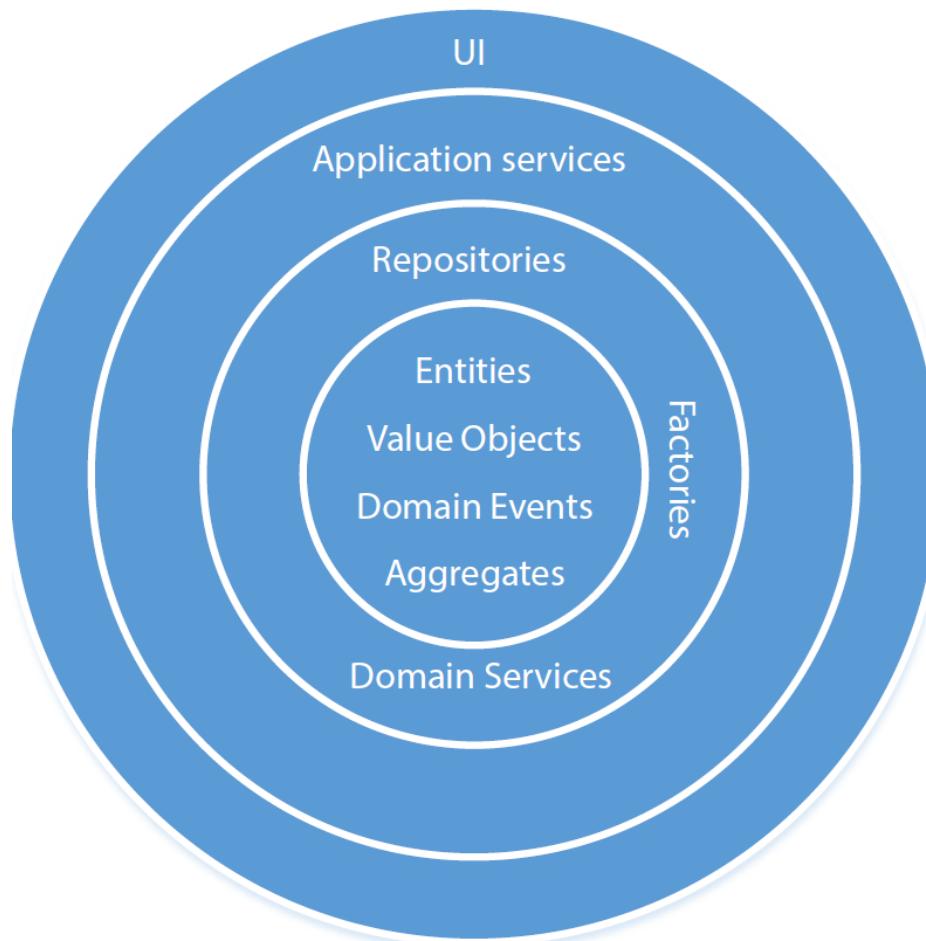
- Data attributes (properties)
- Business logic, behavior, and validation

View

- Renders the UI (HTML, CSS, PDF)
- Binds to the model

1. Inleiding ASP.NET MVC

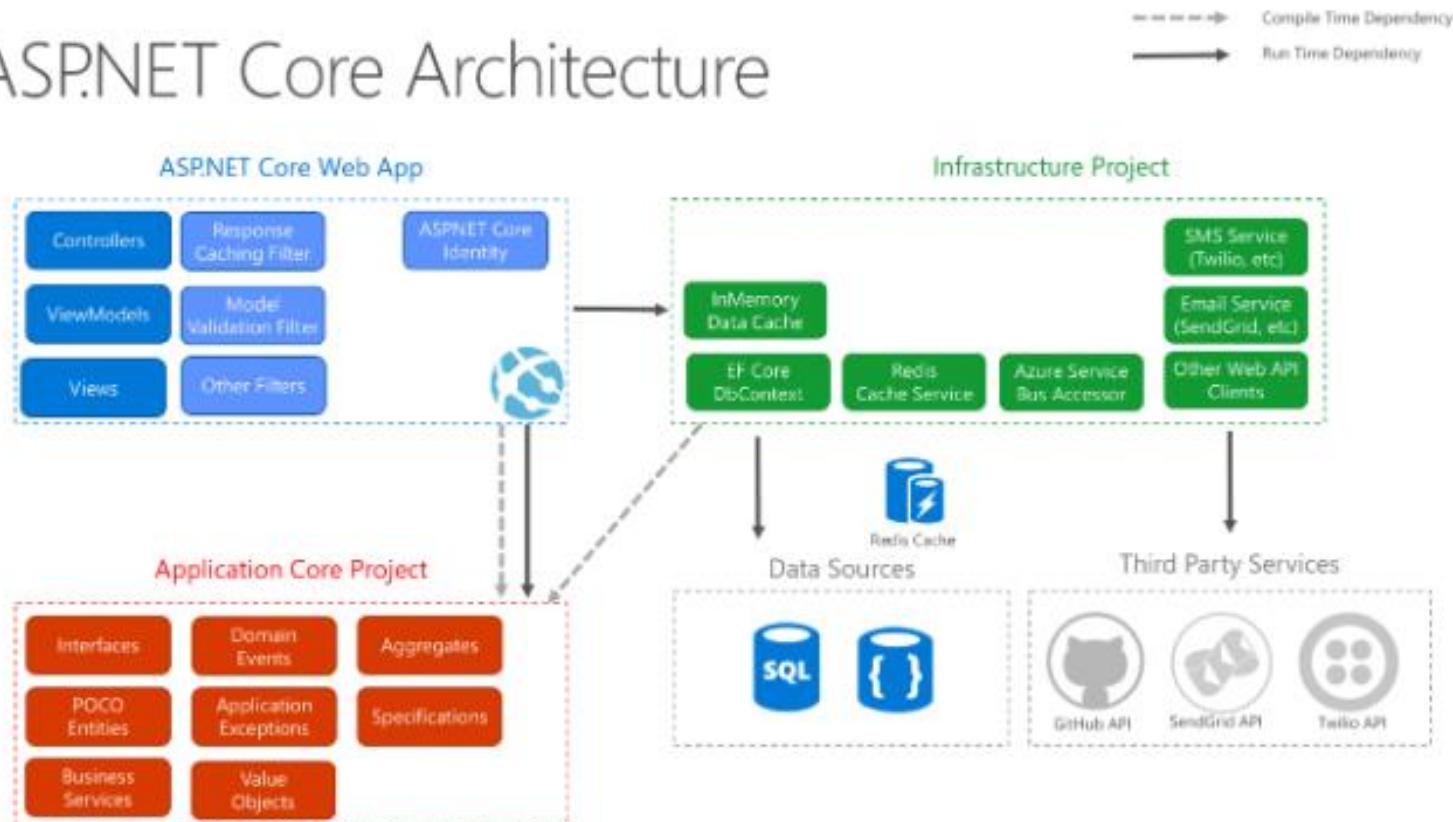
► Clean architecture/Onion Architecture



1. Inleiding ASP.NET MVC

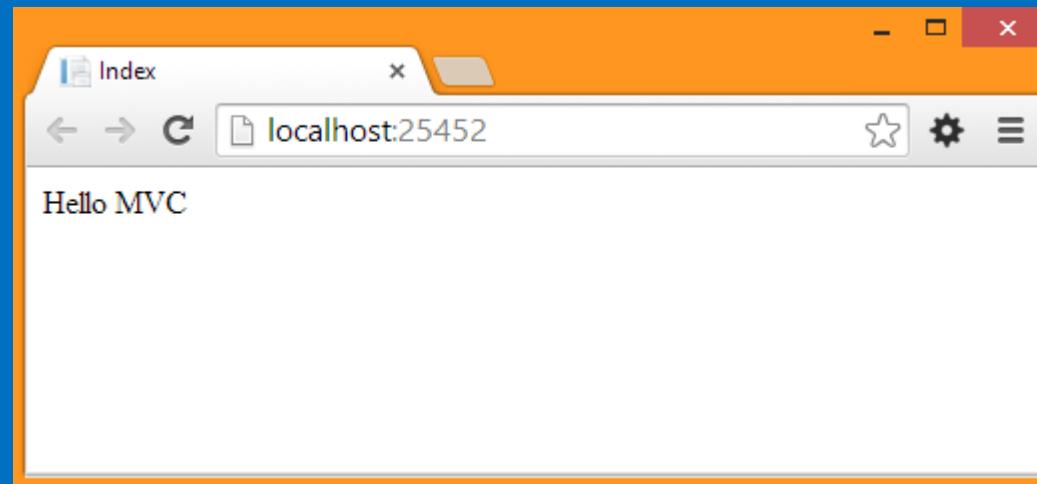
- ASP.NET Core architecture diagram following Clean Architecture.

ASP.NET Core Architecture



Een eerste MVC applicatie

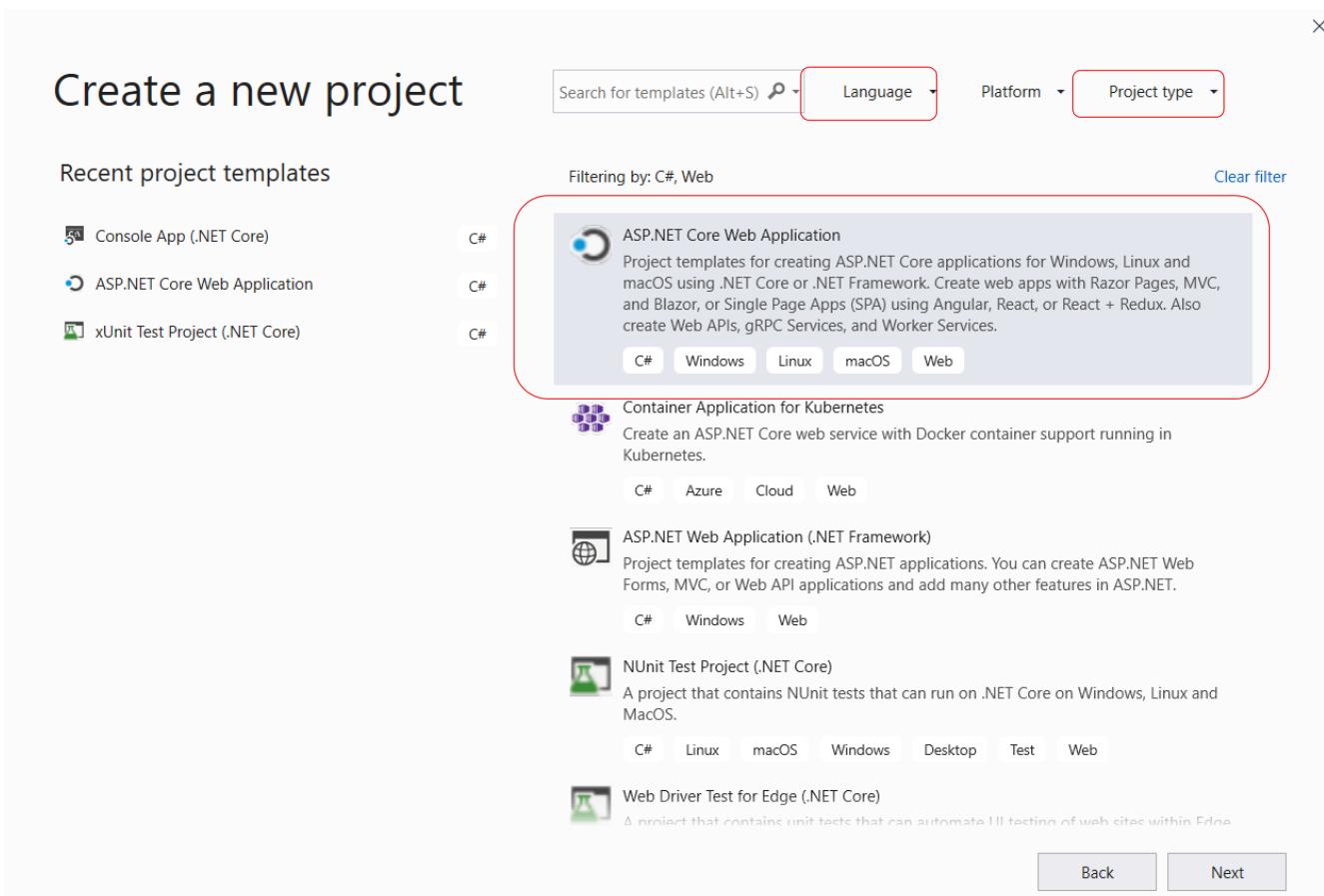
Hello MVC



2. Hello MVC – Aanmaken project

▶ Aanmaken project

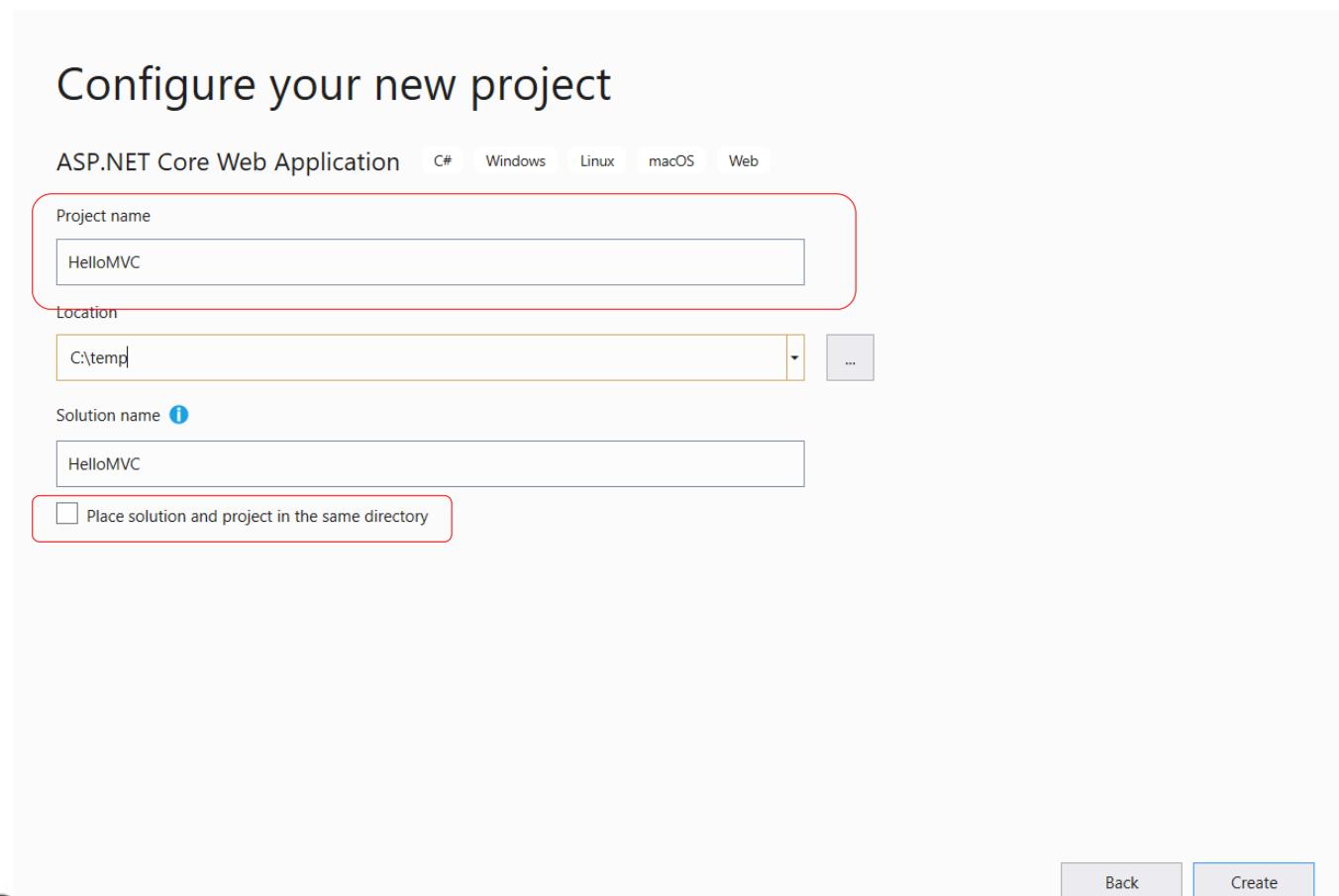
- Maak een nieuwe MVC Core applicatie genaamd HelloMVC
 - Create a new project > Kies Visual C# en web als Project type > ASP.NET Core Web Application. Klik Next



2. Hello MVC – Aanmaken project

▶ Aanmaken project

- Geef de naam van de applicatie in en kies de locatie. Vink place project and solution in the same directory uit



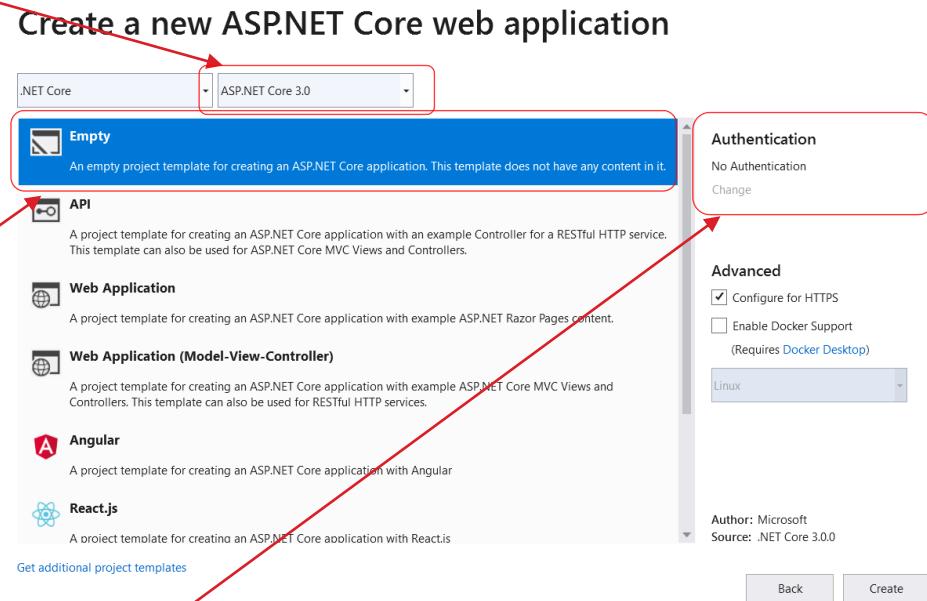
2. Hello MVC – Aanmaken project

▶ Stel features in van Web app

- Kies voor **ASP.NET Core 3.0**
- Select a template : **Empty**
 - Empty : leeg project.
 - API : bouwen van REST Services
 - Web application (Razor Pages)
 - Web application (Model-View-Controller)
 - Angular: Web IV
 - React.js: Web IV
 - React.js and Redux:Web IV

Kies Empty

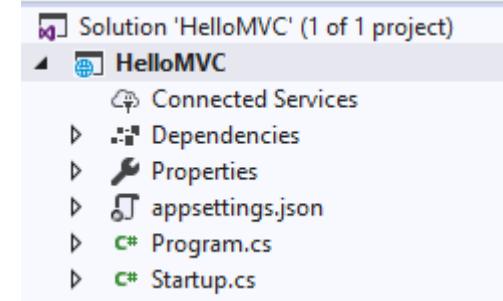
- Authenticatie : **No authentication**



2. Hello MVC – Folder structuur

▶ In Solution Explorer

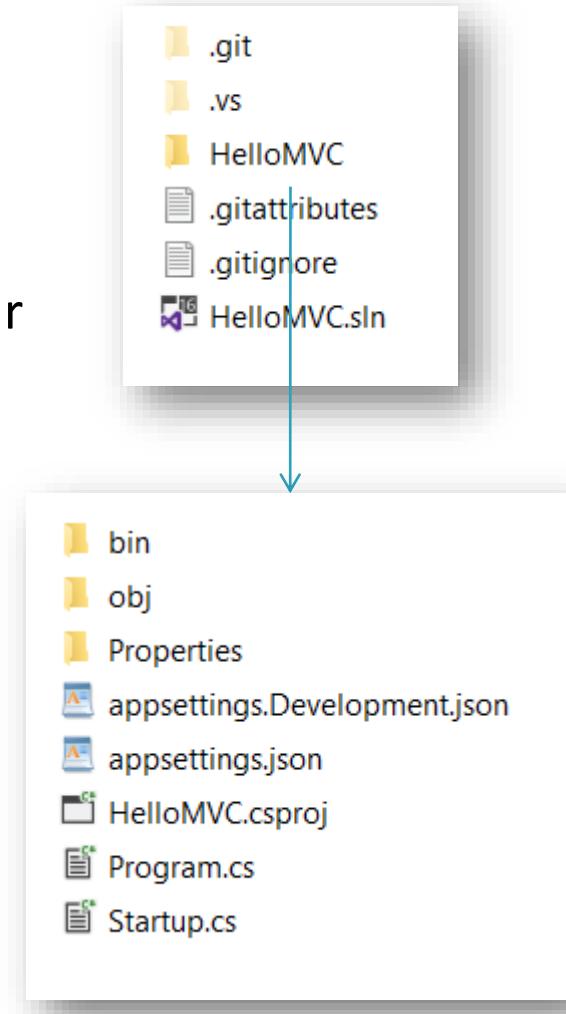
- Connected Services: zoals Azure Blob, Azure IoT Hub,...
- Dependencies : gebruikte assemblies
- Properties : eigenschappen van het project
- Program.cs: ASP.NET Core projecten zijn Console applicaties die een Web server execution environment opstarten.
- Startup.cs : Het hart van de applicatie.
Een configuratie bestand, dat wordt uitgevoerd bij de start up van de webapplicatie – stelt het startpunt en de omgeving in voor de ASP.NET Core applicatie. Het creëert services en injecteert dependencies zo dat de applicatie er gebruik kan van maken.
- **Rechtsklik solution in solution Explorer > add**



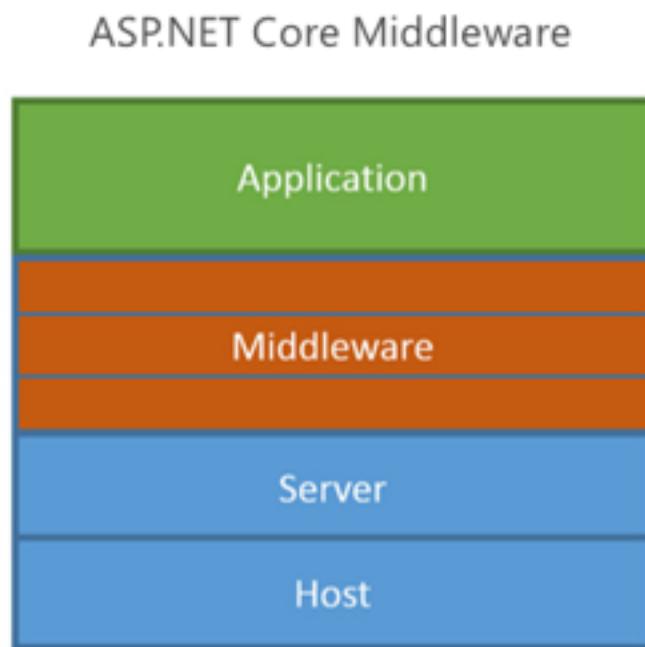
2. Hello MVC – Folder structuur

▶ in Windows Verkenner

- Rechtsklik de solution > Open folder in file explorer
 - .sln : solution file. xml file met verwijzing naar projecten,... die solution bevat
(<https://msdn.microsoft.com/en-us/library/bb165951.aspx>)
 - HelloMVC (= root applicatie)
 - File system bepaalt wat in project zit. Je kan bestanden toevoegen aan de folder en deze verschijnen automatisch in solution explorer.
 - VS zal een nieuw bestand ook compileren.



2. Hello MVC - Architecture



2. Hello MVC – Hosting

▶ Program.cs

- Bevat een **Main** methode : ASP.NET Core applications zijn zelfstandige Console applications die verantwoordelijk zijn voor de hosting en configuratie (Startup).
 - CreateWebHostBuilder methode zal een default hosting environment creëeren voor je webapplicatie. Een host wordt opgezet, die een server configureert en ook de applicatie pipeline
 - CreateDefaultBuilder is verantwoordelijk voor de creatie en configuratie van de host. Zie volgende slide.
 - UseStartUp : Instantieert de Startup klasse. De runtime zal door de ConfigureServices gaan en vervolgens Configure uitvoeren. Zie verder Startup: request pipeline builden.
 - CreateHostBuilder(args).Build().Run() : build en start je asp.net core app. Vanaf nu is de applicatie geen console applicatie meer maar een asp.net core web app.

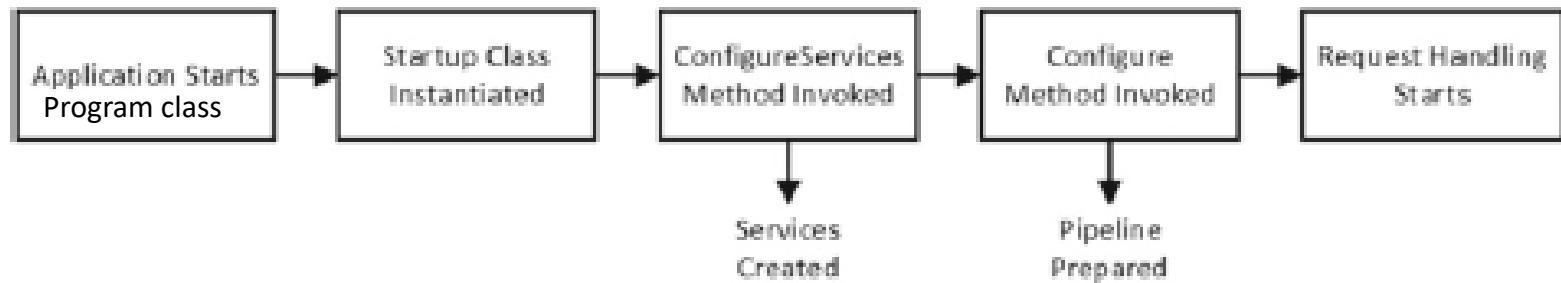
```
public class Program
{
    public static void Main(string[] args)
    {
        CreateHostBuilder(args).Build().Run();
    }

    public static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
            .ConfigureWebHostDefaults(webBuilder =>
            {
                webBuilder.UseStartup<Startup>();
            });
}
```

2. Hello MVC – Application startup

▶ Startup.cs

- Configuratie en startup code voor asp.net core applicatie
- Bevat een StartUp klasse.



2. Hello MVC – Application startup

▶ Startup.cs

- ConfigureServices:
IoC container wordt ingesteld.
- Configure: bouwt de request pipeline.

```
public class Startup
{
    // This method gets called by the runtime. Use this method to add services to the container.
    // For more information on how to configure your application, visit https://go.microsoft.com/fwlink/?LinkID=2028313
    public void ConfigureServices(IServiceCollection services)
    {
    }

    // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }

        app.UseRouting();

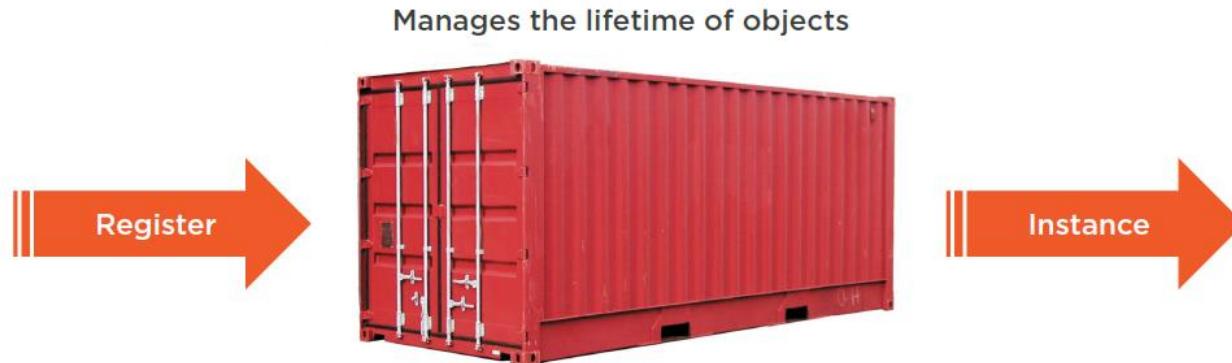
        app.UseEndpoints(endpoints =>
        {
            endpoints.MapGet("/", async context =>
            {
                await context.Response.WriteAsync("Hello World!");
            });
        });
    }
}
```

2. Hello MVC – Application startup

▶ Startup.cs

- ConfigureServices methode
 - Opzetten van services en IoC Container (dependency injection)
 - *Service = elk object dat functionaliteit verschafft aan andere delen van je applicatie.* Vb : services.AddControllersWithViews();
 - IoC Container

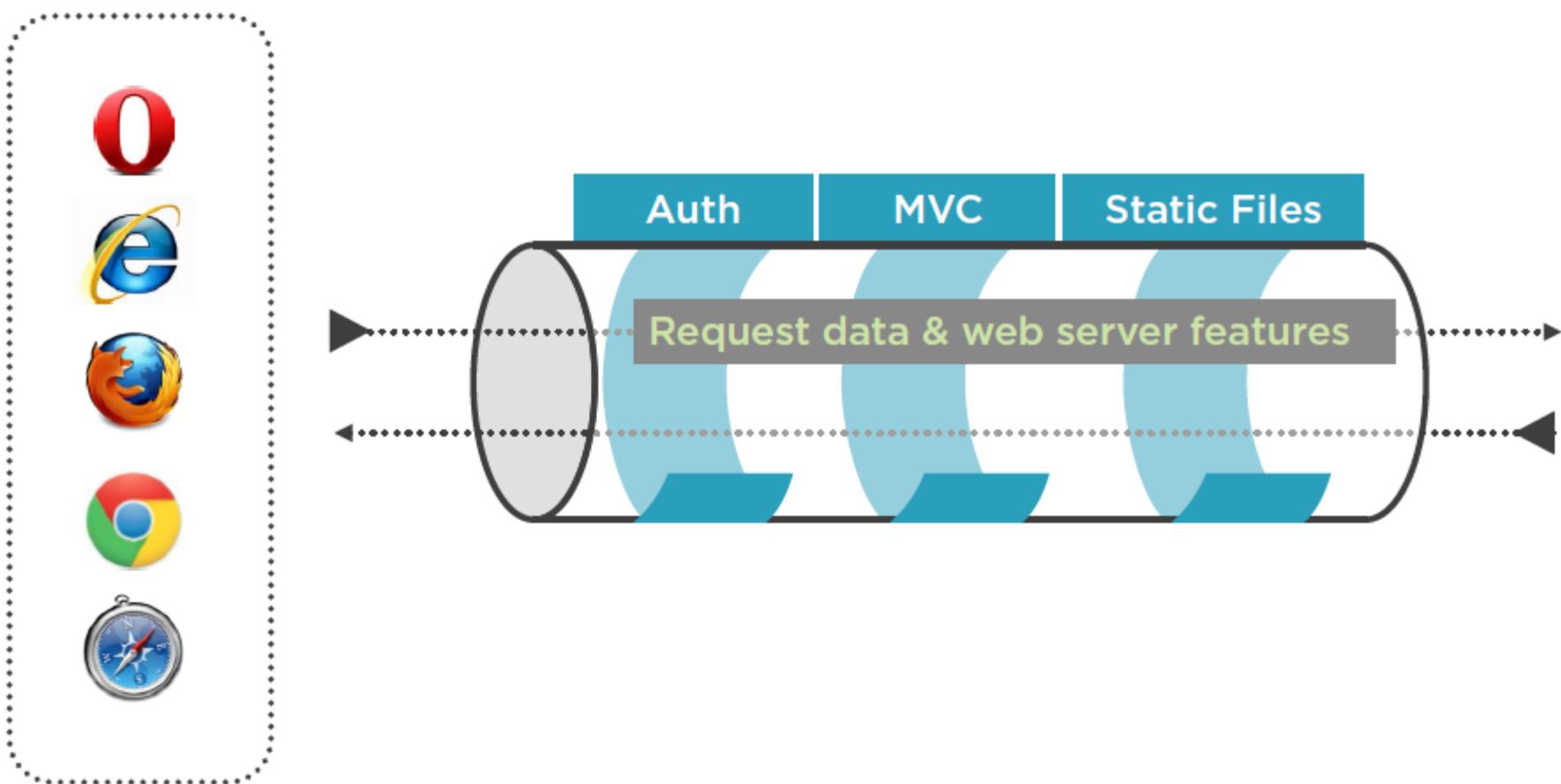
Dependency Injection



2. Hello MVC – Application Startup

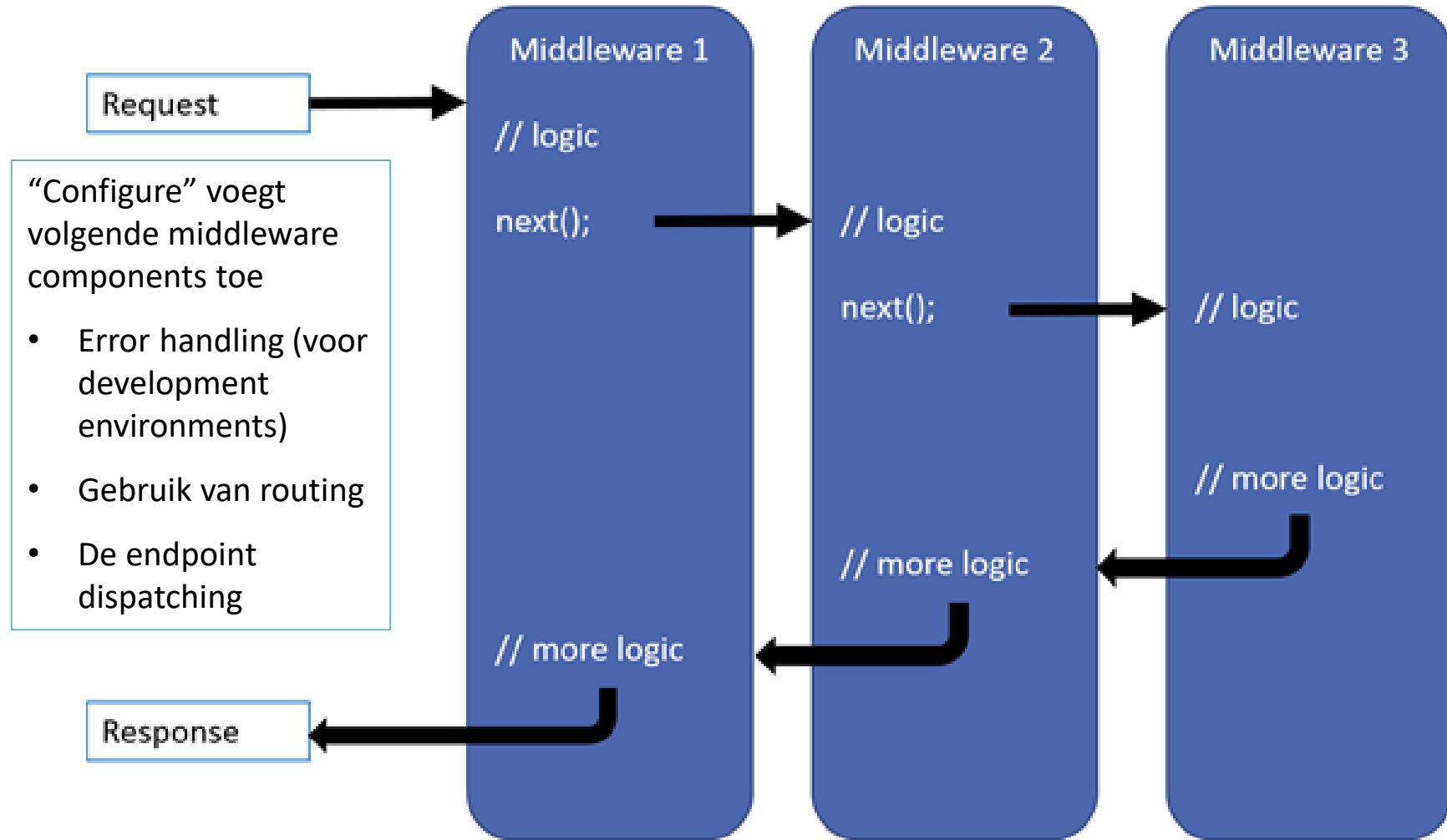
▶ Startup.cs

- Configure methode :
 - bouwt de http pipeline



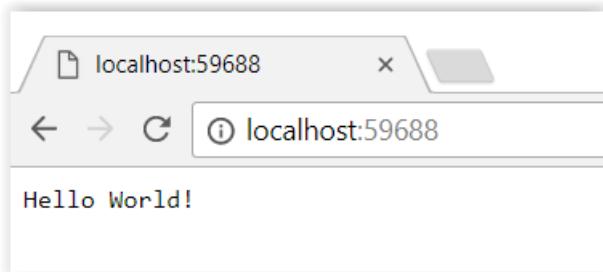
2. Hello MVC –Application Startup

▶ Middleware



2. Hello MVC – Application Startup

- ▶ Run nu de applicatie.
- ▶ De pipeline wordt opgebouwd en afgesloten door app.Run.
Alle code na app.Run wordt niet verder uitgevoerd.



```
public class Startup
{
    // This method gets called by the runtime. Use this method to add services to the container.
    // For more information on how to configure your application, visit https://go.microsoft.com/fwlink/
    public void ConfigureServices(IServiceCollection services)
    {
    }

    // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }

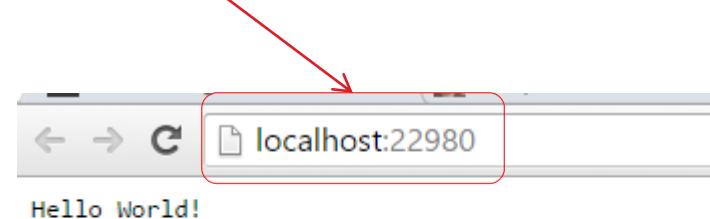
        app.UseRouting();

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapGet("/", async context =>
            {
                await context.Response.WriteAsync("Hello World!");
            });
        });
    }
}
```

env.IsDevelopment() checkt de environment. Open properties van het project, ga naar Debug > Environment variables > ASPNETCORE_ENVIRONMENT (kan ook production of staging zijn)

2. Hello MVC – Run applicatie

- ▶ Run de applicatie : F5,  of Debug > Start Debugging
- ▶ In de dropdown  kan je de browser selecteren
- ▶ Visual Studio
 - Compileert de applicatie
 - Start IIS Express, zie notificatie  onderaan in taakbalk of Task Manager). Kiest random een vrije poort.
 - Opent de browser
- ▶ Klik  : om te stoppen



2. Hello MVC – Application Startup

- ▶ Throw een Exception in het begin van app.Run.

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapGet("/", async context =>
    {
        throw new NotImplementedException("error");
        await context.Response.WriteAsync("Hello World!");
    });
});
```

- ▶ Run de applicatie opnieuw.
 - De pipeline blijft hetzelfde maar nu gaat de ExceptionHandling Middleware de runtimefout opvangen en een developer foutpagina retourneren. (In Production wordt dit een errorpage).

An unhandled exception occurred while processing the request.

NotImplementedException: error
HelloMVC.Startup+<>c+<<Configure>b__1_1>d.MoveNext() in **Startup.cs**, line 35

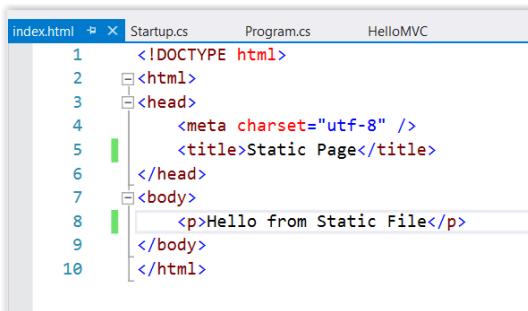
Stack Query Cookies Headers Routing

NotImplementedException: error

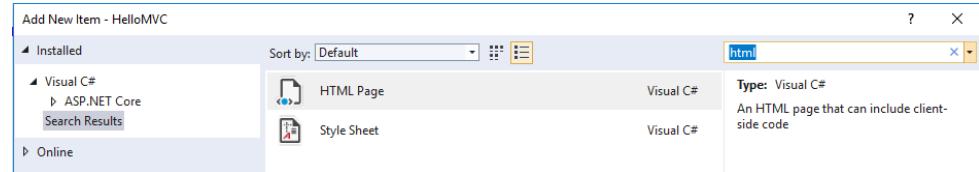
HelloMVC.Startup+<>c+<<Configure>b__1_1>d.MoveNext() in **Startup.cs**
35. throw new NotImplementedException("error");
Microsoft.AspNetCore.Routing.EndpointMiddleware.<Invoke>g__AwaitRequestTask|6_0(Endpoint endpoint, Task requestTask, ILogger logger)
Microsoft.AspNetCore.Diagnostics.DeveloperExceptionPageMiddleware.Invoke(HttpContext context)

2. Hello MVC – Application Startup

- ▶ We gaan nu trachten een statische pagina weer te geven (index.html).
- ▶ Add> new folder > wwwroot
- ▶ Add>New Item>html



```
index.html  X Startup.cs  Program.cs  HelloMVC
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8" />
5      <title>Static Page</title>
6  </head>
7  <body>
8      <p>Hello from Static File</p>
9  </body>
10 </html>
```



- ▶ Run de applicatie: Geef in de browser de index.html toe aan de url: We krijgen een 404 opgemaakt door de browser



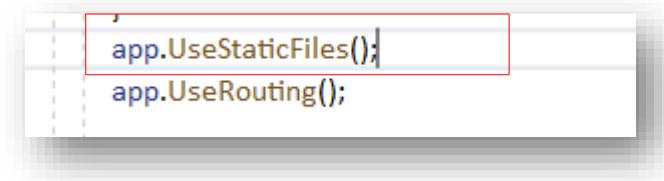
This localhost page can't be found

No webpage was found for the web address: <https://localhost:44337/index.html>

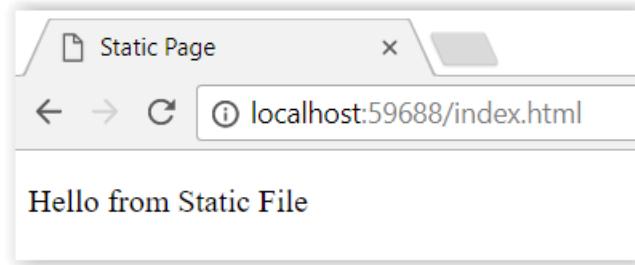
HTTP ERROR 404

2. Hello MVC – Application Startup

- ▶ We moeten aan de pipeline toevoegen dat er static files kunnen gebruikt worden. (voor app.useRouting)



- ▶ Run de applicatie en voeg index.html toe aan de url.



- ▶ Voeg voor app.UseStaticFiles() app.UseDefaultFiles() toe en dan hoef je index.html niet meer in te geven.

2. Hello MVC – Static Files

▶ Static files – wwwroot folder

- Statische bestanden kunnen opgeslagen worden in elke map onder de wwwroot folder en worden toegankelijk met een relatief pad naar die root. Bijvoorbeeld, wanneer u een standaard web project maakt met Visual Studio, zijn er meerdere mappen gemaakt in de wwwroot-map - css, afbeeldingen en js.

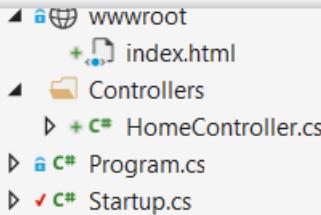
De URI heeft toegang tot een afbeelding in de afbeeldingenmap: `http://<app>/images/<imageFileName>`

2. Hello MVC – MVC gebruiken

- ▶ Verwijder de index.html
- ▶ Om MVC te gebruiken dienen we het mappen naar Controllers in UseEndpoints te configureren als volgt

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

- ▶ Vanaf nu zal de request van de client op zoek gaan naar een HomeController met een methode Index in de map Controllers. Zie verder.

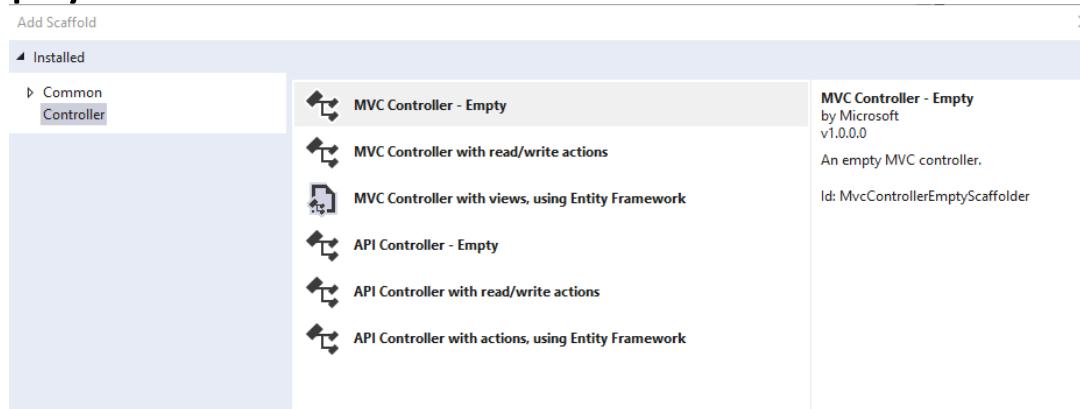


```
public class HomeController
{
    public string Index()
    {
        return "Hello from index in homecontroller";
    }
}
```

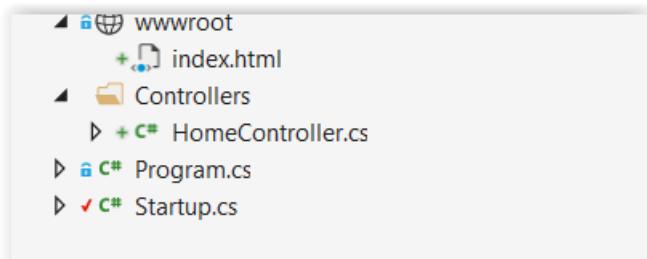
2. Hello MVC – MVC gebruiken

▶ Maak de HomeController aan

- Voeg een nieuwe map Controllers toe
- Rechtsklik Controllers > Add > Controller. Kies MVC Controller – Empty. Noem de Controller HomeController



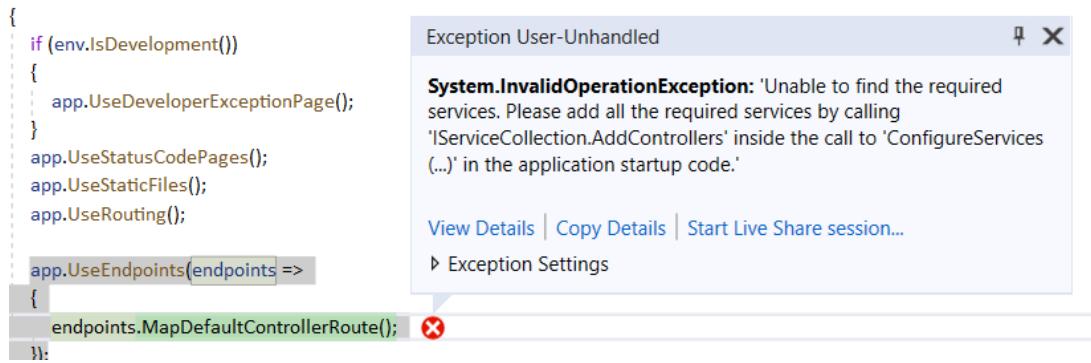
- Pas de code aan



```
public class HomeController
{
    public string Index()
    {
        return "Hello from index in homecontroller";
    }
}
```

2. Hello MVC – MVC gebruiken

- ▶ We krijgen echter een Error.

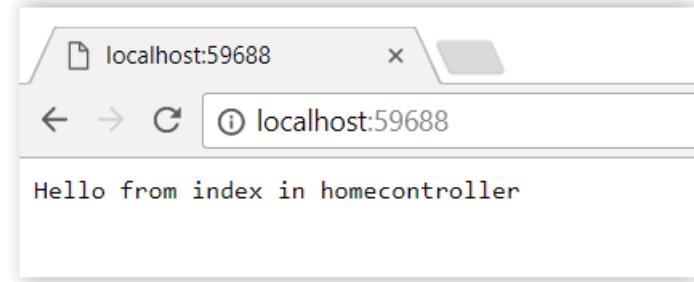


- ▶ Voeg de service toe aan de Dependency Injection Container.

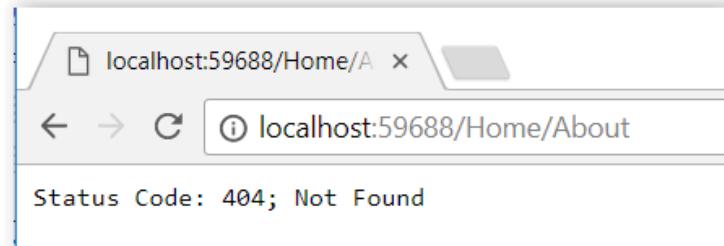
```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllersWithViews();
}
```

2. Hello MVC – MVC gebruiken

- ▶ Nu zien we de pagina.



- ▶ Surf nu naar /Home/About -> In chrome krijgen we een 404, Page Not Found. (In explorer een blanco pagina). By default, voorziet asp.net core geen status code page. De app retourneert een 404 en een lege response body. (Bekijk dit in de Chrome developer tools)
- ▶ Om toch in status code pages te voorzien : opvangen in de pipeline : app.UseStatusCodePages()



2. Hello MVC – Conventions

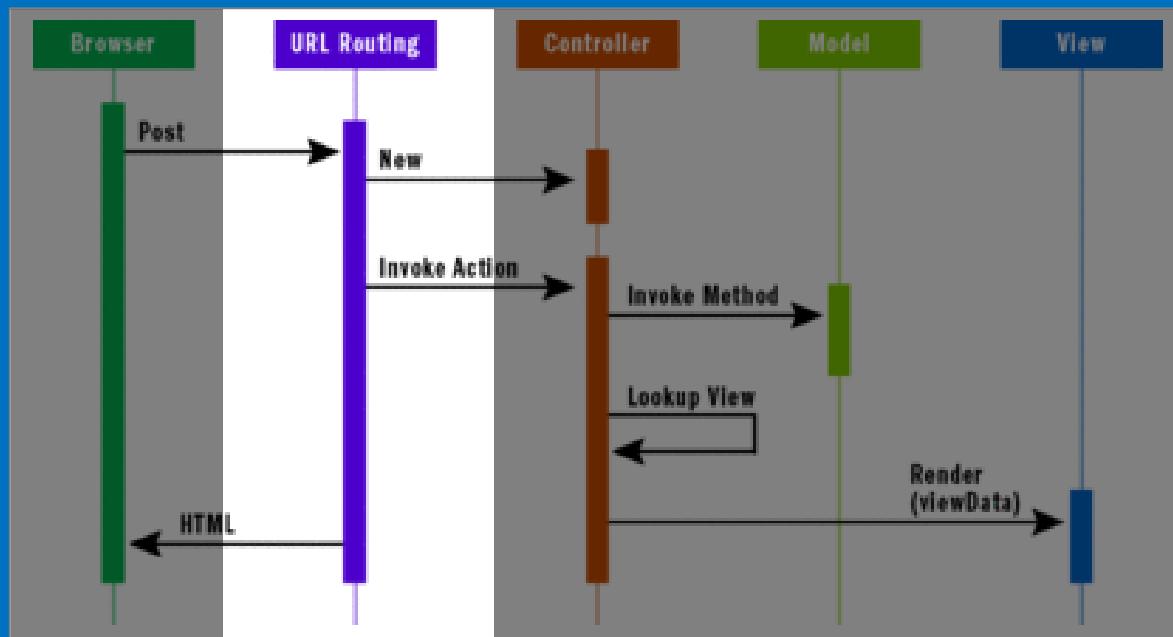
► Convention over configuration

We know, by now, how to build a web application. Let's roll that experience into the framework so we don't have to configure absolutely everything again.

- ASP.NET MVC gebruikt waar mogelijk Convention over configuration
 - 3 folders voor de elementen van MVC patroon :
 - **Controllers** folder : bevat de controller klassen. Alle Controller klassen hebben *Controller* suffix
 - **Views** folder : 1 subfolder per controller. De naam van de View is de naam van de methode in de Controller
 - **Models** folder : bevat de domeinklassen, voegen we later toe

Een eerste MVC applicatie

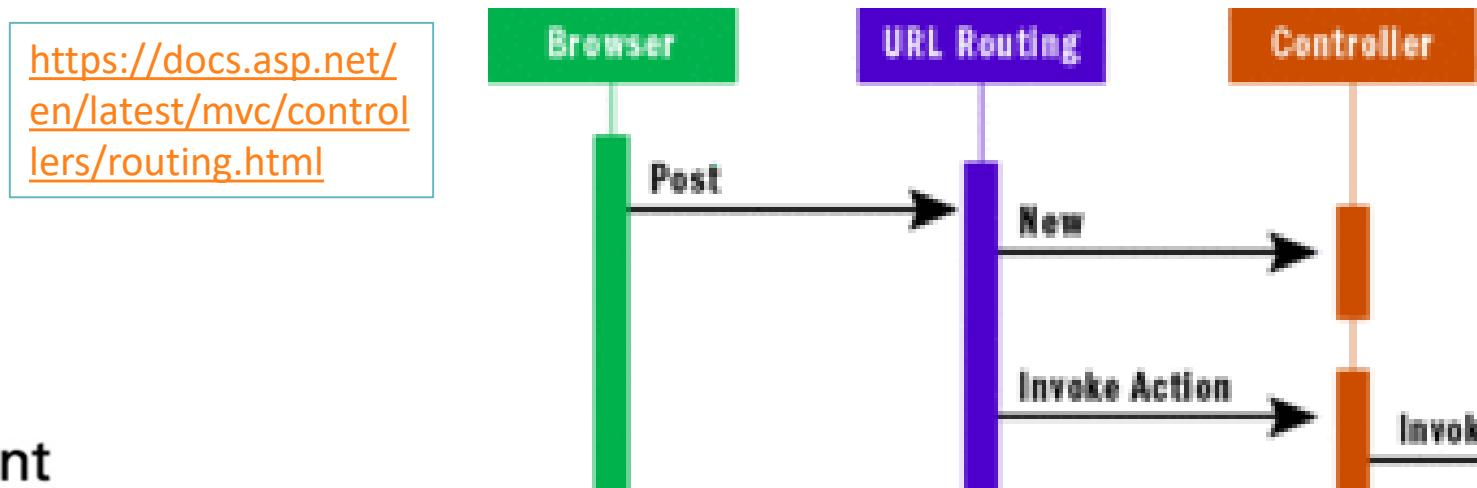
Hello MVC - URL Routing



2. Hello MVC – URL Routing

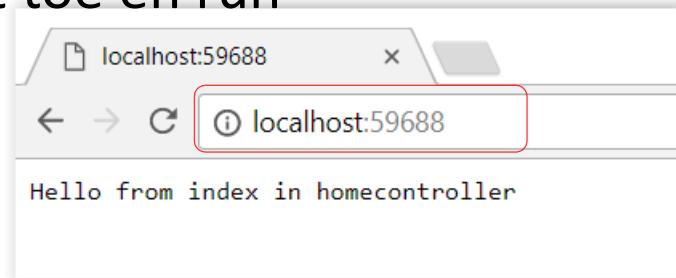
▶ URL Routing

- ASP.NET Core MVC gebruikt
 - de **Endpoint Routing Middleware** die binnenkomende HTTP requests analyseert en maapt op een endpoint
 - de **Endpoint (dispatch) Middleware**, de laatste MW in de pipeline, voert de endpoint uit. Het roept de bijhorende Action Method in de Controller aan. Ook bijkomende informatie (parameters) wordt geanalyseerd door de routing en doorgegeven aan de action methode (model binding). Routes worden gedefinieerd in startup code of attributes.

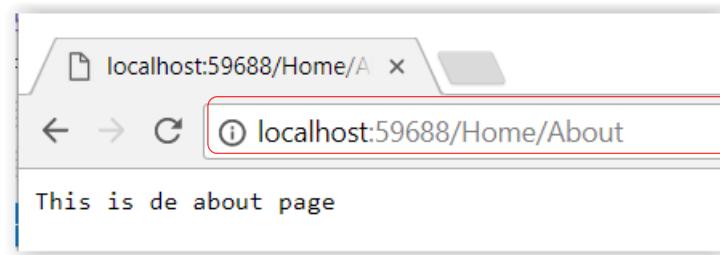


2. Hello MVC – URL Routing

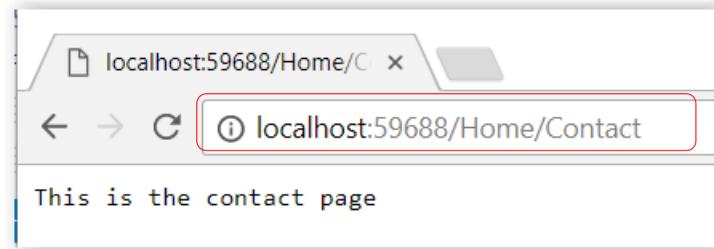
- Pas de HomeController verder aan: Voeg zelf een methode About en Contact toe en run



- Ga naar Home



- Ga naar About



- Ga naar Contact

- URL Routing : analyseert de url.

2. Hello MVC – URL Routing

► Controller

```
namespace HelloMVC.Controllers
{
    public class HomeController : Controller
    {
        public string Index()
        {
            return "Hello from index in HomeController";
        }

        public string About()
        {
            return "This is the about page";
        }

        public string Contact()
        {
            return "This is the contact page";
        }
    }
}
```

The code defines a `HomeController` with three methods: `Index()`, `About()`, and `Contact()`. Each method returns a string representing the content of the page. To the right of the code, there are three browser icons with URLs: `localhost:1988` (for `Index()`), `localhost:1988/Home/Index` (for `About()`), and `localhost:1988/Home/Contact` (for `Contact()`). Blue arrows point from the text of each method's return value to its corresponding URL.

Controller klasse : bevat methodes die uitgevoerd worden bij request van de gebruiker.
O.b.v. de url bepaalt de **Routing Middleware**

- welke Controller geïnstantieerd zal worden
- en welke methode zal worden uitgevoerd in die Controller.

Routing : wordt uitgevoerd bij elke request. De url wordt geanalyseerd :

Url : .../Home/Index

HomeController

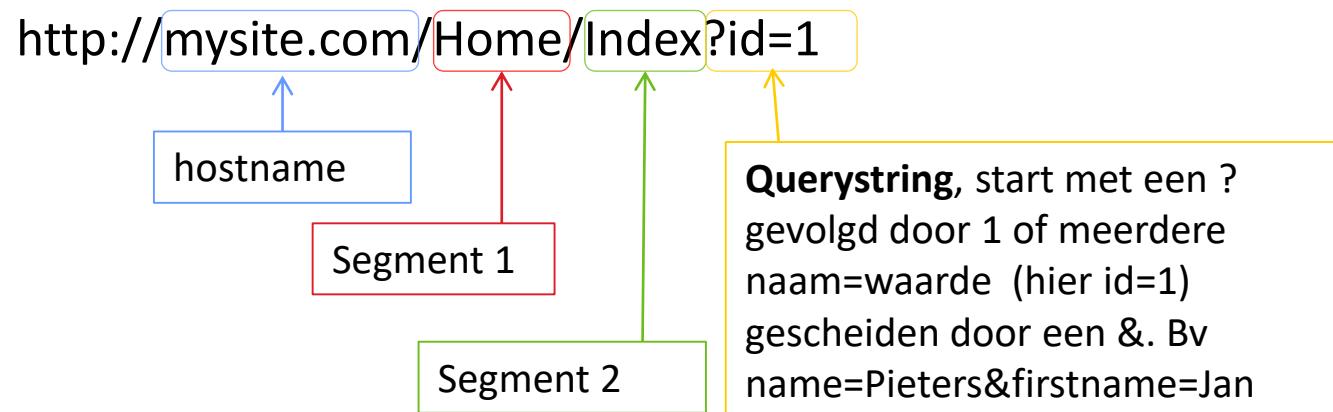
Index methode

Methode `About` in de `HomeController` wordt uitgevoerd als men surft naar de Url `.../Home/About`

Methode `Contact` in `HomeController` wordt uitgevoerd als men surft naar Url : `.../Home/Contact`

2. Hello MVC : Routing

- ▶ MVC gebruikt een **resource centric routing**.
 - Een URL stelt een **resource** op het web voor (~REST)
 - Een resource is in ASP.NET MVC een **stukje code** die de request afhandelt
 - MVC Routing maapt een request (URL) naar **een actie methode in een Controller**, eventueel met parameters.
 - De URL (exclusief hostname en querystring) wordt hiervoor opgesplitst in segmenten



2. Hello MVC : Routing

▶ URL patronen (Convention based routing)

- De URL matching werkt met ***URL patronen***
- In een patroon wordt elk segment van een url voorgesteld door een **segment variabele**. Een segment variabele is een naam tussen { }.

pattern: “{controller}/{action}”

- {controller} => de naam van de controller klasse die MVC zal instantiëren. MVC voegt het woord Controller toe aan de naam. Bvb Home. De bijhorende controller klasse HomeController.cs.
- {action} => naam van de *publieke* methode binnen de controller die MVC zal aanroepen
- Beide zijn voorgedefinieerde segment variabelen

2. Hello MVC : Routing

▶ URL patronen

pattern: “{controller}/{action}”

- Deze route mapt enkel met url's bestaande uit 2 segmenten
- De querystring kan je niet mappen. Dit wordt een parameter van de betreffende actie methode

Request URL	Segment variables
http://mysite.be/Home/Index	controller = Home, action = Index
http://mysite.be/Cart/Add	controller = Cart, action = Add
http://mysite.be/Cart/Add?id=10	controller = Cart, action = Add, parameter id
http://mysite.be/Home	geen match - te weinig segmenten
http://mysite.be/Index/Home/Soccer	geen match - te veel segmenten

2. Hello MVC : Routing

▶ URL patronen

- Defaultwaarden voor ontbrekende segmenten van URL mogelijk

pattern: "{controller=Home}/{action=Index}";

- action : indien url geen action bevat, roep methode Index aan
- controller : als url geen controller bevat instantieer HomeController
- Dit mapt naar de url's bestaande uit 0 tot 2 segmenten

Request URL	Segment variables
http://mysite.be/Home/Index	controller = Home, action = Index
http://mysite.be/Cart/Add	
http://mysite.be/Cart/Add?id=10	
http://mysite.be/Home	
http://mysite.be/Cart	
http://mysite.be	
http://mysite.be/Index/Home/Soccer	

2. Hello MVC : Routing

- ▶ URL patronen en custom segmenten
 - Je kan ook zelf segmenten definiëren

pattern: “**{controller}**/**{action}**/**{id}**”

- Dit patroon matcht met url's bestaande uit 3 segmenten.
 - **{controller}** => de naam van de controller klasse die MVC zal instantiëren. MVC voegt het woord Controller toe aan de naam. Bvb Home. De bijhorende controller klasse HomeController.cs.
 - **{action}** => naam van de *publieke* methode binnen de controller die MVC zal aanroepen
 - **{id}** => een parameter met naam id. De actionmethode moet dan een parameter id bevatten. Bvb public IActionResult Index(int id)

2. Hello MVC : Routing

▶ URL patronen en custom segmenten

- Voor een custom segment kan een defaultwaarde worden opgeven of kan opgeven worden dat het optioneel is(?)

pattern: "{controller=Home}/{action=Index}/{id?}"

URL	CONTROLLER CLASS	ACTION METHOD	PARAMETERS PASSED
/Dinners/Details/2	DinnersController	Details(id)	id=2
/Dinners/Edit/5	DinnersController	Edit(id)	id=5
/Dinners/Create	DinnersController	Create()	N/A
/Dinners	DinnersController	Index()	N/A
/Home	HomeController	Index()	N/A
/	HomeController	Index()	N/A

2. Hello MVC : Routing

- ▶ URL patronen en custom segmenten
 - Voor een custom segment kan ook een route constraint worden opgegeven

```
pattern: "{controller=Home}/{action=Index}/{id:int}"
```

- Meer info op
<https://docs.asp.net/en/latest/fundamentals/routing.html#route-constraint-reference>

2. Hello MVC : Routing

▶ Opzetten van de **Routing middleware**

- De klasse StartUp, methode Configure bevat de instellingen
 - UseRouting() : voegt de Endpoint Routing Middleware toe aan de pipeline. Deze middleware maapt de url met een endpoint en voegt een Endpoint object toe aan de HttpContext.
 - UseEndpoints : voegt Endpoint Dispatch Middleware toe, steeds als laatste in de pipeline. Voert het endpoint uit.
 - Moet worden voorafgegaan door UseRouting()
 - MapControllerRoute voegt controller based en Web API routing toe en definieert een URL patroon van een route (endpoint)

```
app.UseRouting();  
  
app.UseEndpoints(endpoints =>  
{  
    endpoints.MapControllerRoute(  
        "default",  
        "{controller=Home}/{action=Index}/{id?}",);  
});
```

Een naam die je aan een bepaalde route geeft. Willekeurig te kiezen

Het URL patroon en defaultwaarden voor ontbrekende onderdelen van het URL patroon

2. Hello MVC : Routing

- Men kan het ook op volgende manier declareren:
 - name: zelfgekozen naam
 - pattern: bevat patroon (sections)
 - defaults: bevat default waarden voor pattern values
- Onderstaand voorbeeld hebben dezelfde routing.

```
endpoints.MapControllerRoute(  
    "default",  
    "{controller=Home}/{action=Index}/{id?}");  
  
endpoints.MapControllerRoute(  
    name: "default",  
    pattern: "{controller=Home}/{action=Index}/{id?}");  
  
endpoints.MapControllerRoute(  
    name: "default",  
    pattern: "{controller=Home}/{action=Index}/{id?}",  
    defaults: new { controller = "Home", action = "Index"});
```

2. Hello MVC : Routing

► De routing configuratie

- *MapControllerRoute* : definieert Controller based routing. Elke route patroon heeft een naam (willekeurig te kiezen), een url patroon en eventueel een default, constraints of datatokens.
- Definieer alle route patronen voor je applicatie in *routes*
- Bij een binnenkomende request zal URL routing de patronen 1 voor 1 overlopen. Het eerste patroon die matcht met de request wordt uitgevoerd. Plaats dus de meest specifieke route bovenaan.
- Routes worden ook gebruikt om URLs te genereren in responses. Door gebruik van route values en een pattern wordt de url gegenereerd
- Routing is onderdeel van de middleware pipeline dmv de RouterMiddleware class.
- Je kan ook een Route toevoegen aan een Controller (attribute based routing)

2. Hello MVC : Routing

► De routing configuratie

- Bij binnengkomende request wordt in de endpoints gezocht naar de eerste match (volgorde is belangrijk)
- Voorbeeld

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        "regions",
        "Regions/{name?}",
        new { controller = "Brewer", action = "Search" });
});
```

```
endpoints.MapControllerRoute(
    "default",
    "{controller=Home}/{action=Index}/{id?}");
```

```
});
```

url /Regions/Oostvlaanderen =>
Controller Brewer,
Action=Search, parameter
name=Oostvlaanderen

2. Hello MVC : Routing

▶ Attribute based routing

- Voor speciale gevallen in routing
- Boven de controller of actionname
 - [Route("home/about")]
 - Public class AboutController{
 - [Route("phone")]
 - Public string Phone(){}
- Of als je werkt met controller naam
 - [Route(company/[controller])]
- Meer op <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/routing?view=aspnetcore-3.0#use-routing-middleware>

```
[Route("home/about")]
public IActionResult About()
{
    //...
}
```

2. Hello MVC – Routing

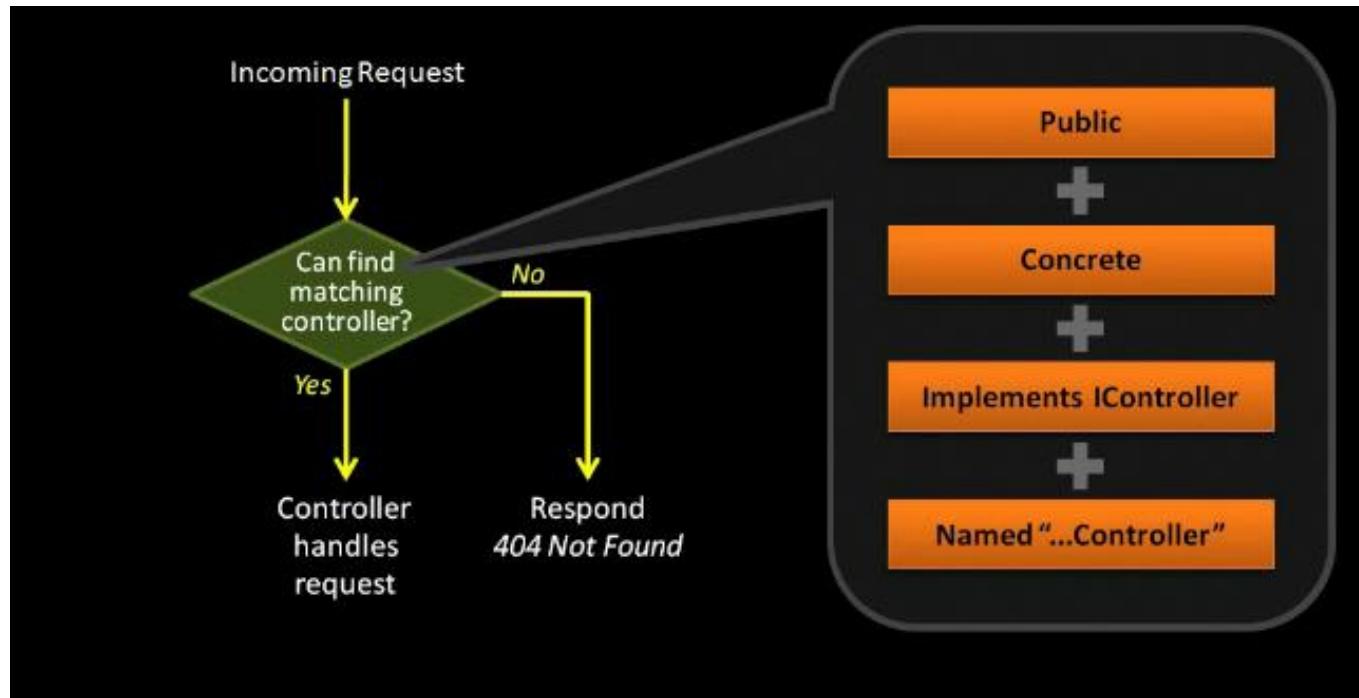
- ▶ Run de applicatie. Surf eens naar .../Shop/Contact. Je krijgt een 404 status pagina te zien....

Als je in StartUp Configure voor app.UseStaticFiles, app.UseStatusCodePages() toevoegt, wordt een 404 status pagina getoond



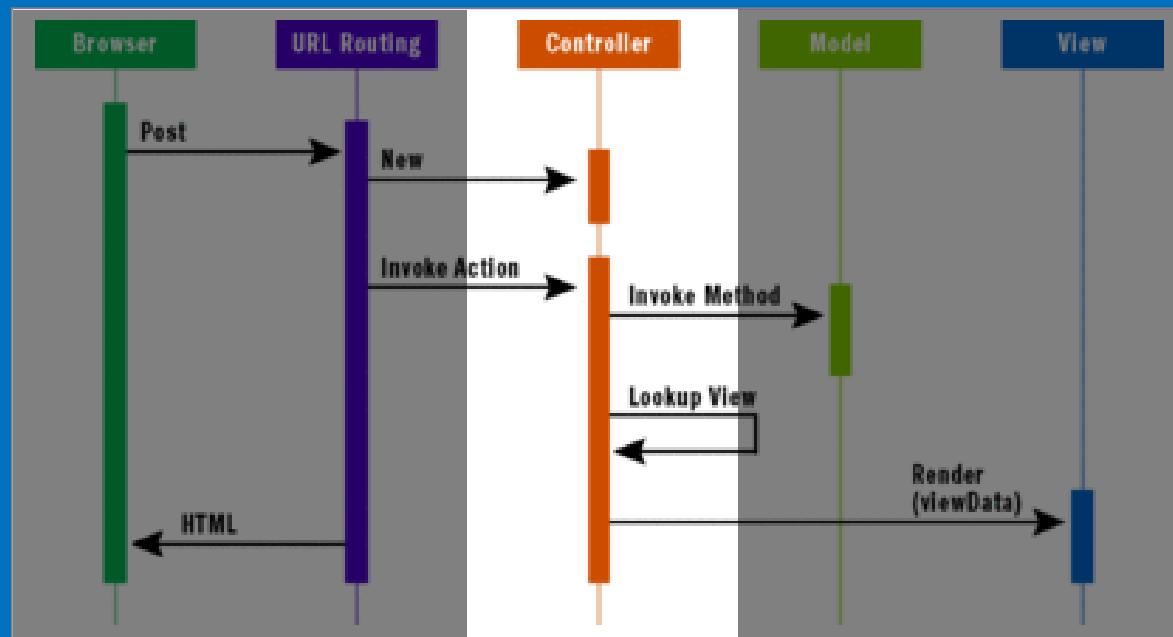
2. Hello MVC : Routing

- Meer over routing: <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/routing?view=aspnetcore-3.0#routing-basics>



Een eerste MVC applicatie

Hello MVC - De Controller



2. Hello MVC: Controller

▶ Controller

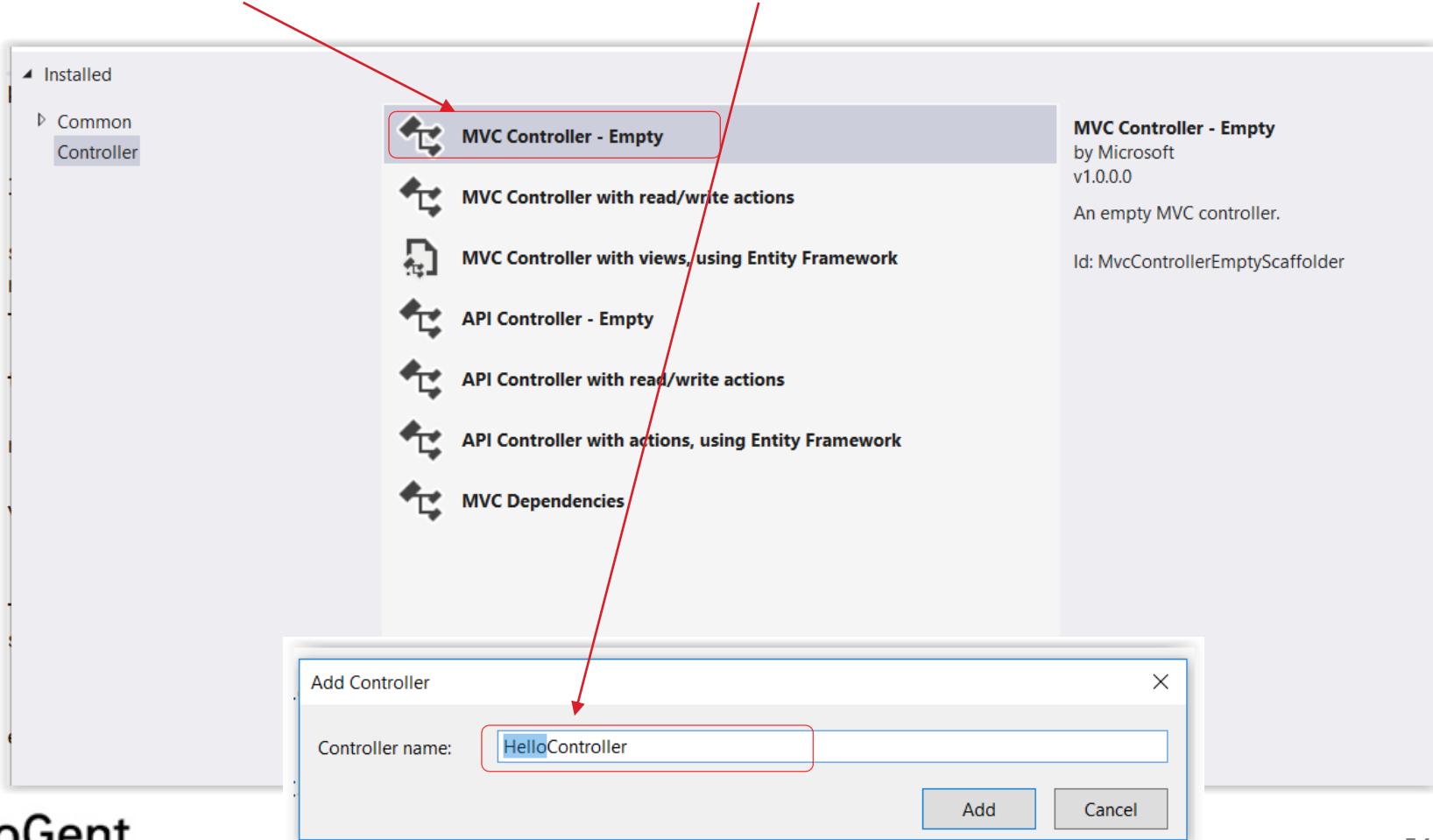
- Verantwoordelijk voor
 - Verwerken binnenkomende **HTTP requests**
 - Verwerken van de input van de gebruiker, uitvoeren van de applicatielogica. Eventueel ophalen en opslaan van de gegevens.
 - Bepalen van de **HTTP response** (het antwoord naar de browser : html pagina, downloaden van een bestand, redirect naar een andere url,...) en de **data** die daarbinnen moet worden weergegeven.
- Is een klasse met de actie methodes
- Meestal 1 controller per use case. Bevat in feite de systeemoperaties uit het SSD vertaald naar de **naming conventions** voor “Action methods” in ASP.NET MVC

2. Hello MVC: Controller

▶ Aanmaken Controller

- Rechtsklik op map **Controllers** > Add > Controller

Kies voor Empty. Vervolgens HelloController als naam.



2. Hello MVC: Controller

► Aanmaken controller

```
using Microsoft.AspNetCore.Mvc;  
namespace HelloMVC.Controllers  
{  
    public class HelloController : Controller  
    {  
        public IActionResult Index()  
        {  
            return View();  
        }  
    }  
}
```

- Maakt in map Controllers een bestand HelloController.cs met de klasse HelloController aan.
 - **Convention over configuration:** de naam van een Controller klasse moet eindigen op Controller.
 - Erft van klasse Controller uit **Microsoft.AspNetCore.Mvc**
 - Elke publieke methode = **action** en kan dus opgeroepen worden adhv url. Deze method zal dan worden uitgevoerd en op het einde een View retourneren

2. Hello MVC: Controller

▶ Controller aanpassen

- Twee action methodes die tekst retourneren:
 - Index():string
 - About(int id):string
 - Naam parameter moet id zijn, zie Startup.cs

```
0 references | 0 changes | 0 authors, 0 changes | 0 requests | 0 exceptions
public string Index()
{
    return "Hello from index in HelloController";
}..
0 references | 0 changes | 0 authors, 0 changes | 0 requests | 0 exceptions
public string About(int id)
{
    return $"Hello from About with id={id}";
}
```

```
app.UseMvc(routes =>
{
    routes.MapRoute(
        name: "default",
        template: "{controller=Home}/{action=Index}/{id?}");
});
```

2. Hello MVC: Controller

- ▶ Run de applicatie
- ▶ Bekijk hoe de routing de mapping tussen URL en action method in Controller verzorgt
 - Surf naar (poort varieert, hier 1234)
 - <http://localhost:1234>Hello>
 - <http://localhost:1234>Hello/Index>
 - <http://localhost:1234>Hello/About/1>
 - <http://localhost:1234>Hello/About?id=10>
 - > *querystring wordt naar parameters in methode vertaald*

2. Hello MVC: Controller

- ▶ Probeer onderstaandeUrls uit. Wat gebeurt bij 4,5,6?
 1. <http://localhost:1234>Hello>
 2. <http://localhost:1234>Hello/Index>
 3. <http://localhost:1234>Hello/Index/1>
 4. <http://localhost:1234>Hello/About>
 5. <http://localhost:1234>Hello/About/abc>
 6. <http://localhost:1234>Hello/About?id=abc>
 7. <http://localhost:1234>Hello/Contact>

2. Hello MVC: Controller - action method

► De action method

- De action methode **handelt een request af**. Elke publieke methode in een controller is callable als een HTTP endpoint. Een HTTP endpoint is een **url**
- En moet een **antwoord** (response) **terugsturen** naar de browser.
 - De response kan een HTML pagina zijn, maar ook een XML bestand, een JSON bestand, een HTTP redirect , ...
 - De actie methode retourneert hiervoor steeds een instantie van **ActionResult** (een default implementatie van de interface **IActionResult**). Deze instantie zal verder worden uitgevoerd door het MVC framework
(<https://docs.asp.net/projects/api/en/latest/autoapi/Microsoft/AspNetCore/Mvc/IActionResult/>)

2. Hello MVC: Controller - action method

► IActionResult: [https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.iactionresult?f1url=https%3A%2F%2Fmsdn.microsoft.com%2Fquery%2Fdev16.query%3FappId%3DDev16IDEF1%26l%3DEN-US%26k%3Dk\(Microsoft.AspNetCore.Mvc.IActionResult\);k\(DevLang-csharp\)%26rd%3Dtrue&view=aspnetcore-2.2](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.iactionresult?f1url=https%3A%2F%2Fmsdn.microsoft.com%2Fquery%2Fdev16.query%3FappId%3DDev16IDEF1%26l%3DEN-US%26k%3Dk(Microsoft.AspNetCore.Mvc.IActionResult);k(DevLang-csharp)%26rd%3Dtrue&view=aspnetcore-2.2)

```
namespace Microsoft.AspNetCore.Mvc
{
    //
    // Summary:
    //   Defines a contract that represents the result of an action method.
    public interface IActionResult
    {
        //
        // Summary:
        //   Executes the result operation of the action method asynchronously. This method
        //   is called by MVC to process the result of an action method.
        //
        // Parameters:
        //   context:
        //     The context in which the result is executed. The context information includes
        //     information about the action that was executed and request information.
        //
        // Returns:
        //   A task that represents the asynchronous execute operation.
        Task ExecuteResultAsync(ActionContext context);
    }
}
```

2. Hello MVC: Controller - action method

▶ ActionResult:

- Een default implementation van [Microsoft.AspNetCore.Mvc.IActionResult](#).
- Een abstracte basisklasse met verschillende concrete subklassen
 - Elke subklasse is een andere afhandeling van een request
- Een instantie wordt door het framework uitgevoerd (**Command pattern**) en zal een resultaat terugsturen naar de browser. De controller moet hem hiervoor alle nodige informatie aanleveren

The screenshot shows a code snippet from the Microsoft Docs API reference for the `ActionResult` class. The `ExecuteResult` method is highlighted in blue. Below the code, there is a summary and a parameters section.

```
public virtual void ExecuteResult(Microsoft.AspNetCore.Mvc.ActionContext context)
    Member of Microsoft.AspNetCore.Mvc.ActionResult
```

Summary:
Executes the result operation of the action method synchronously. This method is called by MVC to process the result of an action method.

Parameters:
`context`: The context in which the result is executed. The context information includes information about the action that was executed and request information.

2. Hello MVC: Controller - action method

▶ ActionResult: [https://docs.microsoft.com/en-](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.actionresult?f1url=https%3A%2F%2Fmsdn.microsoft.com%2Fquery%2Fdev16.query%3FappId%3DDev16IDEF1%26l%3DEN-US%26k%3Dk(Microsoft.AspNetCore.Mvc.ActionResult);k(DevLang-csharp)%26rd%3Dtrue&view=aspnetcore-2.2)

[https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.actionresult?f1url=https%3A%2F%2Fmsdn.microsoft.com%2Fquery%2Fdev16.query%3FappId%3DDev16IDEF1%26l%3DEN-US%26k%3Dk\(Microsoft.AspNetCore.Mvc.ActionResult\);k\(DevLang-csharp\)%26rd%3Dtrue&view=aspnetcore-2.2](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.actionresult?f1url=https%3A%2F%2Fmsdn.microsoft.com%2Fquery%2Fdev16.query%3FappId%3DDev16IDEF1%26l%3DEN-US%26k%3Dk(Microsoft.AspNetCore.Mvc.ActionResult);k(DevLang-csharp)%26rd%3Dtrue&view=aspnetcore-2.2)

```
public abstract class ActionResult : IActionResult
```

```
class Microsoft.AspNetCore.Mvc ActionResult
```

Methods ↴

ExecuteResult(*Microsoft.AspNetCore.Mvc.ActionContext*)

Executes the result operation of the action method synchronously. This method is called by MVC to process the result of an action method.

Arguments: *context* (*Microsoft.AspNetCore.Mvc.ActionContext*) – The context in which the result is executed. The context information includes information about the action that was executed and request information.

```
public virtual void ExecuteResult(ActionContext context)
```

2. Hello MVC: Controller - action method

▶ ActionResult subklassen

ContentResult	the action result sends a response whose body contains a specified object
EmptyResult	Represents an ActionResult that when executed will do nothing.
FileResult	Represents an ActionResult that when executed will write a file as the response.
JsonResult	An action result which formats the given object as JSON.
ObjectResult	An ActionResult that will use content negotiation to send an object to the client
RedirectResult	Sends a response with the HTTP 301, 302, 307 or 308 status code, redirecting the client to a new url
RedirectActionResult	redirects the client to a specific action in the controller
RedirectToActionResult	redirects the client to a specific page
RedirectToRouteResult	redirects the client to a URL generated from a specific route
LocalRedirectResult	An ActionResult that returns a Found (302), Moved Permanently (301), Temporary Redirect (307), or Permanent Redirect (308) response with a Location header to the supplied local URL.
StatusCodeResult	Represents an ActionResult that when executed will produce an HTTP response with the given response status code. Bevat subklassen OkResult, BadRequestResult, NotFoundResult,...

2. Hello MVC: Controller - action method

▶ ActionResult subklassen

ViewComponentResult	An ActionResult which renders a view component to the response.
ViewResult	Represents an ActionResult that renders a view to the response
PartialViewResult	Represents an ActionResult that renders a partial view to the response.
PageResult	An ActionResult that renders a Razor Page.
ForbidResult	An ActionResult that on execution invokes AuthenticationManager.ForbidAsync.
ChallengeResult	An ActionResult that on execution invokes AuthenticationManager.ChallengeAsync.
SignInResult	An ActionResult that on execution invokes AuthenticationManager.SignInAsync.
SignOutResult	An ActionResult that on execution invokes AuthenticationManager.SignOutAsync.

2. Hello MVC: Controller - action method

▶ ActionResult en de helper methodes

- De controller maakt gebruik van de **helper methodes** voor het aanmaken van een concrete instantie van een ActionResult
- De methode View maakt een ViewResult object aan

Name	Description
View(): ViewResult	Creates a ViewResult object that renders the default view to the response.
View(Object): ViewResult	Creates a ViewResult object by using the model that renders the default view to
View(String, Object): ViewResult	Creates a ViewResult object by using the model that renders the view with the specified name to the response.

- <https://docs.asp.net/projects/api/en/latest/autoapi/Microsoft/AspNetCore/Mvc/Controller/index.html?highlight=Controller>

2. Hello M

► De klasse ViewResult

```
// Represents an Microsoft.AspNetCore.Mvc.ActionResult that renders a view to the
// response.
public class ViewResult : ActionResult, IActionResult, IStatusCodeActionResult
{
    public ViewResult();
    // Gets or sets the Content-Type header for the response.
    public string ContentType { get; set; }

    // Gets the view data model.
    public object Model { get; }

    // Gets or sets the HTTP status code.
    public int? StatusCode { get; set; }

    // Gets or sets the Microsoft.AspNetCore.Mvc.ViewFeatures.ITempDataDictionary for
    // this result.
    public ITempDataDictionary TempData { get; set; }

    // Gets or sets the Microsoft.AspNetCore.Mvc.ViewFeatures.ViewDataDictionary for this result.
    public ViewDataDictionary ViewData { get; set; }

    // Gets or sets the Microsoft.AspNetCore.Mvc.ViewEngines.IViewEngine used to locate views.
    // When null, an instance of Microsoft.AspNetCore.Mvc.ViewEngines.ICompositeViewEngine
    // from ActionContext.HttpContext.RequestServices is used.
    public IViewEngine ViewEngine { get; set; }

    // Gets or sets the name or path of the view that is rendered to the response.
    // When null, defaults to Microsoft.AspNetCore.Mvc.Controllers.ControllerActionDescriptor.ActionName
    public string ViewName { get; set; }    //
    [DebuggerStepThrough]
    public override Task ExecuteResultAsync(ActionContext context);
}
```

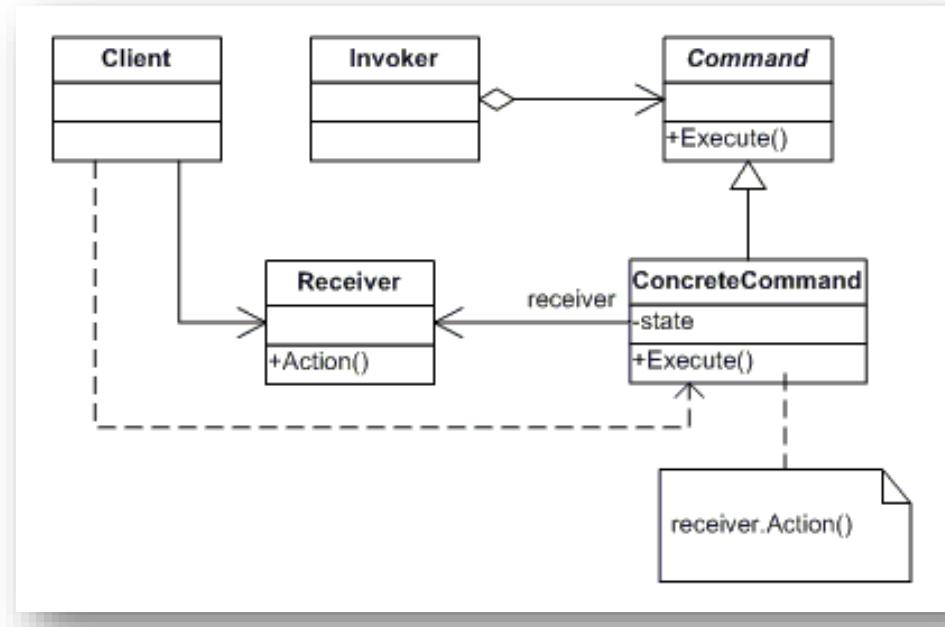
2. Hello MVC: Controller - action method

► De subklasse ViewResult

Name	Description
ContentType	Gets or sets the Content-Type header for the response.
Model	Gets the view data model.
StatusCode	Gets or sets the HTTP status code.
TempData	Gets or sets the Microsoft.AspNetCore.Mvc.ViewFeatures.ITempDataDictionary for this result.
ViewData	Gets or sets the Microsoft.AspNetCore.Mvc.ViewFeatures.ViewDataDictionary for this result.
ViewEngine	Gets or sets the Microsoft.AspNetCore.Mvc.ViewEngines.IViewEngine used to locate views
ViewName	Gets or sets the name of the view to render.
ExecuteResultAsync	Executes the result operation of the action method asynchronously. This method is called by MVC to process the result of an action method. The default implementation of this method calls the ExecuteResult method and returns a completed task.

2. Hello MVC: Controller - action method

- ▶ ActionResult en het command pattern
 - Command = ActionResult met ExecuteResult methode
 - ConcreteCommand = ViewResult, JsonResult,....



Het **Command Pattern** schermt een aanroep af door middel van een object, waarbij je verschillende aanroepen in verschillende objecten kan opbergen, in een queue kan zetten of op schijf kan bewaren;

2. Hello MVC: Controller - action method

- ▶ Pas de controller als volgt aan

```
using Microsoft.AspNetCore.Mvc;
namespace HelloMVC.Controllers
{
    public class HelloController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }
        public IActionResult About(int id)
        {
            return Content($"Hello from about {id}");
        }
    }
}
```

- **View()** : MVC framework rendert de default View. MVC maakt hiervoor gebruik van **naming conventions**

2. Hello MVC: Controller

▶ Run de applicatie ...



An unhandled exception occurred while processing the request.

InvalidOperationException: The view 'Index' was not found. The following locations were searched:
/Views/Hello/Index.cshtml
/Views/Shared/Index.cshtml

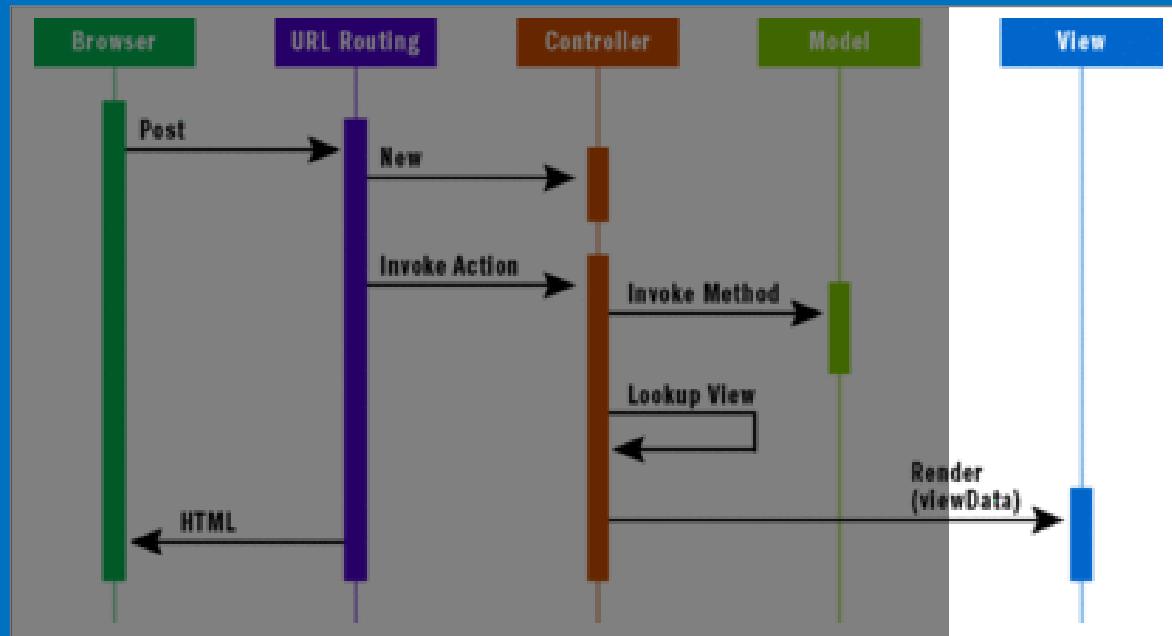
EnsureSuccessful

return View() geeft de instructie aan MVC framework om de **default view** van de action methode te renderen. Views die bij een actie horen worden automatisch gevonden als ze in de directory gezet worden met de naam van de Controller of in de Shared folder geplaatst worden binnen de Views folder. (=**Convention over configuration**)

Meer over controllers toevoegen: <https://docs.asp.net/en/latest/tutorials/first-mvc-app/adding-controller.html>

Een eerste MVC applicatie

Hello MVC - De View



2. Hello MVC : View

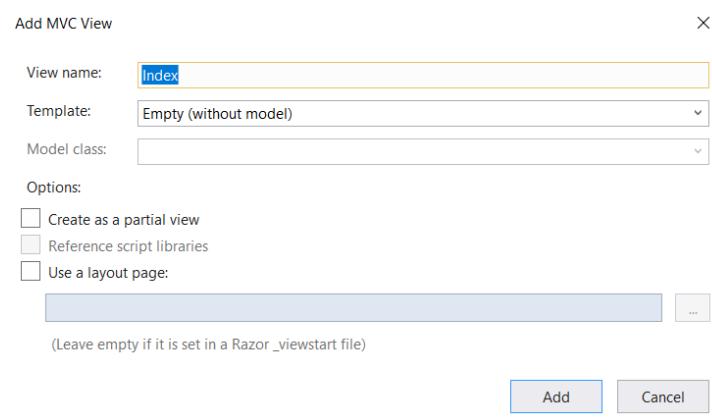
- ▶ View: de UI van de applicatie
 - Bevat **enkel en alleen presentatielogica**
 - Een **view** is een combinatie van html, css, javascript en Razor (C#). Razor code dient om de data die de Controller aan de View doorgeeft (**model, ViewBag**) binnen de HTML pagina te plaatsen.
 - De action methode creëert een **ViewResult**, die wordt uitgevoerd door de View Engine van het MVC Framework. Deze zal de View renderen en het resultaat (een html pagina) naar de browser zal sturen.
 - De default view engine is the Razor View Engine.



2. Hello MVC : View

▶ Creatie van een View

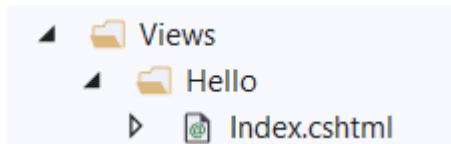
- Vervolgens maak je per methode in de Controller die een view retourneert een view aan.
- Rechtsklik de methode Index > Add View.
 - De naam van de view is de naam van de methode uit de controller, extensie is .cshtml
 - Men kan kiezen uit meerdere templates. Kies **Empty (without model)**. Vink **Use a layout page** uit (Vink use layout page uit)
- De code voor de View wordt gegenereerd



2. Hello MVC : View

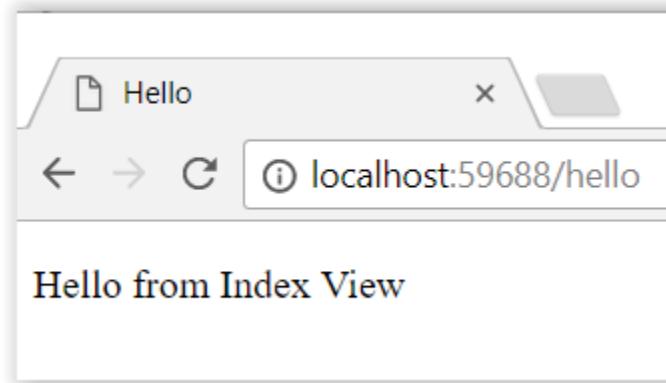
▶ Creatie van een View

- Views worden toegevoegd aan de Views folder.
- per Controller is er een map, met de naam van de controller in de Views folder: *Views/[ControllerName]* map
- Per methode in de Controller die een View retourneert is er (in principe) een view. De naam van de view is de naam van de methode uit de Controller, extensie is .cshtml



2. Hello MVC : View

- De view is een standaard template voor een html pagina.
- Bovenaan staat aangeduid (@ = razor syntax) dat er geen gemeenschappelijke layout (zie later) gebruikt wordt.
- Voeg een title “Hello” toe in de html. En een p-tag in de body:
`<p>Hello from Index View</p>`
- Run applicatie.



- Het is natuurlijk niet de bedoeling dat we enkel html gaan gebruiken in de view. Dan kunnen we ook static pages gebruiken.

2. Hello MVC : View

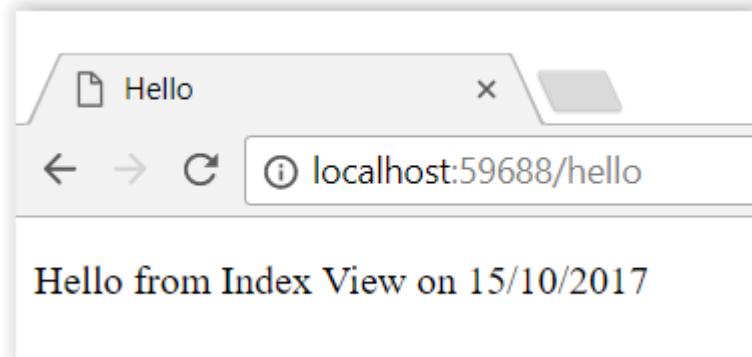
▶ Razor View Engine

- Razor is een markup syntax voor het toevoegen van server based code in een webpagina. De Razor syntax bestaat uit Razor markup, C# en HTML.
- Rendert html
- Razor syntax ondersteunt C#. Gebruikt **@** om van HTML over te gaan naar C#
 - Code nuget: **@** gevolgd door 1 C# instructie die inline geëvalueerd en gerenderd wordt. Razor herkent zelf einde van instructie
 - Code blok : **@{...}**. bevat meerdere C# instructies
- Voorbeeldjes
 - `<div> Hello MVC, @DateTime.Today.ToShortDateString() :</div>`
 - Razor weet hier dat een spatie na de methode ToShortDateString() geen geldige identifier is, dus schakelt hij terug over naar HTML
 - `@{Layout=null}`
 - Maak geen gebruik van de gemeenschappelijke layout

2. Hello MVC : View

- ▶ Pas de code aan in index.cshtml

```
<p>Hello from Index View on @DateTime.Today.ToShortDateString()</p>
```



2. Hello MVC : View

▶ View en dynamische output

- Taak van **Controller** om de **data** voor de View te genereren
- Taak van **View** om die data in de html pagina weer te geven
- Controller bevat property **ViewBag** van type **dynamic**.
“Dynamic means you can dynamically get/set values and add any number of additional fields, properties, methods without need of strongly-typed classes. Properties on a dynamic object are defined at run-time, so to add a new property you just use it”
- In de **ViewBag** kan informatie geplaatst worden die doorgegeven wordt aan de View page bij aanroep van de View() methode. Maak hiervoor gewoon een property aan voor de ViewBag en plaats er de waarde in.

2. Hello MVC : View

▶ View en dynamische output

- Controller genereert de data die de View zal moeten renderen

```
0 references | 0 changes | 0 authors, 0 changes | 16 requests | 0 exceptions
public IActionResult Index()
{
    ViewBag.Greeting = DateTime.Now.Hour <= 12 ? "Good morning!!" : "Good afternoon";
    return View();
}
```

- Index.cshtml

```
<p>Hello from Index View on @DateTime.Today.ToShortDateString()</p>
<p>@ViewBag.Greeting</p>
```

Hello from Index View on 05-Oct-19

Good morning!!

2. Hello MVC : View

▶ View en dynamische output

- De View kan de ViewBag lezen (is ook een property van View)
 - @: Razor code nugget
 - Start met een @ teken
 - gevolgd door de C# statement die html/tekst retourneert
 - Razor engine zal C# instructie uitvoeren en het resultaat sturen naar de output en bovendien html encoden.
 - < wordt vertaald naar < en > naar > ...

The screenshot shows a development environment with two windows. On the left is a code editor displaying C# code for an Index action method. On the right is a browser window showing the resulting page output.

Code Editor (Index.cs)

```
0 references | 0 changes | 0 authors, 0 changes | 1 request | 0 exceptions
public IActionResult Index()
{
    ViewBag.Greeting = "<script>alert('Hello from script')</script>";
    return View();
}
```

Browser Output

Hello from Index View on 05-Oct-19

<script>alert('Hello from script')</script>

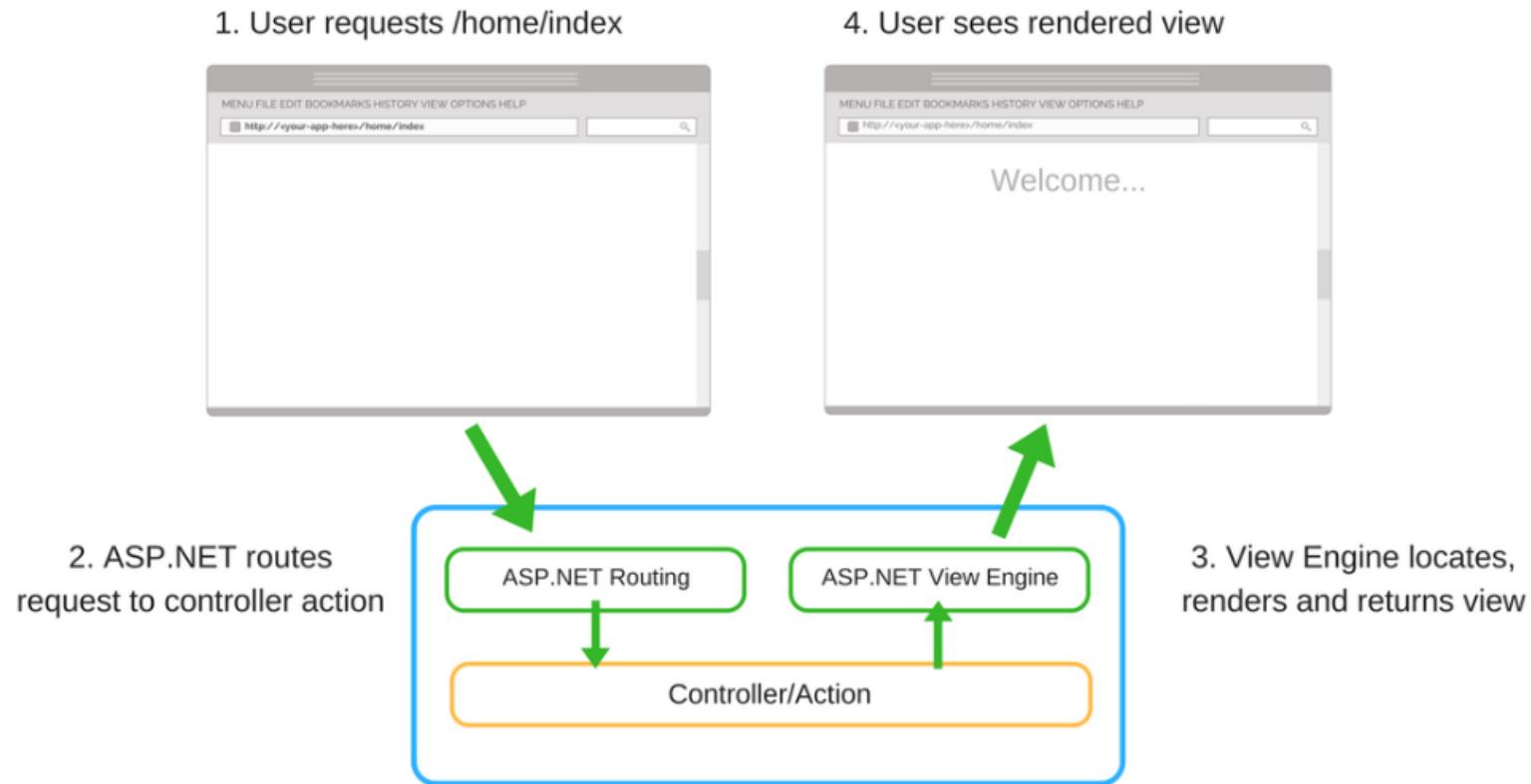
2. Hello MVC: Controller - View

▶ Samenvatting

- Een actie methode in de Controller handelt een request af en stuurt een resultaat, **response** genaamd, (meestal html) terug naar de browser.
- Indien we HTML retourneren naar de browser moet de actie methode een **ViewResult** object retourneren. Dit erft van de basisklasse **ActionResult**.
- **ActionResult** is de abstracte basisklasse voor alles wat een action methode naar de browser kan retourneren. Hier wordt het **Command Pattern** gebruikt. Een ActionResult stelt een **Command** voor, die het resultaat van een action methode bevat en dat het framework zal uitvoeren in naam van de action methode.
- Voor het aanmaken van een ViewResult roept een actie methode de **View()** methode aan.
- **View()** maakt een instantie van **ViewResult** (subklasse van ActionResult) aan en geeft deze aan het MVC framework die ExecuteResult aanroeft. Hierdoor wordt een HTML pagina gegenereerd o.b.v. een View template (.cshtml) en de data aangeleverd door de controller en wordt deze naar de browser gestuurd.

2. Hello MVC: Controller - View

▶ Samenvatting



Hilton, J. (2019). MVC vs Razor Pages - A quick comparison. Retrieved from <https://jonhilton.net/razor-pages-or-mvc-a-quick-comparison/>

Een eerste MVC applicatie

Snake Eyes - Een demo...



HoGent

3. SnakeEyes

- ▶ Demo SNAKE Eyes GAME



- Ofwel start je met een nieuw project
- Ofwel clone je <https://github.com/WebIII/06thmvcintroduction> en kan je inpikken bij een commit door het aanmaken van een branch

3. SnakeEyes

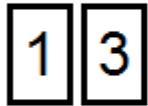
- ▶ Demo SNAKE Eyes GAME
 - Ontwerp
 - De starter MVC web applicatie
 - Model
 - Controller
 - View

3. SnakeEyes : Ontwerp

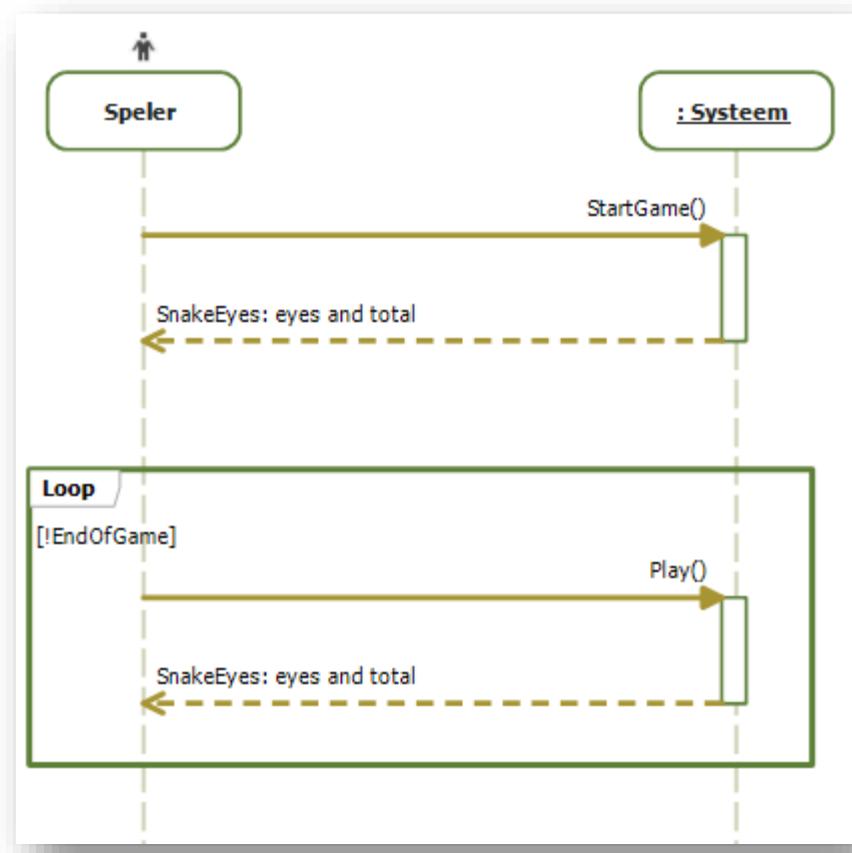
► Ontwerp

Snake Eyes

Keep going but don't get 2 one's

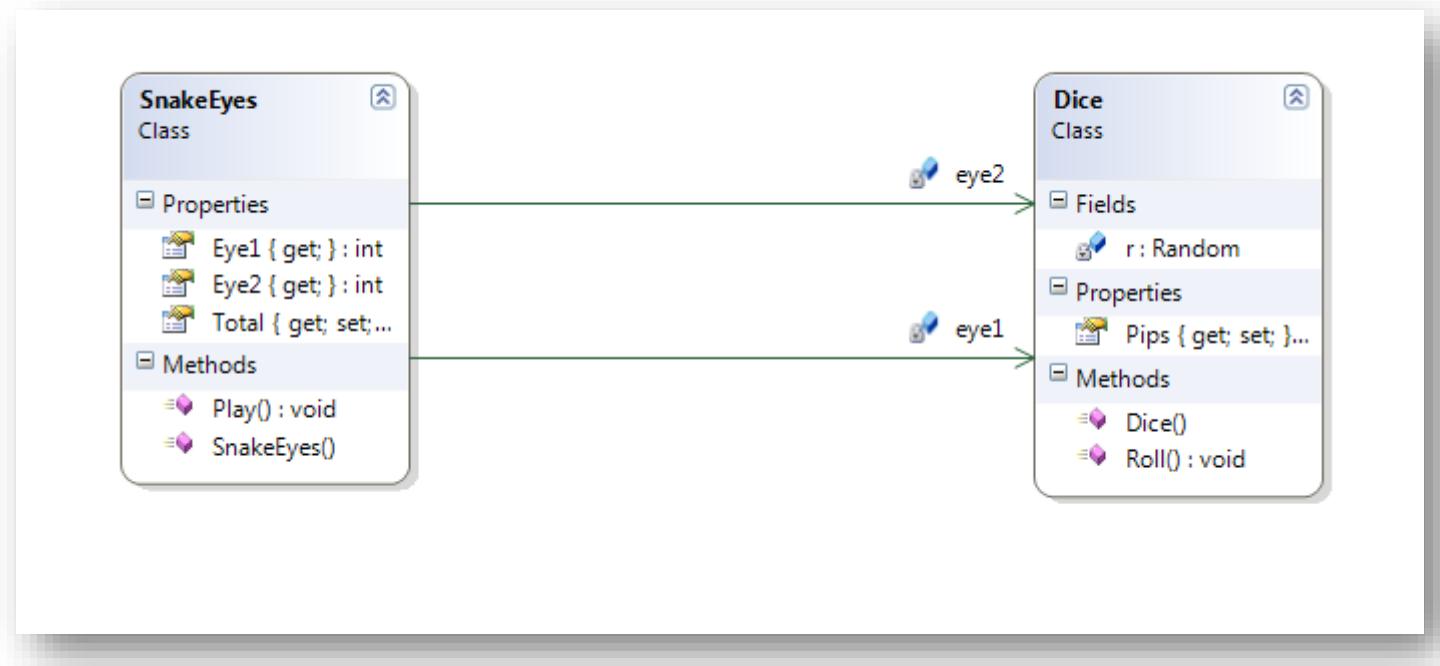


Total : 29



3. SnakeEyes : Ontwerp

▶ Ontwerp domain



3. SnakeEyes : nieuw project

- ▶ Maak een nieuw Project aan, noem dit **SnakeEyesGame**.

Create a new project

Recent project templates

Filtering by: C#, Web [Clear filter](#)

Template	Language	Platform	Project type
ASP.NET Core Web Application	C#	Windows, Linux, macOS	Web
Console App (.NET Core)	C#		
xUnit Test Project (.NET Core)	C#		

ASP.NET Core Web Application
Project templates for creating ASP.NET Core applications for Windows, Linux and macOS using .NET Core or .NET Framework. Create web apps with Razor Pages, MVC, and Blazor, or Single Page Apps (SPA) using Angular, React, or React + Redux. Also create Web APIs, gRPC Services, and Worker Services.

Configure your new project

ASP.NET Core Web Application C# Windows Linux macOS Web

Project name

Location ...

Solution

Solution name

Place solution and project in the same directory

Create a new ASP.NET Core web application

.NET Core ASP.NET Core 3.0

Empty
An empty project template for creating an ASP.NET Core application. This template does not have any content in it.

API
A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.

Web Application
A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content.

Web Application (Model-View-Controller)
A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.

Authentication
No Authentication [Change](#)

Advanced
 Configure for HTTPS
 Enable Docker Support (Requires Docker Desktop)
Linux

3. Snakeyes: mvc project

- ▶ Geef de volgende service mee in Startup.cs

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllersWithViews();
}
```

3. Snakeyes: mvc project

- ▶ Vervolgens configurer je je applicatie door in Startup.cs (Middleware) volgende toe te voegen aan de Configure method (Pipeline).

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.UseRouting();
    app.UseStaticFiles();
    app.UseStatusCodePages();
    app.UseEndpoints(endpoints =>
    {
        endpoints.MapDefaultControllerRoute();
    });
}
```

3. SnakeEyes : Model

- ▶ Maak een nieuwe folder Models aan in je project. Dit bevat de domain klassen
- ▶ Voeg de nieuwe klassen SnakeEyes en Dice toe aan de Models folder.



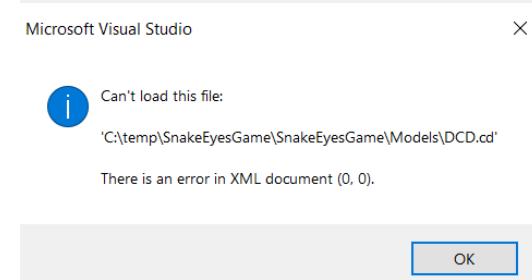
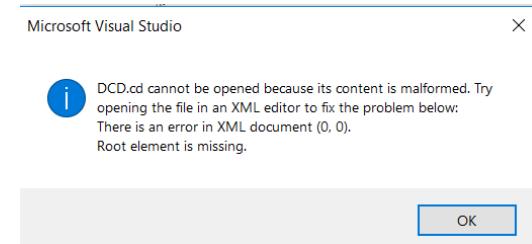
- ▶ Implementeer zowel Dice als SnakeEyes

3. SnakeEyes : Model

- ▶ Aanmaken van een class diagram in .net core web app kan via een omweg 😊
 - Right click Models folder > Add > new Item.
 - Zoek op xml, en kies een xml file. De extensie .cd is belangrijk. Noem het bvb DCD.cd
 - Je krijgt 2 meldingen. Don't worry!
 - Right click DCD.cd > Open with > XML(Text) Editor.
 - Pas de code aan en sla op.

```
<?xml version="1.0" encoding="utf-8" ?>
]<ClassDiagram MajorVersion="1" MinorVersion="1">
    <Font Name="Segoe UI" Size="9" />
</ClassDiagram>
```

- Nu kan je DCD.cd openen en klassen aanmaken/bestaande klassen droppen



3. SnakeEyes : Model

▶ Klasse Dice

```
using System;
namespace SnakeEyesGame.Models
{
    public class Dice
    {
        #region Fields
        private static Random _random = new Random();
        #endregion

        #region Properties
        public int Pips { get; private set; }
        #endregion
    }
}
```

Namespaces die binnen de code gebruikt worden.

Namespace van de klasse Dice

Automatic property

3. SnakeEyes : Model

- Methode **Roll()**
 - Maakt gebruik van een Random generator.
 - Zoek klasse Random op in Help/Object Browser.
 - Hoe genereer je een getal tussen 1 en 6?
 - Declareer attribuut van type Random en maak instantie aan
 - Bij declaratie
 - Of in de constructor. Initialiseer ook Pips op 6.

```
#region Fields
private Random _random;
#endregion
```

```
public Dice()
{
    _random = new Random();
    Pips = 6;
}
```

- Codeer de methode Roll.

3. SnakeEyes : Mode

► Klasse SnakeEyes

- Vervolledig de methode **Play**.
- Merk op :
 - Properties Eye1 en Eye2 hebben geen setter
 - Gebruik Regions

```
public class SnakeEyes
{
    #region Fields
    private readonly Dice _eye1;
    private readonly Dice _eye2;
    #endregion

    #region Properties

    public int Total { get; private set; }

    public int Eye1 => _eye1.Pips;

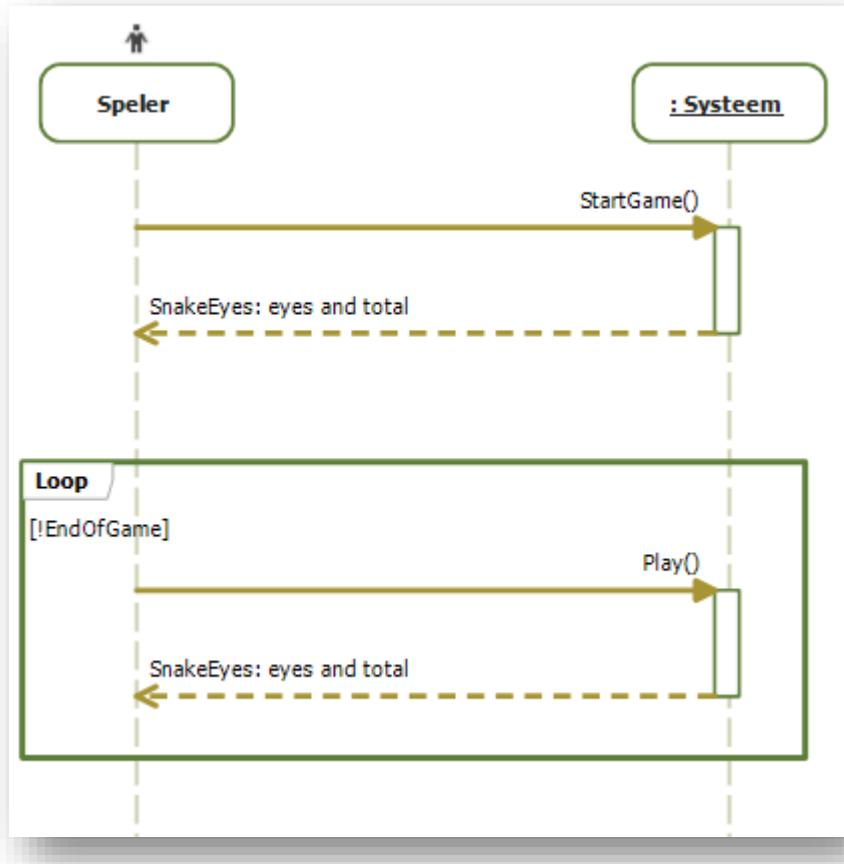
    public int Eye2 => _eye2.Pips;
    #endregion

    #region Constructor
    public SnakeEyes()
    {
        _eye1 = new Dice();
        _eye2 = new Dice();
        Total = 0;
    }
    #endregion

    #region Methods
    /// <summary>
    /// Rolls the dices. Calculates the total.
    /// </summary>
    public void Play() ...
    #endregion
}
```

3. SnakeEyes : Controller

► Ontwerp ?



HomeController

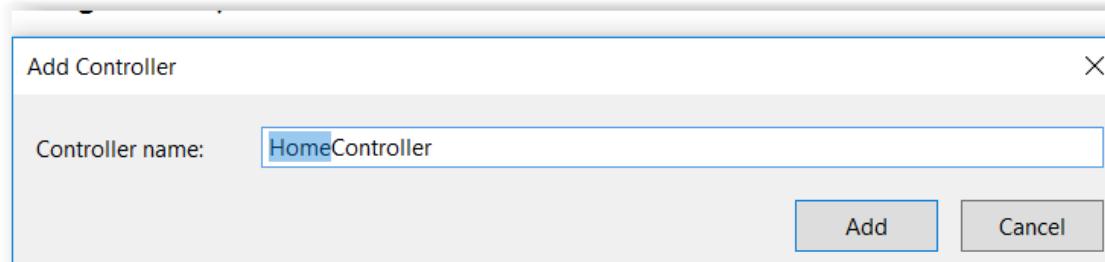
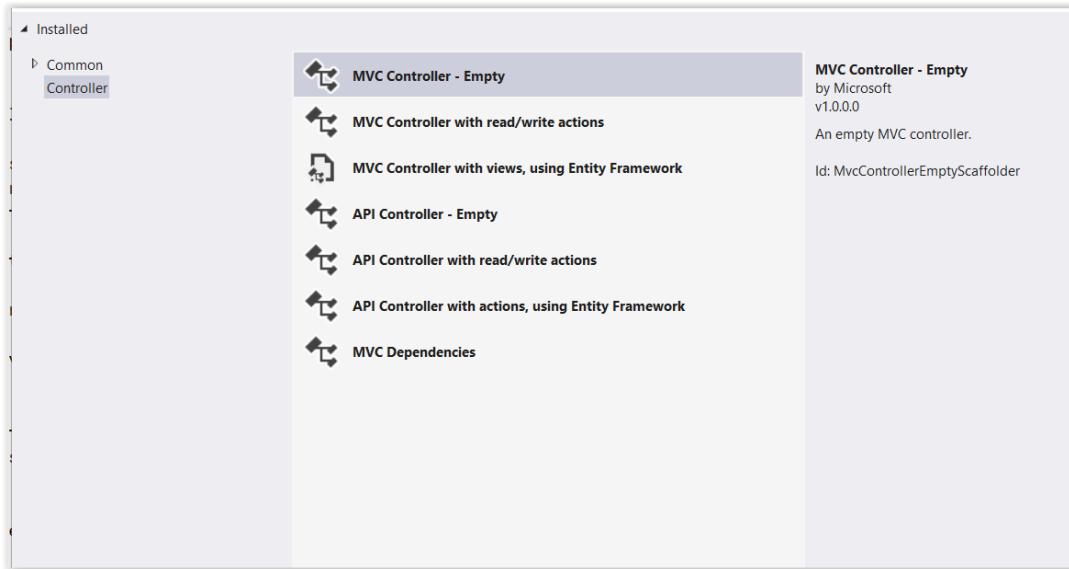
Class
→ Controller

Hou bij de methodenamen van de actiemethodes rekening met de conventies

- Index is beter dan StartSpel
- Play

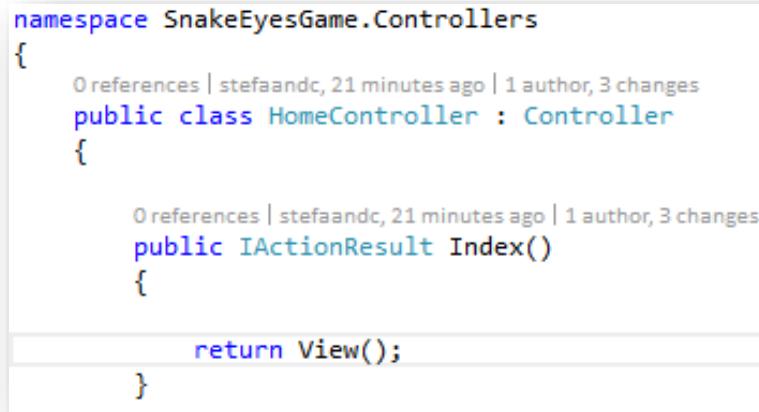
3. SnakeEyes : Controller

- ▶ Maak een nieuwe controller aan : HomeController



3. SnakeEyes : Controller

- ▶ Maak een nieuwe controller aan : HomeController
 - Declareer een **private attribuut _snakeEyes** van type SnakeEyes
 - Declareer een **action method Index**

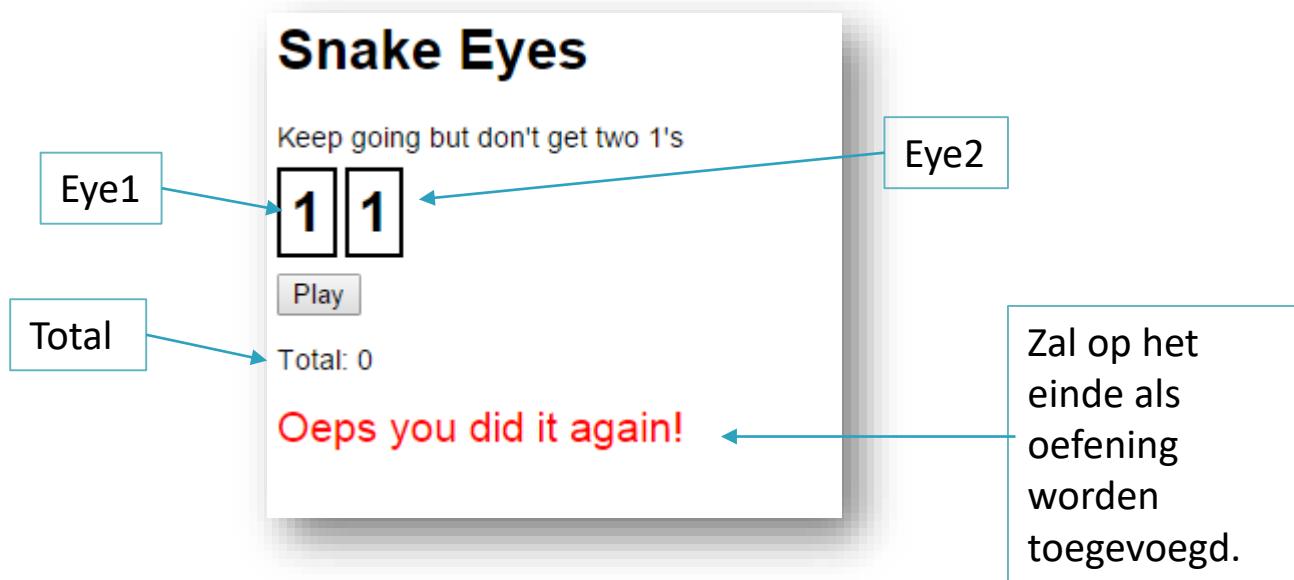


```
namespace SnakeEyesGame.Controllers
{
    public class HomeController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }
    }
}
```

- De Index methode zal
 - een instantie aanmaken van SnakeEyes
 - de methode View(...) aanroepen en de nodige gegevens via een **model** doorgeven

3. SnakeEyes : Controller

- ▶ De domeinklasse SnakeEyes bevat de gegevens die nodig zijn in de UI, de view



3. SnakeEyes : Controller

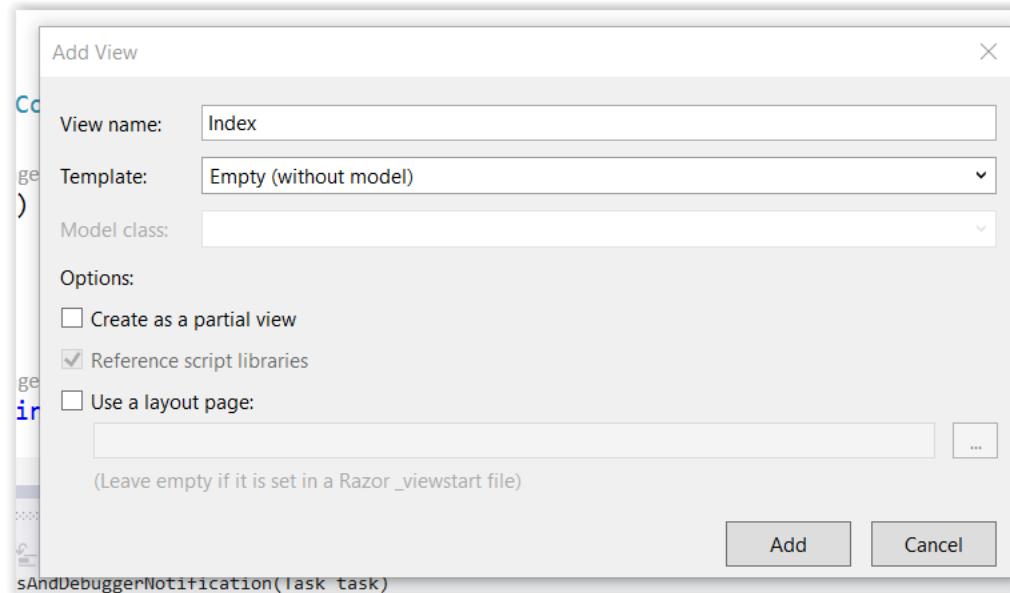
► HomeController - Index :

- Maak een instantie aan van *SnakeEyes*
- Roep View aan en geef een instantie van *SnakeEyes* door. Verwijs naar de juiste **namespace**. Retourneer het resultaat.

```
namespace SnakeEyesGame.Controllers
{
    public class HomeController : Controller
    {
        private SnakeEyes _snakeEyes;
        public IActionResult Index()
        {
            _snakeEyes = new SnakeEyes();
            return View(_snakeEyes);
        }
    }
}
```

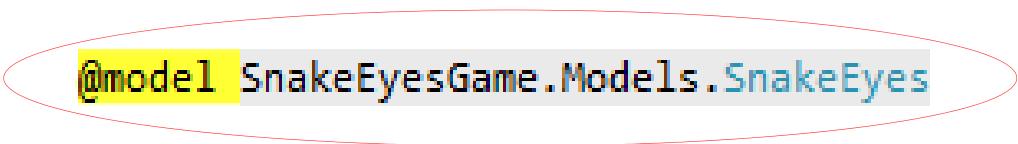
3. SnakeEyes : View

- ▶ Maak nu een nieuwe MVC View aan. Rechtsklik Index method > Add view



3. SnakeEyes : View

- ▶ Geef titel in en voeg bovenaan het model in.



```
@model SnakeEyesGame.Models.SnakeEyes
```

@model = strongly typed versie van ViewPage (generic class). De ViewPage bevat nu een Model.

Merk op

- Gebruik de **fully qualified name**
- Of voeg **using** toe

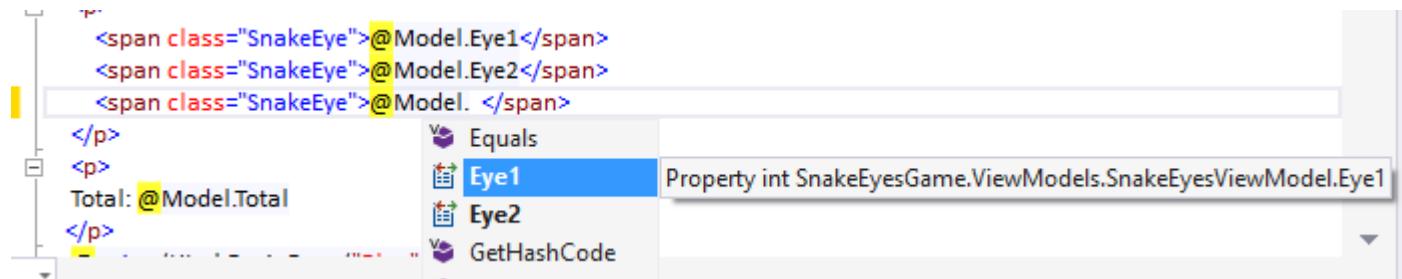
```
@using SnakeEyesGame.Models
```

```
@model SnakeEyes
```

3. SnakeEyes : View

▶ Index.cshtml

- model toegankelijk via Model property van ViewPage
 - Intellisense



3. SnakeEyes : View

▶ Index.cshtml

- Vervolledig de view en run de app

```
@model SnakeEyesGame.Models.SnakeEyes;
{
    Layout = null;
}

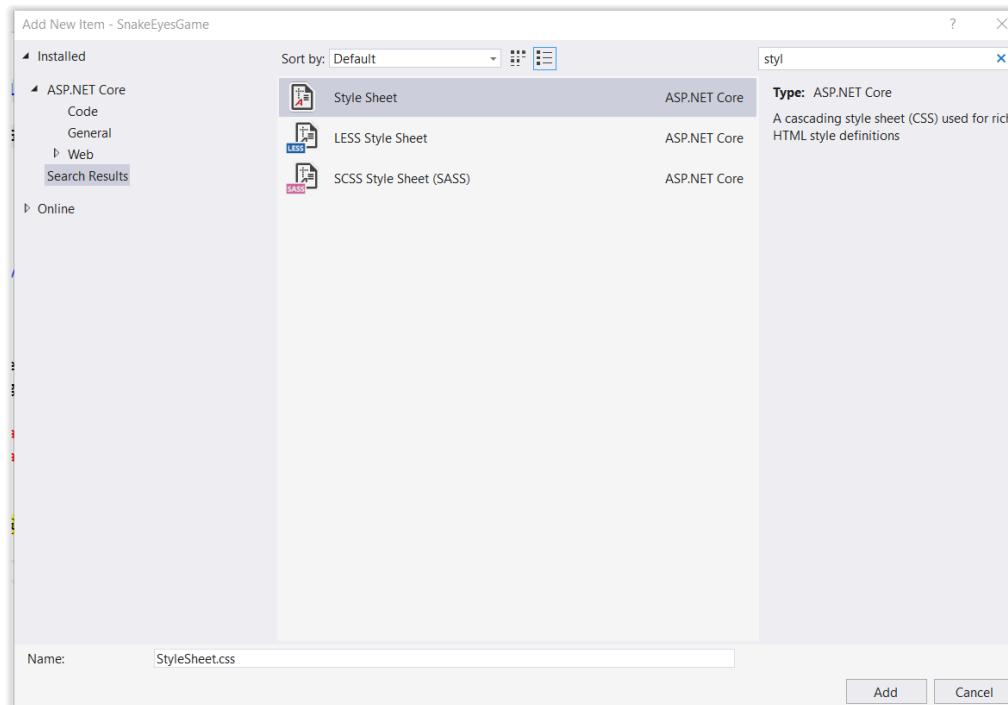
<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Snake Eyes</title>
</head>
<body>
    <h1>Snake Eyes</h1>
    <p>Keep going but don't get 2 one's</p>
    <p>@Model.Eye1</p>
    <p>@Model.Eye2</p>
    <p>Total : @Model.Total</p>
</body>
</html>
```

3. SnakeEyes : View

▶ Toevoegen stylesheet :

- Voeg een folder wwwroot toe (**conventie**). Hierbinnen een folder css.
- Maak een bestand site.css aan (Rechtsklik css > Add > New Item > Stylesheet en geef naam site.css in)



3. SnakeEyes : View

▶ Toevoegen stylesheet :

- Open site.css
- Creatie nieuwe CSS rule. Geef onderstaande stijlregels in

```
.snakeEye {  
    border: 2px solid black;  
    font-size: 2em;  
    font-weight: bold;  
    padding: 5px;  
}
```

- Pas toe in de view Index.cshtml

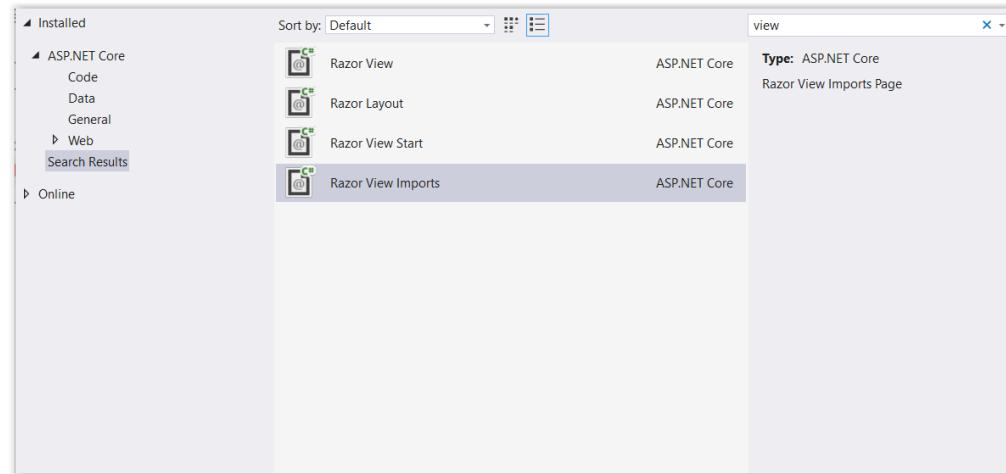
```
<link href="~/css/site.css" rel="stylesheet" />  
  
<span class="SnakeEye">@Model.Eye1</span>  
<span class="SnakeEye">@Model.Eye2</span>
```

- Run de app

3. SnakeEyes : View

► Toevoegen van een formulier aan View voor Play knop

- Hier wordt gebruik gemaakt van **tag helpers**.
- Hiervoor dien je in de View Map een MVC View Imports bestand toe te voegen: `_ViewImports.cshtml`. Zo kan je taghelpers gebruiken in elke view. Voeg volgende toe aan de Views folder:



- Voeg code toe

```
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

Views can use Razor directives to do many things, such as importing namespaces or performing dependency injection. Directives shared by many views may be specified in a common

`_ViewImports.cshtml` file. The `_ViewImports` file supports the following directives:

3. SnakeEyes : View

- ▶ Toevoegen van een formulier aan View voor Play knop
 - <https://docs.asp.net/en/latest/mvc/views/tag-helpers/intro.html>

Views can use Razor directives to do many things, such as importing namespaces or performing dependency injection. Directives shared by many views may be specified in a common

`_ViewImports.cshtml`

The `@addTagHelper` directive makes Tag Helpers available to the view. In this case, the view file is `Views/_ViewImports.cshtml`, which by default is inherited by all view files in the `Views` folder and sub-directories; making Tag Helpers available. The code above uses the wildcard syntax ("*") to specify that all Tag Helpers in the specified assembly (`Microsoft.AspNetCore.Mvc.TagHelpers`) will be available to every view file in the `Views` directory or sub-directory. The first parameter after `@addTagHelper` specifies the Tag Helpers to load (we are using "*" for all Tag Helpers), and the second parameter "`Microsoft.AspNetCore.Mvc.TagHelpers`" specifies the assembly containing the Tag Helpers. `Microsoft.AspNetCore.Mvc.TagHelpers` is the assembly for the built-in ASP.NET Core Tag Helpers.

3. SnakeEyes : View

- ▶ Toevoegen van een formulier
 - In Index.cshtml voeg je het formulier toe met twee attributen die verwijzen naar de controller en de action method. (taghelpers zijn attributen die beginnen met asp-)

```
<form asp-controller="Home" asp-action="Play" method="post">
    <input type="submit" value="Play" />
</form>
```

- Voeg ook een submit knop toe.
 - Zelf code ingeven

3. SnakeEyes : Controller

- ▶ Afhandelen van submitten van de form

- Formulier wordt verstuurd met http post

```
<form asp-controller="Home" asp-action="Play" method="post">
    <input type="submit" value="Play" />
</form>
```

- De actie die request ontvangt : **Play** in de **HomeController**
 - De action method Play in de HomeController
 - Moet de business operatie **Play** aanroepen van SnakeEyes
 - Kiest dan de **View** die gerenderd moet worden (mag hier ook de Index zijn) en geeft het **Model** door.

3. SnakeEyes : Controller

- ▶ Pas controller aan. Voeg Action method Play toe.

```
public IActionResult Play()
{
    _snakeEyes.Play();
    return View("Index", _snakeEyes);
}
```

3. SnakeEyes : Controller

- ▶ Afhandelen van submitten van de form
 - Run de applicatie

```
private SnakeEyes _snakeEyes;
0 references | Stefaan De Cock, Less t
public IActionResult Index()
{
    _snakeEyes = new SnakeEyes();
    return View(_snakeEyes);
}

0 references | 0 changes | 0 authors, 0
public IActionResult Play()
{
    _snakeEyes.Play(); ✖
    return View("Index", _snakeEyes);
}
```

Exception User-Unhandled

System.NullReferenceException: 'Object reference not set to an instance of an object.'

_snakeEyes was null.

[View Details](#) | [Copy Details](#)

► [Exception Settings](#)

_snakeEyes bestaat niet meer???

3. SnakeEyes : Controller

▶ State bijhouden

- Reden fout : HTTP is een **stateless protocol**
 - De toestand van een object wordt NIET bijgehouden tussen opeenvolgende requests. Er wordt telkens een nieuw object aangemaakt.
 - Oplossing : Session variabelen
 - **Session**
 - Resource op de server voor 1 gebruiker. Hierin kan je gegevens nodig voor die gebruiker tijdens 1 sessie (= 1 visit van de gebruiker aan de site) bijhouden => DUS **iedere bezoeker heeft zijn eigen sessie object**.
 - Die gegevens zijn toegankelijk vanuit alle pagina's binnen die webapplicatie en bevatten de info van die specifieke bezoeker
 - Hoe weet server nu welk sessie object tot welke bezoeker behoort? Bij het eerste verzoek van 1 gebruiker, maakt de server 1 sessie object aan met een uniek gegenereerd userID. De server stuurt een cookie met de userID naar de browser. Telkens de bezoeker dan een pagina opvraagt binnen de webapplicatie wordt cookie meegestuurd.

3. SnakeEyes : Controller

- ▶ <https://docs.asp.net/en/latest/fundamentals/app-state.html>

Session

Session storage relies on a cookie-based identifier to access data related to a given browser session (a series of requests from a particular browser and machine). You can't necessarily assume that a session is restricted to a single user, so be careful what kind of information you store in Session. It is a good place to store application state that is specific to a particular session but which doesn't need to be persisted permanently (or which can be reproduced as needed from a persistent store). See [Installing and Configuring Session](#), below for more details.

3. SnakeEyes : Controller

- ▶ Configureer de applicatie voor het gebruik van sessions: voeg in StartUp.cs
 - in de ConfigureServices

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddSession();
    services.AddControllersWithViews();
}
```

- en in de Configure **voor** app.UseEndpoints , app.UseSession()

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

- Opm : .Net Core 3.0 bevat een System.Net.Json namespace, maar aanduiden van fields en constructor ontbreekt. Vandaar maken we gebruik van JSON.Net

3. SnakeEyes : Controller

- ▶ State bijhouden door sessie gegevens te lezen en te schrijven. Aangezien we met objecten werken moet het object geserializeerd. Dit gebeurt door serializeren/deserializeren naar json.
 - Sessie gegevens lezen en schrijven
 - Schrijven
 - Lezen

```
HttpContext.Session.SetString("SnakeEyes", JsonConvert.SerializeObject(_snakeEyes));
```

```
• Lezen
```

```
_snakeEyes = JsonConvert.DeserializeObject<SnakeEyes>(HttpContext.Session.GetString("SnakeEyes"));
```

- Het session object wordt vernietigd ivm
 - Bij time out van de sessie : default 20 min na laatste

3. SnakeEyes : Controller

- ▶ We maken Extension methods aan om Session objecten aan te maken en te lezen.
- ▶ Maak map Extensions aan in Project

```
using Microsoft.AspNetCore.Http;
using Newtonsoft.Json;

namespace SnakeEyesGame.Extensions
{
    public static class SessionExtensions
    {
        public static void SetObject<T>(this ISession session, string key, T value)
        {
            session.SetString(key, JsonConvert.SerializeObject(value));
        }

        public static T GetObject<T>(this ISession session, string key)
        {
            var value = session.GetString(key);
            return value == null ? default(T) :
                JsonConvert.DeserializeObject<T>(value);
        }
    }
}
```

3. SnakeEyes : Controller

- ▶ State bijhouden
 - De code in de controller wordt

```
public IActionResult Index()
{
    _snakeEyes = new SnakeEyes();
    HttpContext.Session.SetObject("SnakeEyes", _snakeEyes);
    return View(_snakeEyes);
}
```

0 references | 0 changes | 0 authors, 0 changes | ✓ 4 requests | 0 exceptions

```
public IActionResult Play()
{
    _snakeEyes = HttpContext.Session.GetObject<SnakeEyes>("SnakeEyes");
    _snakeEyes.Play();
    HttpContext.Session.SetObject("SnakeEyes", _snakeEyes);
    return View("Index", _snakeEyes);
}
```

3. SnakeEyes : Controller

- ▶ Objecten kunnen niet zo maar geserializeerd worden. Daarom moeten de fields en properties aangeduid worden die mogen geserializeerd worden. Dit gebeurt door [JsonProperty] te plaatsen boven de fields/members die mogen geserializeerd worden.
- ▶ Bovenaan de klasse plaats je
[JsonObject(MemberSerialization.OptIn)]
- ▶ Meer informatie:
<http://www.newtonsoft.com/json/help/html/SerializationGuide.htm>
- ▶ Opm. de default is MemberSerialization.OptOut : alle public members worden dan geserializeerd

3. SnakeEyes : Controller

▶ Code voor Dice en SnakeEyes

```
[JsonObject(MemberSerialization.OptIn)]
5 references | stefaandc, 13 hours ago | 2 authors, 4 changes
public class Dice
{
    #region Fields
    private static Random _random = new Random();
    #endregion

    #region Properties
    [JsonProperty]
    8 references | Stefaan Samyn, 1 day ago | 2 authors, 2 changes
    public int Pips { get; private set; }
}
```

```
[JsonObject(MemberSerialization.OptIn)]
4 references | stefaandc, 13 hours ago | 2 authors, 5 changes
public class SnakeEyes
{
    #region Fields
    [JsonProperty]
    private Dice _eye1;
    [JsonProperty]
    private Dice _eye2;
    #endregion

    #region Properties
    [JsonProperty]
    3 references | stefaandc, 13 hours ago | 2 authors, 4 changes
    public int Total {get; private set; }
```

3. SnakeEyes : Controller

▶ Code voor Dice en SnakeEyes

```
[JsonObject(MemberSerialization.OptIn)]
5 references | stefaandc, 13 hours ago | 2 authors, 4 changes
public class Dice
{
    #region Fields
    private static Random _random = new Random();
    #endregion

    #region Properties
    [JsonProperty]
    8 references | Stefaan Samyn, 1 day ago | 2 authors, 2 changes
    public int Pips { get; private set; }
}
```

```
[JsonObject(MemberSerialization.OptIn)]
4 references | stefaandc, 13 hours ago | 2 authors, 5 changes
public class SnakeEyes
{
    #region Fields
    [JsonProperty]
    private Dice _eye1;
    [JsonProperty]
    private Dice _eye2;
    #endregion

    #region Properties
    [JsonProperty]
    3 references | stefaandc, 13 hours ago | 2 authors, 4 changes
    public int Total {get; private set; }
```

4. MVC Flow

▶ MVC Flow

- Step 1
 - Binnenkomende request wordt verstuurd naar Controller. Daar wordt de bijhorende actie uitgevoerd.
 - De controller en actie wordt bepaald door de routing

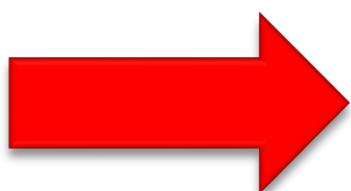


Request

4. MVC Flow

▶ MVC Flow : Step 2

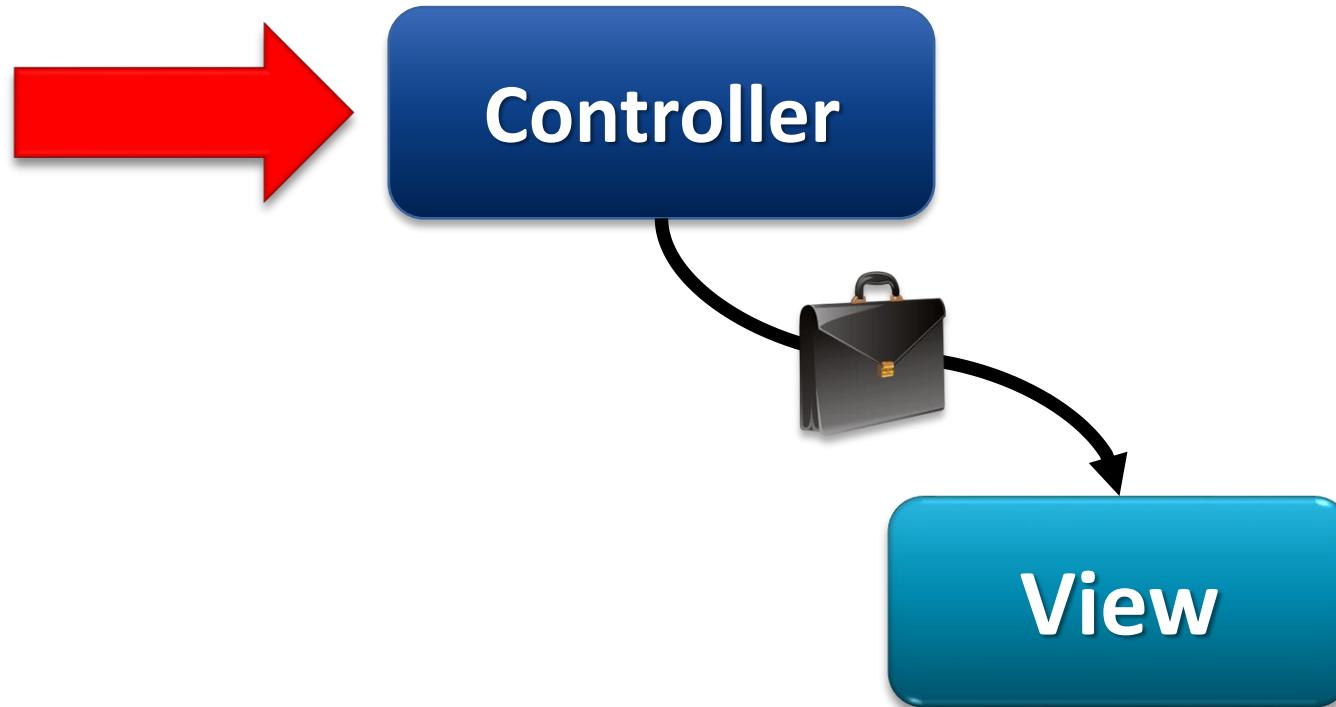
- **Controller** verwerkt request, communiceert met Domein en maakt **Model** (de gegevens die gerenderd moeten worden. Kan een ViewBag(ViewData) of domein of ViewModel object zijn)



Model

4. MVC Flow

- ▶ MVC Flow : Step 3
 - ViewBag en Model wordt doorgegeven aan de View



4. MVC Flow

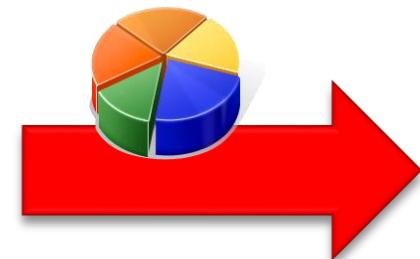
- ▶ MVC Flow : Step 4
 - View transformeert Model naar juiste output formaat



4. MVC Flow

▶ MVC Flow : Step 5

- Response is aangemaakt en wordt naar de browser verstuurd als antwoord op de request



Response

5. Oefening

- ▶ SnakeEyes : Voeg de opmerking “Oeps you did it again!” toe als 2 * 1 gegooid werd.
 - Pas klasse SnakeEyes aan. Voorzie property HasSnakeEyes. Pas methode Play aan zodat deze hier gebruik van maakt.
 - Pas css aan. Voeg class comment toe
 - font-size : x-large; color:red
 - Pas View aan



```
@if (Model.HasSnakeEyes)  
{  
    <p class="comment">Oeps, you did it again!</p>  
}
```

5. Oefening

- ▶ SnakeEyes :
 - Hou de hoogste score bij en geef dit ook weer in de View