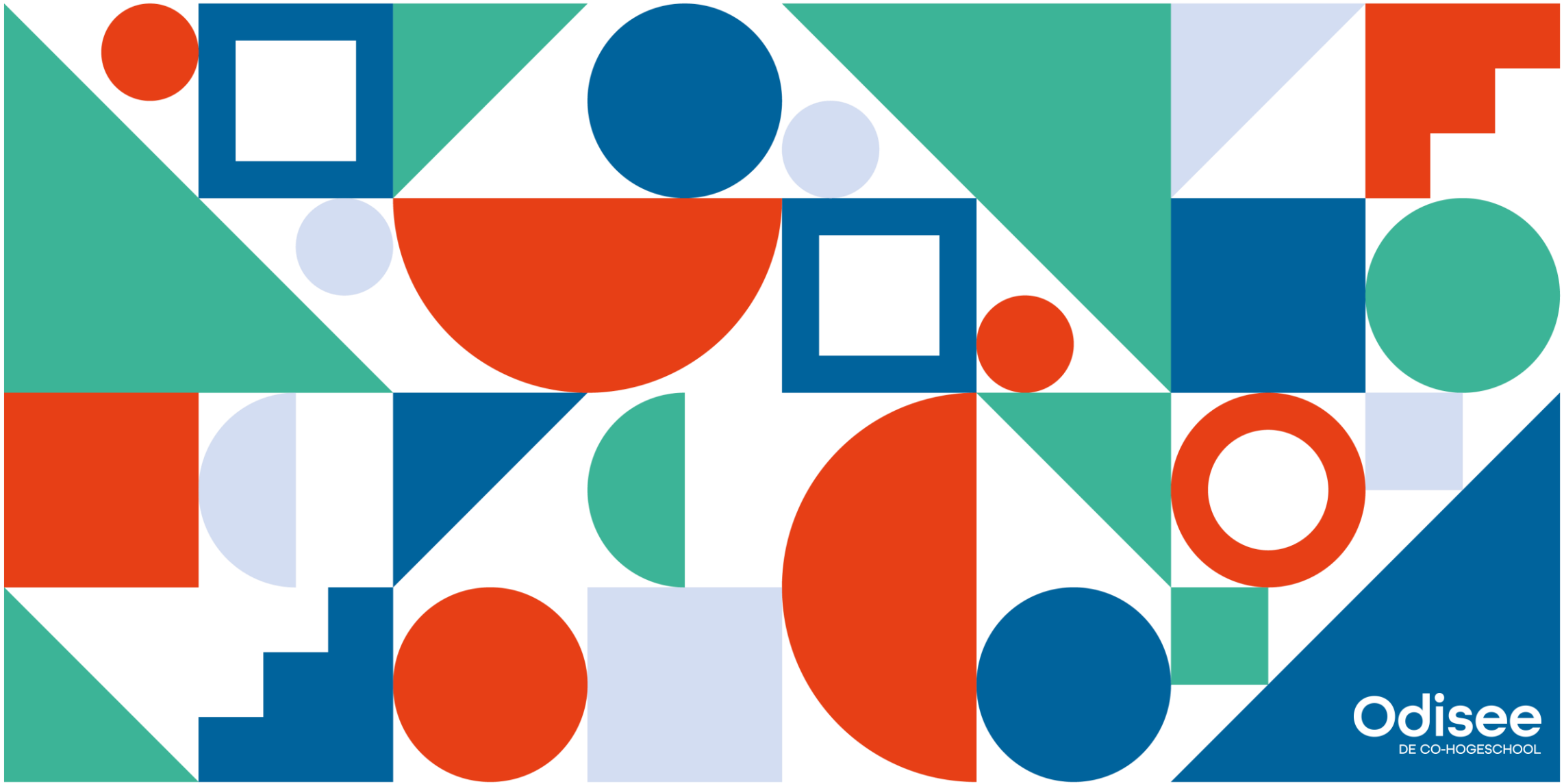

Odisee
DE CO-HOGESCHOOL

# Application Development

## Inleiding MVC

**Sam Van Buggenhout**

Academiejaar 2023-2024



Odisee
DE CO-HOGESCHOOL

# MVC-design pattern

# Het MVC-design pattern

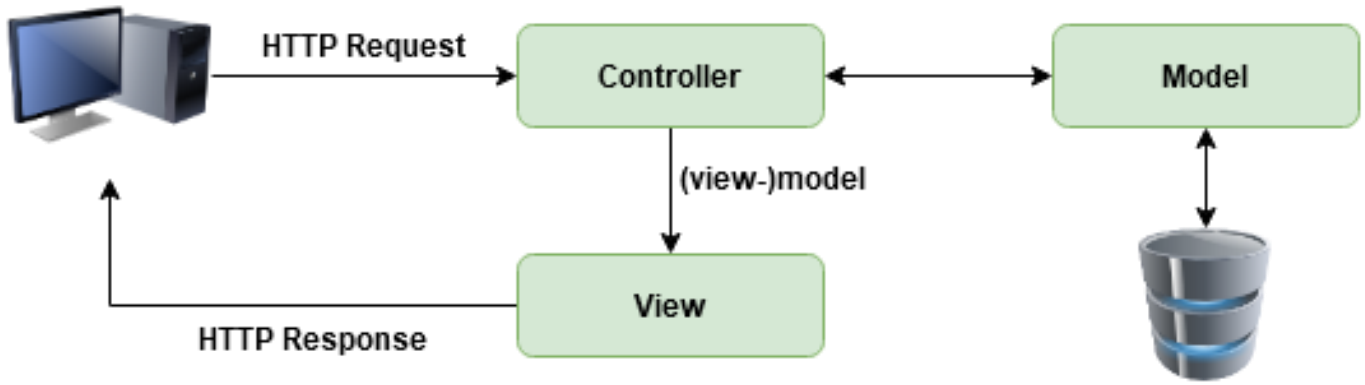
- MVC is een veelgebruikt design pattern voor de ontwikkeling van applicaties met een GUI
- Oorspronkelijk voor rich-client toepassingen (desktop-applicaties)
- Nadien ook nut bewezen voor andere toepassingen (web-toepassingen en mobiele applicaties)
- Doel: duidelijke afscheiding tussen manipulatie/verwerking van data en de visuele weergave ervan

# Het MVC-design pattern

- Bij **MVC** wordt de applicatie **opgedeeld** in **drie componenten:**
  - **Model:** de data van de applicatie (ophalen/persisteren van data), business rules, validatie, ...
  - **View:** weergave (representatie) van model (HTML, WPF, JSON, XML, ...)
  - **Controller:** coördinator tussen model en view. Update het model en geeft data door aan de juiste view.
- Strikte **scheiding** tussen verschillende onderdelen
- Vaak ook gewerkt met **ViewModel**: model dat specifieke property's bevat die in View weergegeven moeten worden

# Het MVC-design pattern

- MVC-design pattern in .NET 6 MVC:

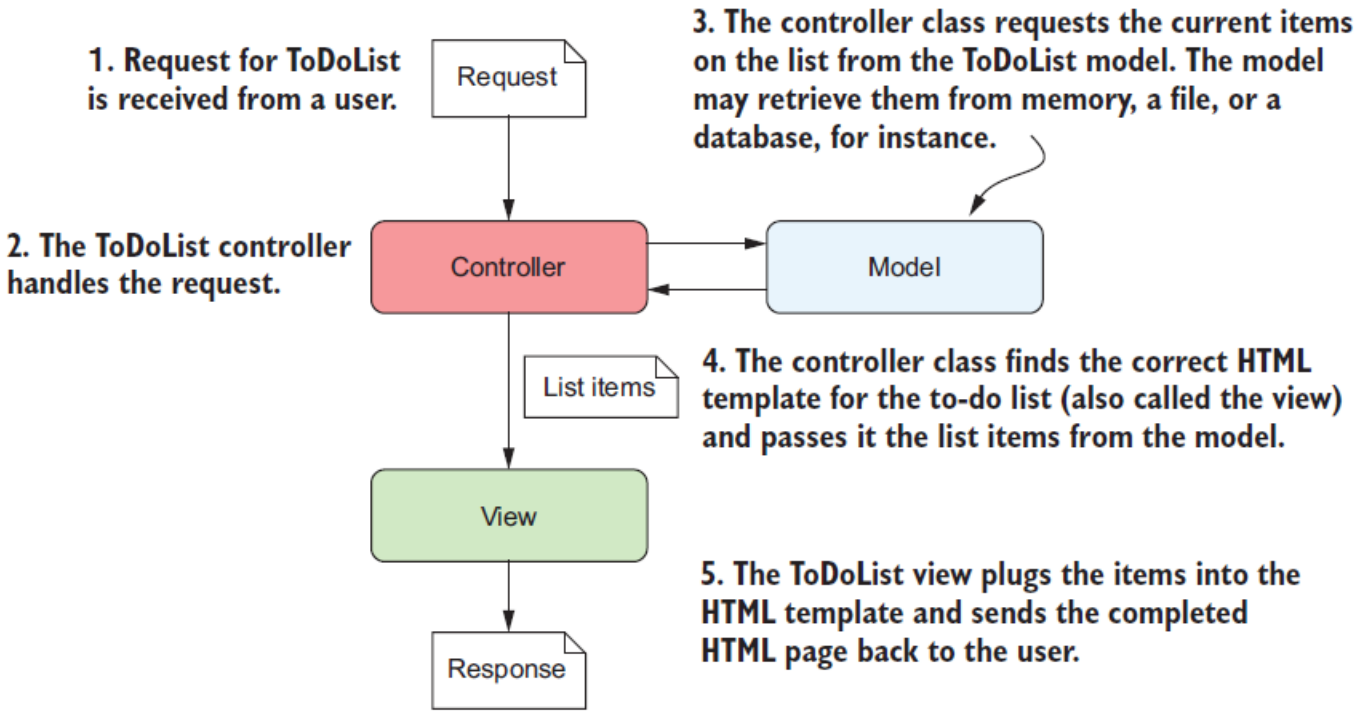


# Het MVC-design pattern

1. De controller ontvangt de request
2. Afhankelijk van het type request, haalt de controller de nodige data op uit het model of update het model
3. Controller selecteert een view en geeft het model mee aan deze view om dit weer te geven
4. De view gebruikt de data in het model om de GUI (bv.: HTML) te genereren

# Het MVC-design pattern

- Voorbeeld: opvragen TODO-list:




*Bron: Lock, A. (2018). Asp.net Core in Action. Shelter Island, NY: Manning Publications.*

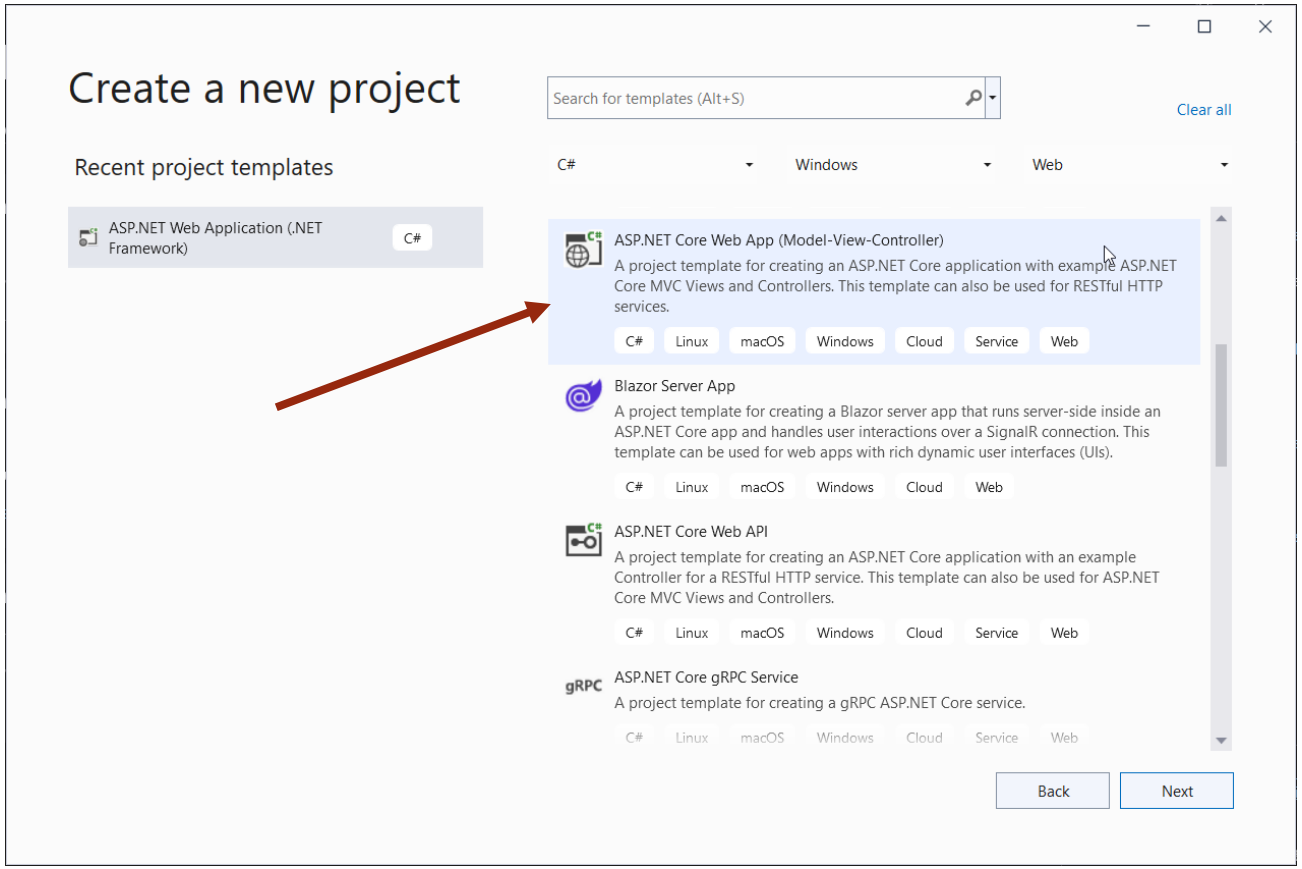
# Het MVC-design pattern: voordelen

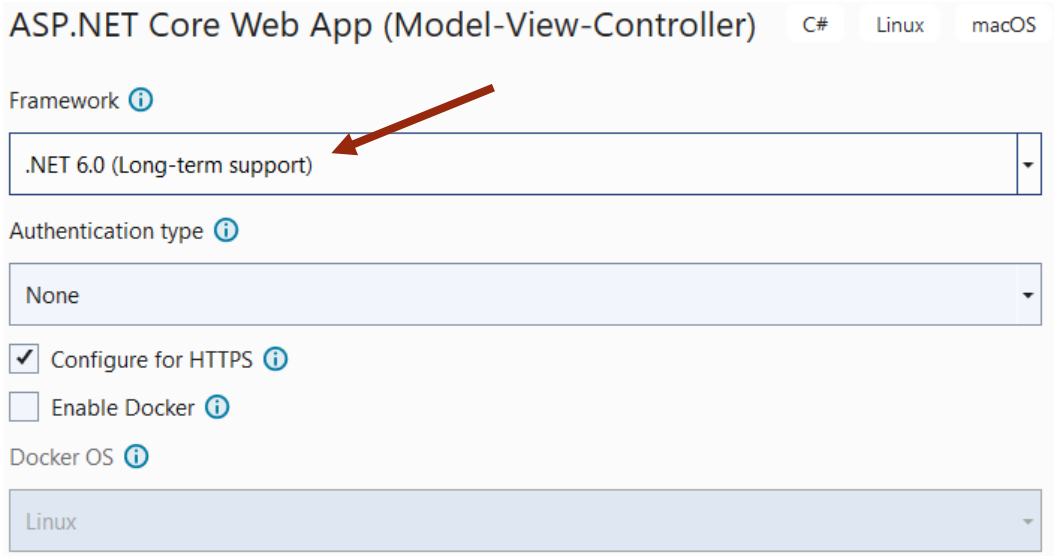
- Voordelen:
  - 👍 Betere SoC (separation of concerns): designers werken aan views, developers aan controllers
  - 👍 Componenten kunnen **afzonderlijk gewijzigd** worden
  - 👍 **Meerdere views** (HTML, JSON, mobiele app, ...) op eenzelfde applicatie mogelijk
  - 👍 **Geen dependencies** tussen GUI en applicatie-logica
    → Verhoogt **testbaarheid**

# Een eerste project

# Een nieuw project op basis van een template

## Type project: ASP.NET Core Web App (Model-View-Controller)

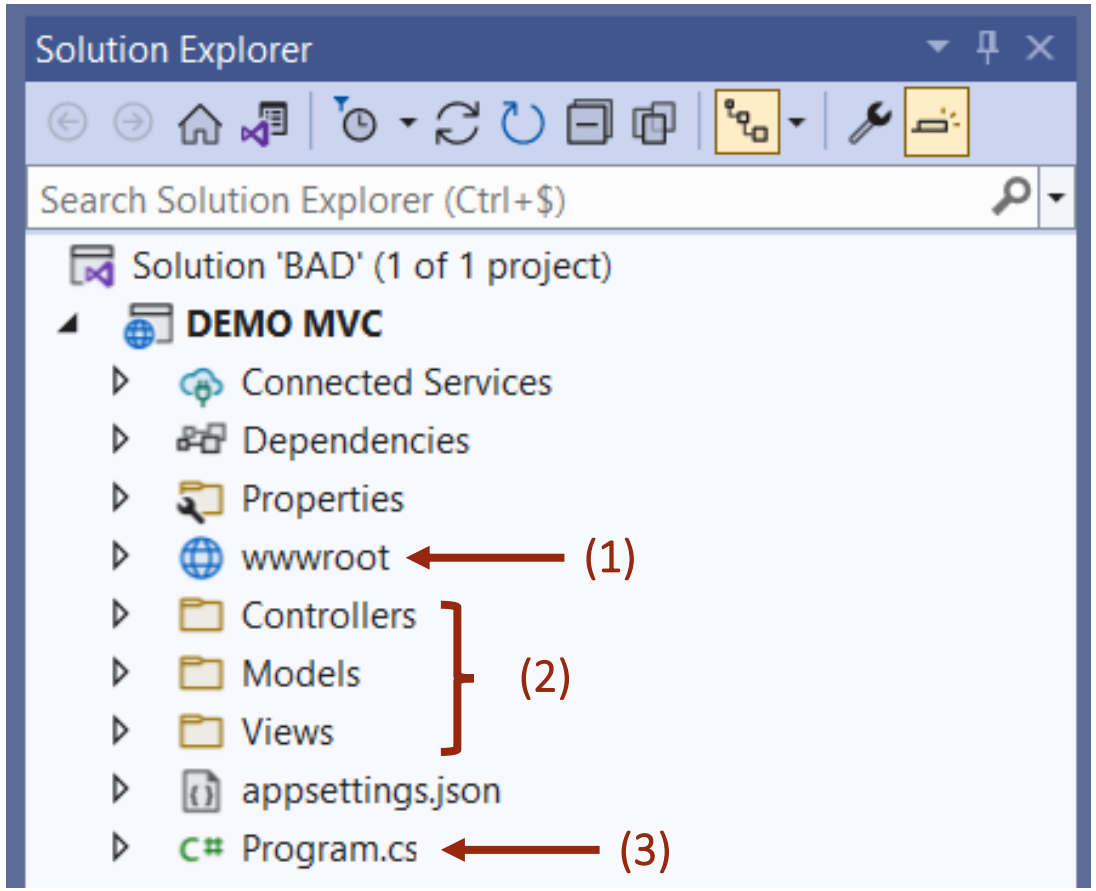





Odisee
DE CO-HOGESCHOOL

# Projectstructuur




1. wwwroot:
   - enige folder waar **browsers onmiddellijk** toegang tot hebben
   - Gebruikt voor **statische** files (CSS, JavaScript, afbeeldingen, ...)
2. MVC-folders:
   - Bevatten **applicatie-code** (ingedeeld in Models-, Views- en Controllers-folders)
   - Niet verplicht, maar **conventie**
3. Program.cs:
   - **opstart** en **configuratie** van de web-applicatie



Odisee
DE CO-HOGESCHOOL

# Projectstructuur: Program.cs

```
var builder = WebApplication.CreateBuilder(args);
// Add services to the container.
builder.Services.AddControllersWithViews();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

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

app.Run();
```

dependency's

middleware

- Bevat de configuratie van de applicatie:
  - Configuratie WebHost (server)
  - Dependency's (services)
  - Middleware

# ASP.NET 6: Middleware

- **Middleware** = klassen die HTTP-requests/responses afhandelen:
  - HTTP-request afhandelen door **HTTP-response** te **genereren**
  - HTTP-request **aanpassen** en **doorsturen** naar volgende middleware-component
  - **HTTP-response bewerken** en doorsturen naar **volgende middleware-component** óf naar **webserver**

# ASP.NET 6: Middleware

- Voorbeelden:
  - Logging-middleware (gegevens van request loggen, bv.: wegschrijven in log-bestand)
  - Image-resizing middleware: aanvragen voor afbeeldingen afhandelen en resultaat terugsturen
  - Requests voor statische bestanden afhandelen
  - Foutafhandeling (tonen van gebruiksvriendelijke foutboodschap voor Exceptions)
- Elke component heeft (typisch) één verantwoordelijkheid

# ASP.NET 6: Middleware

- Verschillende **middleware-componenten** kunnen **“gechained”** worden tot een **pipeline**
- Elke component bewerkt request/response en stuurt deze door naar volgende middleware-component in de pipeline of de webserver
- Volgorde waarin middleware-componenten toegevoegd worden aan pipeline is van belang!

# ASP.NET 6: Pipelines

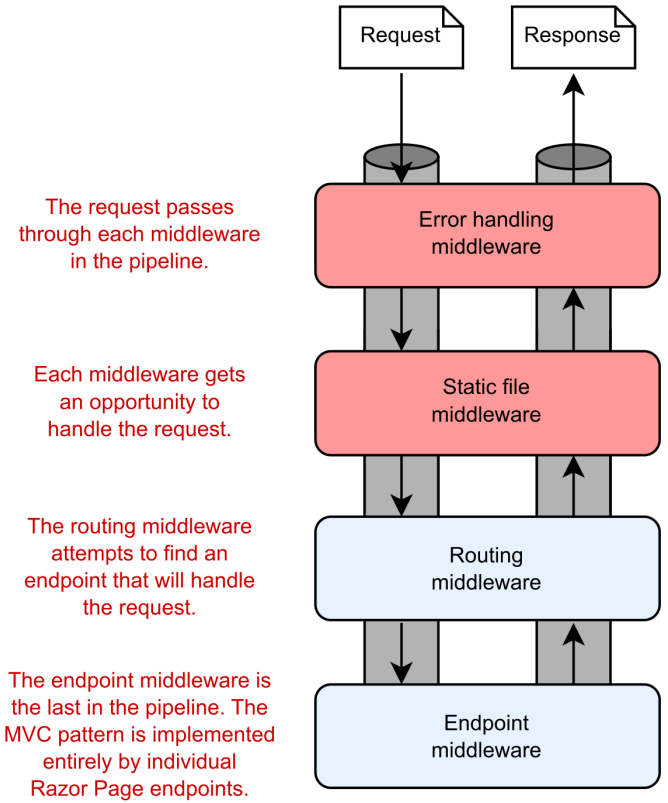
```
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

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

Error-handling middleware

Static file middleware

Routing middleware

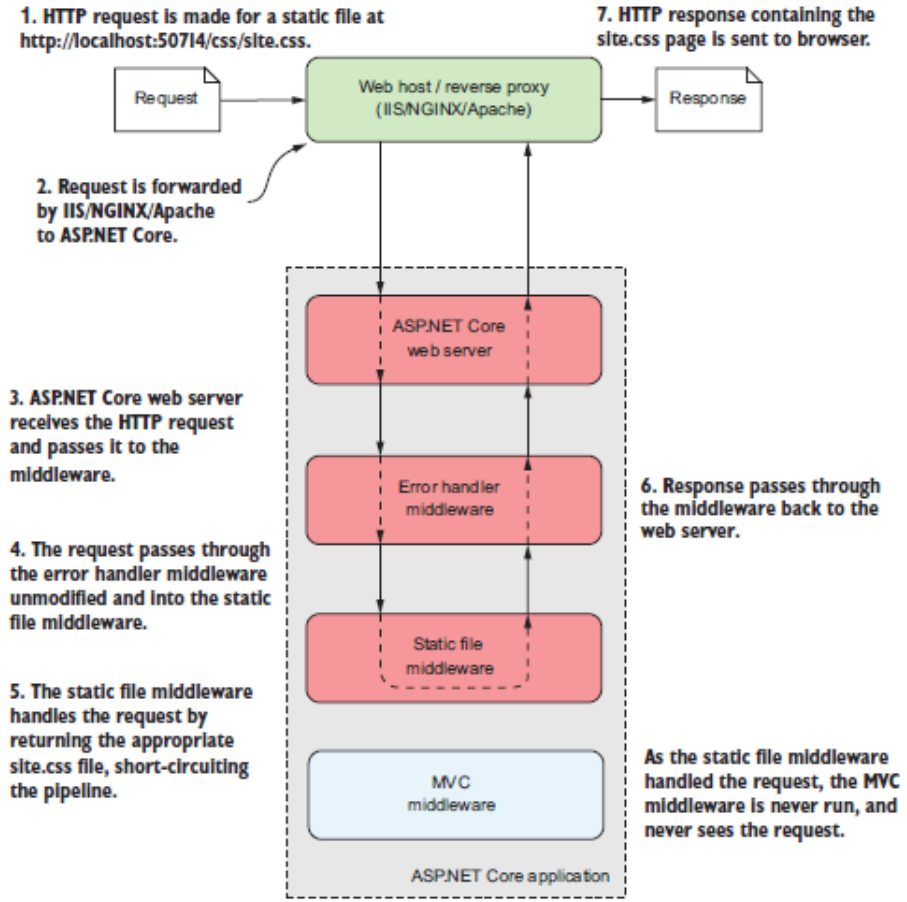
Endpoint middleware




Bron: Lock, A. (2021). *ASP.NET Core in Action, Second Edition*. Shelter Island, NY: Manning Publications.

# ASP.NET 5: Pipelines

- Voorbeeld afhandelen request statische file via pipeline:




Bron: Lock, A. (2021). *ASP.NET Core in Action, Second Edition*. Shelter Island, NY: Manning Publications.

# MVC: Controllers

# MVC: Controller

```
public class HomeController : Controller {
    private readonly ILogger<HomeController> _logger;

    public HomeController(ILogger<HomeController> logger) {
        _logger = logger;
    }

    public IActionResult Index() {
        return View();
    }

    public IActionResult Privacy() {
        return View();
    }

    [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None,
        NoStore = true)]
    public IActionResult Error() {
        return View(new ErrorViewModel { RequestId = Activity.Current?.Id ??
                                         HttpContext.TraceIdentifier });
    }
}
```

Action

Action

Action

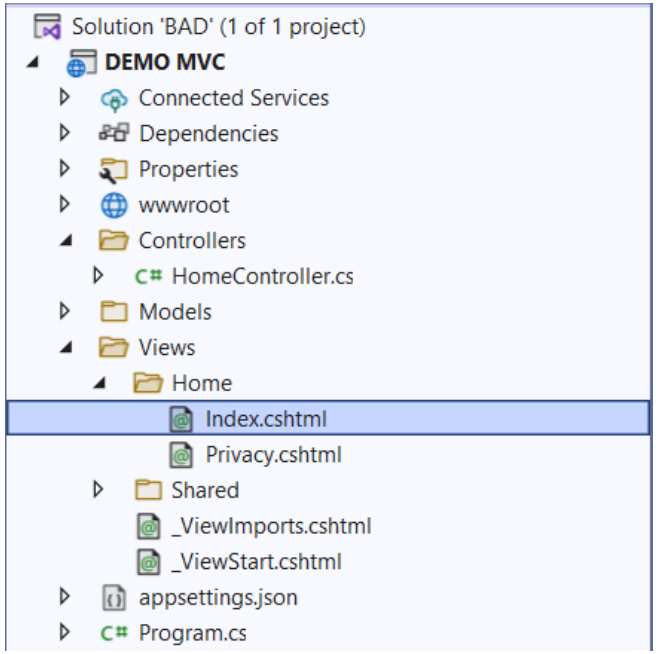
- Bij conventie: in folder *Controllers*
- Controller erft over van klasse **`Controller`**
- Bevat één of meerdere **Actions**

# MVC: Controller - Actions

- Action = methode die uitgevoerd wordt als response op een HTTP-request
- Controller bevat actions die “logisch bij elkaar horen”
- Elke Action-methode returnt een `IActionResult` → bevat informatie om HTTP-response te genereren (typisch een View = HTML-pagina)

# MVC: Controller - Actions

```
public class HomeController : Controller {

    public IActionResult Index() {
        return View();
    }

    ...

}
```




- **`View()`-methode**: selecteert view om HTML te genereren die teruggestuurd wordt als resultaat
  - Standaard locatie: /Views/*<Controller naam>*/*<Action naam>*
    - *Voorbeeld: Views/Home/Index*
  - Kan ook expliciet meegegeven worden als parameter:
    - `View("Views/Home/Index.cshtml");` → absoluut pad
    - `View("../Account/Login");` → view andere controller
    - `View("./About");` → andere view, zelfde controller



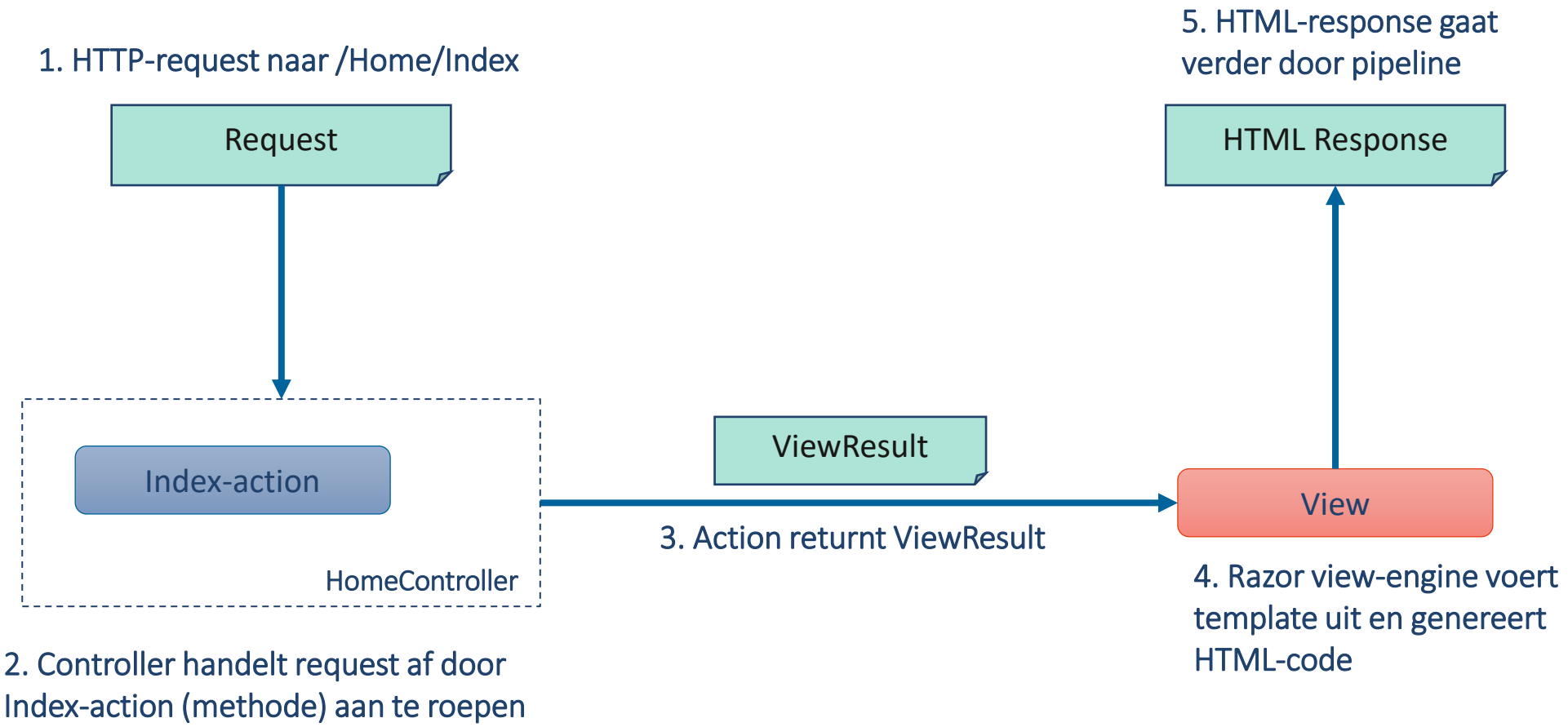
Odisee
DE CO-HOGESCHOOL

# MVC: Controller - Actions

```
public class HomeController : Controller {

    [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
    public IActionResult Error() {
        return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
    }

}
```

- Controller kan **data doorgeven** aan view (wordt in view weergegeven)
- **ViewModel**: overkoepelende klasse die alle gegevens bevat die nodig zijn om view te genereren (bv.: alle properties voor velden van zoekformulier, ...)
- Controller kan **data bv. ophalen** uit databank om door te geven aan view
- Controllers kunnen **data ontvangen** (bv.: van POST-request/GET-parameters) en verwerken

# MVC: Controller - Actions

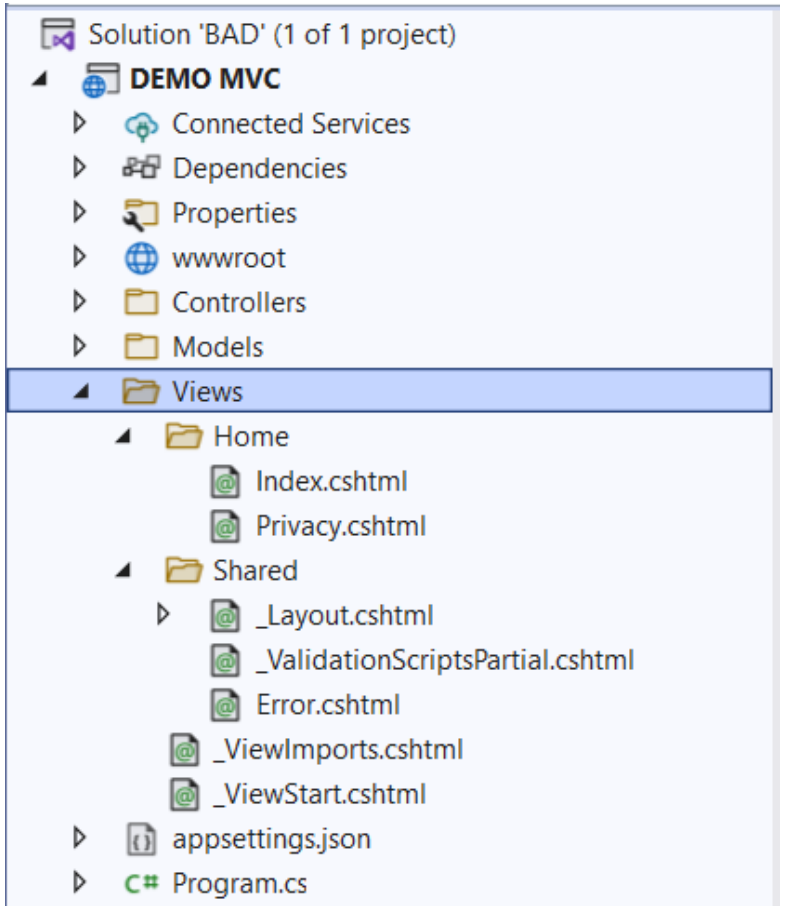





Odisee
DE CO-HOGESCHOOL

# MVC: Views

# MVC: Views




- Views hebben **zelfde naam** als de bijhorende **action**
- Views voor actions in ***HomeController***, zitten in ***Home***-folder
- Views in ***Shared***-folder kunnen door **alle controllers** gebruikt worden



Odisee
DE CO-HOGESCHOOL

# MVC: Views

- Views worden gebruikt om data van model **weer te geven** (bv.: lijst van studenten, zoekresultaten, ...)
- Worden ook gebruikt voor **interactie** met gebruiker (bv.: formulieren, links/knoppen, ...)
- HTML wordt **dynamisch** gegenereerd mbv een **view-engine** (**Razor**):
  - **Template**-gebaseerd
  - Pagina's bevatten combinatie van **HTML- en C#-code**
  - **Extensie: *.cshtml***

```
<h1>Home Page</h1>

<!-- Storing a string -->
@{ var message = "Welcome to our website"; }

<!-- Storing a date -->
@{ DateTime date = DateTime.Now; }

<p>@message</p>
<p> The current date is @date</p>
```

# MVC: Views

- _ViewStart.cshtml:
  - Wordt uitgevoerd **vóór** het genereren van een view
  - Instellen van **Layout-property** voor de pagina
  - Van toepassing op alle views in **zelfde folder** (**en sub-folders**)
- _ViewImports.cshtml:
  - Bevat using's die voor alle pagina's geïmporteerd moeten worden
  - Doel: deze using's niet in alle afzonderlijke pagina's importeren
  - Van toepassing op alle views in **zelfde folder** (**en sub-folders**)

# MVC: Views

- _ViewStart.cshtml:
  - Wordt uitgevoerd **vóór** het genereren van een view
  - Instellen van **Layout-property** voor de pagina
  - Van toepassing op alle views in **zelfde folder** (**en sub-folders**)
- _ViewImports.cshtml:
  - Bevat using's die voor alle pagina's geïmporteerd moeten worden
  - Doel: deze using's niet in alle afzonderlijke pagina's importeren
  - Van toepassing op alle views in **zelfde folder** (**en sub-folders**)

# MVC: Views

- _Layout.cshtml:
  - Template voor verschillende content-pages
  - Doel: gemeenschappelijke lay-out van verschillende pagina's vastleggen
  - Bevat "placeholders" die door specifieke pagina's ingevuld worden
  - Analoog aan Master-page in ASP.NET WebForms

# MVC: Views

/Views/Shared/Layout.cshtml:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title></title>
    <link href="/css/site.css" rel="stylesheet" type="text/css" />
</head>
<body>
    @RenderBody()
</body>
</html>
```

/Views/Home/Index.cshtml:

```
@{
    Layout = "_Layout";
}

<h1>Welcome</h1>
<p>This is our home page.</p>
```

/Views/Home/About.cshtml:

```
@{
    Layout = "_Layout";
}

<h1>About Us</h1>
<p>This is our about page.</p>
```

In plaats van **`@{Layout= "_Layout";}`** toe te voegen aan elke pagina afzonderlijk → centraliseren in *_ViewStart.cshtml*

# MVC: Model

# MVC: Model

- Bevat **data** die in de applicatie moet weergegeven worden (in de view), bevat de **business logica**
- **Application Model** bevat typisch:
  - Domain Model (objecten die in database bewaard worden **= Entities**)
  - Database-interactie code (repository's)
  - Services
- Voor **response** wordt vaak **View Model** aangemaakt
  - Bevat **alle data** die nodig is om **view** te **genereren**
  - **Controller** maakt **View Model** en **geeft** dit **door** aan view om te renderen

# MVC: Model

- Wordt typisch geïmplementeerd als POCO (= plain old C#-object)
- Bij conventie in folder *Models*
- **Voorbeeld:** samenhang tussen Controller, Model, ViewModel en View

Application Model (entities):

```
public class Student {
    public int ID { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
}

public class Course {
    public int ID { get; set; }
    public string Name { get; set; }
}
```

DB-access (Repository):

```
public class StudentRepository {
    public List<Student> GetAll() {
        //...
    }
}

public class CourseRepository {
    public List<Course> GetAll() {
        //...
    }
}
```

# MVC: Model

## **Voorbeeld (vervolg):** samenhang tussen Controller, Model, ViewModel en View

### View Model:

```
public class RegistrationViewModel {
    public List<Student> Students { get; set; }
    public List<Course> Courses { get; set; }
}
```

### Controller:

```
public class StudentsController : Controller {
    private StudentRepository _studentsRepo = new StudentRepository();
    private CourseRepository _courseRepo = new CourseRepository();

    public IActionResult Register() {
        RegistrationViewModel viewModel = new RegistrationViewModel();
        viewModel.Students = _studentsRepo.GetAll();
        viewModel.Courses = _courseRepo.GetAll();

        return View(viewModel);
    }
}
```

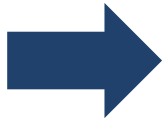
1. Controller haalt data op die getoond moet worden in view
2. Controller maakt ViewModel aan dat alle data bevat die in view getoond wordt (ViewModel “groepeert” deze data in één object)
3. Controller geeft dit object door aan view om deze data weer te geven (bv.: door HTML-pagina te genereren/als JSON/...)

# Razor: basis-syntax

# Razor: basis-syntax

## Inline-expressie: begin de expressie met een @-karakter

```
<h1>Example inline expression</h1>
<h2>@DateTime.Now.ToShortDateString()</h2>
```

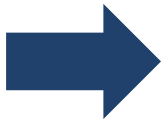


Example inline expression

10/01/2021

## Code-blokken: meerdere regels code plaats je tussen@{...}

```
@{
    var date = DateTime.Now;
    var message = "Hello, World!";
}
<h1>Example Code Block</h1>
<p>Today's date is: @date.ToShortDateString()</p>
<p>@message</p>
```



Example Code Block

Today's date is: 10/01/2021

# Razor: basis-syntax

Om tekst te tonen vanuit een **code-block**, zet je deze tekst tussen **<text>**-tags, of begin je de regel met **@:**

```
@{
    int a = 5;
    int b = 10;
    int sum = a + b;

    <text>The sum of </text> @a <text> and </text> @b <text> is </text> @sum
    <br />
}
```

=

```
@{
    int a = 5;
    int b = 10;
    int sum = a + b;

    @: The sum of @a and @b is @sum
    <br />
}
```

The sum of 5 and 10 is 15

# Razor: basis-syntax

**Condities:** begin **if-statement** met **@-karakter**.
Accolades zijn verplicht! (ook indien slechts één statement)

```
@if (DateTime.IsLeapYear(DateTime.Now.Year))
{
    <p>The year @DateTime.Now.Year is a leap year.</p>
}
else
{
    <p>The year @DateTime.Now.Year is <b>NOT</b> a leap year.</p>
}
```

The year 2021 is **NOT** a leap year.

# Razor: basis-syntax

**Variabelen:** het is mogelijk om een **variabele** te declareren en deze op een **later punt** in de code te gebruiken

```
@{ var greeting = "";}

@if(DateTime.Now.Hour < 10)
{
    greeting = "Good morning!";
}else if(DateTime.Now.Hour < 20)
{
    greeting = "Good day!";
}
else
{
    greeting = "Good evening!";
}

<p>@greeting And welcome to my web page!</p>
```

declaratie

toekenning

waarde opvragen

Good evening! And welcome to my web page!

# Razor: basis-syntax

## Iteraties (for): begin **for-statement** met een @-karakter

```
<ul>
    @for (int i = 1; i <= 5; i++)
    {
        <li>item @i.ToString()</li>
    }
</ul>
```

- item 1
- item 2
- item 3
- item 4
- item 5

***Opmerking:*** *for-iteratie kan tussen <ul>-tags (of andere tags) staan*

# Razor: basis-syntax

**Iteraties (foreach):** op een gelijkaardige manier kan je tevens gebruik maken van een **foreach-iteratie**

```
@{ string[] names = {"Jan", "Piet", "Joris", "Corneel"};}

<ul>
    @foreach (var name in names)
    {
        <li>@name</li>
    }
</ul>
```

- Jan
- Piet
- Joris
- Corneel

# Razor: basis-syntax

**Objecten en property's:** je kunt gebruik maken van **inline-expressies** om de waarde van een property van een object op te vragen en weer te geven

```
@{
    Person p = new Person("Joske", "Vermeulen", 21);
}

<p>Hello, my name is @p.FirstName @p.LastName. I am @p.Age years old.</p>
```

Hello, my name is Joske Vermeulen. I am 21 years old.