

**Odisee**  
DE CO-HOGESCHOOL

# Application Development

Views (basics)



**Sam Van Buggenhout**

Academiejaar 2023-2024

**Routing**

**Controllers & Views**

**View selectie**

**Data doorgeven aan View**

**Razor**



# Routing

## Parameters doorgeven aan Actions

- Soms heeft Action-method **parameter(s)** nodig om pagina te kunnen genereren
  - **Voorbeeld:** pagina om de details van een product op te vragen  
→ Action-method moet weten wélk product moet opgehaald en weergegeven worden
  - **Hoe?** Id van het product meegeven als **parameter** aan Action-methode

# Parameters doorgeven aan Actions

- Voorbeeld:


```
public class ProductController : Controller {  
    private ProductRepository _productRepo = new ProductRepository();  
  
    public ActionResult Details(int id) {  
        //haal product op uit repo (op basis van id-parameter)  
        Product product = _productRepo.FindById(id);  
  
        //geef product mee aan View om weer te geven  
        return View(product);  
    }  
}
```

- Hoe parameter meegeven aan Action?
  - Via URL: *https://localhost:44374/Product/Details/5*

## Parameters doorgeven aan Actions

- Hoe wordt URL naar juiste controller, action en parameter geleid?

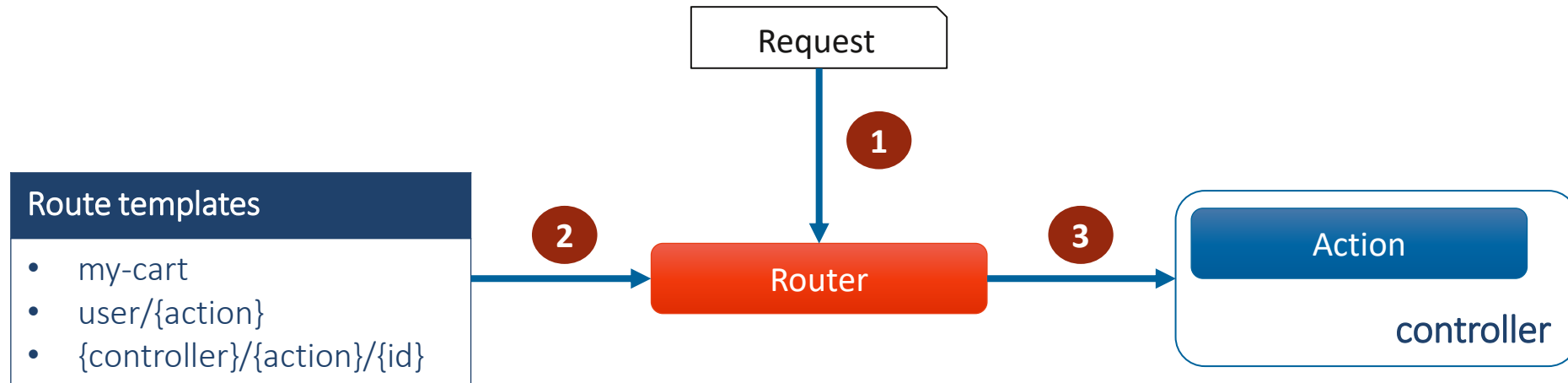
*https://localhost:44374/Product/Details/5*



The diagram illustrates the routing of the URL *https://localhost:44374/Product/Details/5*. Green brackets are placed under the path segments: 'Product' is bracketed and labeled 'controller', 'Details' is bracketed and labeled 'action', and '5' is bracketed and labeled 'id'.

- Routing** = proces waarbij inkomende HTTP-request gemapt wordt naar een specifieke **handler** (*in MVC: handler = **Action-method***)

# Hoe werkt routing?



1. URL van HTTP-request wordt naar router gestuurd
2. Router zoekt in lijst met geconfigureerde route-templates naar eerste template die “matcht” met de URL van de request
3. De router bepaalt op basis van deze template de controller en de action om deze request af te handelen




## Waarom routing?

- Doel: URL **loskoppelen** van handler
- Vroeger: request naar **specifiek bestand** (bv.: *home.aspx*):
  - *Voordeel: eenvoudig te begrijpen*
  - *Nadeel: moeilijk onderhoudbaar (wat als filenaam wijzigt?)*

## Hoe routing configureren?

- Routing configureren kan op twee manieren:
  - **Globaal (= conventional routing):** “afspraken” vastleggen waaraan structuur van URL moet voldoen. Geldt voor volledige applicatie
  - **Via attributen:** per controller/action-method aangeven via een **[Route]**-attribuut
- Verwachte URL's gespecificeerd via **route templates**
  - Bepalen **structuur** van URL's
  - String's met **placeholders** voor **variabelen**, kunnen **optionele waarden** bevatten en mappen naar **controllers** en **action's**

## Hoe routing configureren?

- URL wordt verdeeld in **segmenten**
  - Aaneensluitend deel van URL, gescheiden van andere segmenten door minimum één karakter (meestal '/')
  - Voorbeeld: *Students/Edit/1*  


*3 segmenten*

- URL wordt verdeeld in **segmenten**
- Via **template**: **vaste** strings (literals), **variabele** segmenten, **optionele** segmenten, **default-waarden** voor variabele segmenten en bijkomende **constraints** definiëren



# Routing: templates

# Routing templates

- Voorbeeld routing template: `api/{controller}/{action}`



- **Literal:** segment moet letterlijk in URL voorkomen
- **Route parameters:** kunnen variabel zijn (verschillend van URL tot URL).  
Worden steeds tussen accolades gezet

# Routing templates

- Op basis van route parameters worden **controller** en **action** bepaald
- Voorbeeld:
  - Route template: `{controller}/{action}`
  - URL: `students/overview`
    - Gerouteerd naar: `StudentsController.Overview()`
  - **MAAR:** indien URL “students” → *geen match! (action kan niet bepaald worden)*

## Routing templates

- Voorbeeld (met literal): `/api/{controller}/{action}`

| URL                 | Match? | Controller         | Action |
|---------------------|--------|--------------------|--------|
| /api/products/list  | ✓      | ProductsController | List() |
| /api/orders/all     | ✓      | OrdersController   | All()  |
| /reservations/index | ✗      |                    |        |
| /api/recipes        | ✗      |                    |        |

→ Route parameters `{controller}` en `{action}` moeten verplicht aanwezig zijn in de URL!

# Routing templates

- Via deze weg ook mogelijk om **bijkomende parameters** aan Action-methode mee te geven
- Voorbeeld: {controller}/{action}/{**id**}
- URL: /Product/Details/**3**
- Gerouteerd: Controller=ProductController, Action=Details, **id=3**

```
public class ProductController : Controller {  
    private ProductRepository _productRepo = new ProductRepository();  
  
    public ActionResult Details(int id) {  
        //haal product op uit repo (op basis van id-parameter)  
        Product product = _productRepo.FindById(id);  
  
        //geef product mee aan View om weer te geven  
        return View(product);  
    }  
}
```

## Opgelet:

- naam parameter in template moet overeenkomen met naam parameter Action-methode
- Bij deze notatie: id-parameter verplicht!



# Routing templates

- Opmerkingen:
  - Indien parameter gespecificeerd, is deze verplicht!
    - **Voorbeeld:** {controller}/{action}/{id}
    - **URL:** students/edit → **geen** match!
  - Aantal parameters is onbeperkt
    - **Voorbeeld:** {controller}/{action}/{category}/{id}
    - **URL:** /recipes/view/soups/3 → **match!**
  - Parameters hoeven niet in aparte segmenten te staan
    - **Voorbeeld:** {controller}/{action}/{from}-{to}
    - **URL:** /currencies/convert/USD-EUR → **match!**

# Routing templates

- Om een parameter **optioneel** te maken, zet je een vraagteken (?) achter de naam van deze parameter:
  - **Voorbeeld:** {controller}/{action}/{id<sup>?</sup>}

| URL              | Controller         | Action       | ID-parameter |
|------------------|--------------------|--------------|--------------|
| /students/edit/3 | StudentsController | Edit(int id) | 3            |
| /movies/view/12  | MoviesController   | View(int id) | 12           |
| /home/about      | HomeController     | About()      | <b>null</b>  |

## Routing templates

- Het is tevens mogelijk om **default-waarden** te specificeren voor route parameters
  - Voorbeeld: `api/{controller}/{action=index}/{id?}`

| URL                | Match? | Controller        | Action                                   | ID-parameter |
|--------------------|--------|-------------------|------------------------------------------|--------------|
| api/product/view/3 | ✓      | ProductController | View(int? id)                            | 3            |
| api/product/view   | ✓      | ProductController | View(int? id)                            | null         |
| api/product        | ✓      | ProductController | <b>Index()</b> ( <i>default-waarde</i> ) | null         |
| api                | ✗      |                   |                                          |              |

Opmerking: **int?** is een nullable-type

→ zie <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/nullable-value-types>



# Routing: Configuratie

# Routing configuratie

- Routing templates kunnen op twee manieren **geconfigureerd** worden:
  - **Via attributen:** in controller-klasse zelf (vooral bij Web API's, *zie later*)
  - **Conventionele routing:** in *Program.cs*

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

## Opmerkingen:

- Je kunt **zoveel** routes toevoegen **als nodig**
- Naam wordt enkel gebruikt voor het **genereren** van URL's op basis van template




# Redirects

## Redirect: Action in zelfde controller

- Met behulp van **RedirectToAction()** kan je de gebruiker doorsturen naar een andere Action-methode binnen dezelfde controller:

```
public class AccountController : Controller {  
    public IActionResult Index() {  
        return RedirectToAction("Details");  
    }  
  
    public IActionResult Details() {  
        return View();  
    }  
}
```




- Enkel Action-naam → redirect naar Action-methode met deze naam **binnen dezelfde controller**

## Redirect: Action in zelfde controller (met parameters)

- Gebruik **anonymous type** om bijkomende route-parameters mee te geven als extra parameter

```
public class AccountController : Controller {  
    public IActionResult Index() {  
        return RedirectToAction("Details", new { id = 1 });  
    }  
  
    public IActionResult Details(int id) {  
        return View();  
    }  
}
```



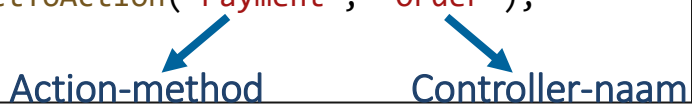
- Naam van property in anonymous type  
= naam van parameter in Action-method



## Redirect: Action in andere controller

- Gebruik tweede String-parameter om te verwijzen naar Action-method in **andere controller**

```
public class CartController : Controller {  
    public IActionResult CheckOut() {  
        return RedirectToAction("Payment", "Order");  
    }  
}
```



- Eerste parameter = naam van Action-method
- Tweede parameter = naam van controller

```
public IActionResult CheckOut() {  
    return RedirectToAction("Payment", "Order", new {currency="EUR"});  
}
```

- Kan ook in combinatie met parameters

## Redirect: doel op basis van route-naam

```
app.MapControllerRoute(
    name: "currencies",
    pattern: "{controller}/{currency}/{action=view}");

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

- Locatie voor routing kan ook gegenereerd worden op basis van **route-template**
- Gebruik hiervoor RedirectToRoute
- Eerste parameter: name-property van de route, tweede parameter = anonymous type met parameters

```
public IActionResult Convert() {
    return RedirectToRoute("currencies", new { controller="Currency", currency="EUR" });
}
```

route-name

route-values



# Controllers & Views

# Inleiding

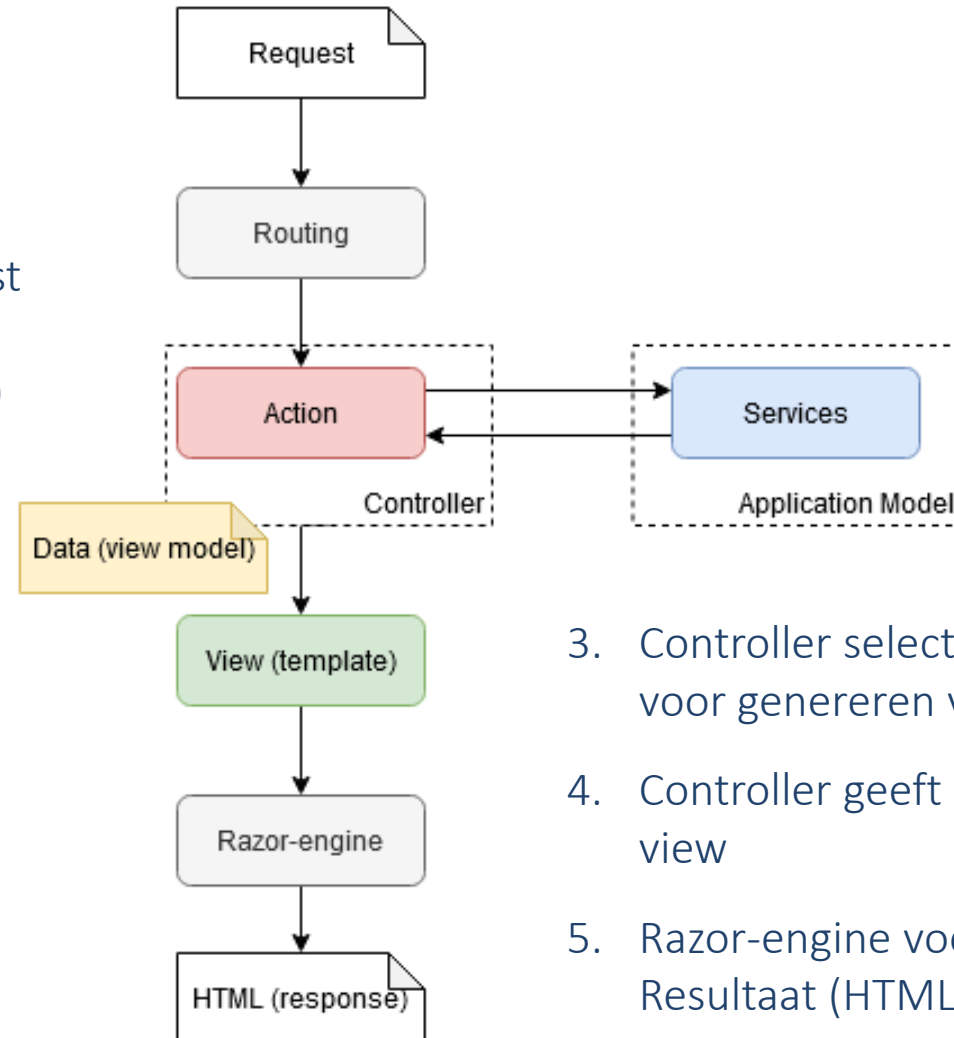
- Taak van controller:
  - Inkomende **HTTP-request afhandelen** (wijzigingen aanbrengen in model, bv.: gegevens valideren, records ophalen/wegschrijven in databank, records updaten in databank, ...). Kan hiervoor gebruik maken van **parameter(s)** die aan Action-method wordt meegegeven (bv.: id van weer te geven entiteit)
  - Correcte **View** (-template) **selecteren** als response op request
  - **Data meegeven** aan View (-template) om response (HTML, JSON, ...) te genereren

# Inleiding

- Views:
  - In ASP.NET MVC webapplicatie worden views vaak **dynamisch gegenereerd** met behulp van **Razor** (= template engine)
  - Razor: **combinatie** van statische **HTML** en **C#-code**
  - Controller geeft data door aan View → wordt via Razor-template weergegeven/gebruikt voor logica in View
  - Voorbeeld features:
    - ✓ HTML-code genereren afhankelijk van conditie (**if-else**)
    - ✓ **foreach** om over items in **List<>** te itereren en voor elk item nodige HTML-code te genereren
    - ✓ Waarden van C#-property's opvragen om weer te geven
    - ✓ ...

# Inleiding

1. Request wordt via routing naar juiste controller en action gestuurd
2. Action-methode handelt request af (haalt gegevens op uit DB, schrijft records weg naar DB, ...)



3. Controller selecteert view (-template) voor genereren van response
4. Controller geeft nodige data mee aan view
5. Razor-engine voert template-code uit. Resultaat (HTML-code) wordt als response teruggestuurd

# Voorbeeld

Request: <https://localhost:1234/Todos/List/3>

## TodosController.cs

```
public IActionResult List(int id) {  
    //data ophalen in DB  
    List<string> todos = _todoRepository.GetTodosForUser(id);  
  
    //selecteer view (default: zelfde als Action-method naam)  
    //nl.: List.cshtml  
    //Geef lijst met todo's mee aan view  
    return View(todos);  
}
```

## List.cshtml

```
@model List<string>  
  
<h1>TODO's</h1>  
<ul>  
    @foreach (var item in Model)  
    {  
        <li>@item</li>  
    }  
</ul>
```

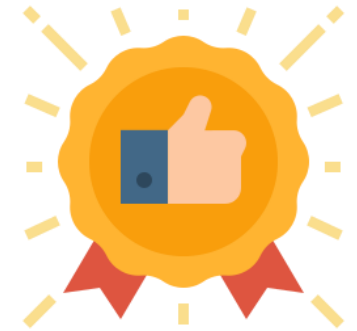


Example Razor   Home

## TODO's

- Auto afwassen
- Boodschappen doen
- Hond uitlaten

# Separation of concerns



- Voordeel:
  - **Separation of concerns:** view bepaalt hoe data wordt weergegeven
    - Zelfde data kan op verschillende manieren weergegeven worden
    - Wijziging in view vereist enkel aanpassing van Razor-template

| Example Razor   Home |                   |
|----------------------|-------------------|
| <b>TODO's</b>        |                   |
| #                    | Title             |
| 1                    | Auto afwassen     |
| 2                    | Boodschappen doen |
| 3                    | Hond uitlaten     |

|                                                                                                                     |  |
|---------------------------------------------------------------------------------------------------------------------|--|
| Example Razor   Home                                                                                                |  |
| <b>TODO's</b>                                                                                                       |  |
| <ul style="list-style-type: none"><li>• Auto afwassen</li><li>• Boodschappen doen</li><li>• Hond uitlaten</li></ul> |  |



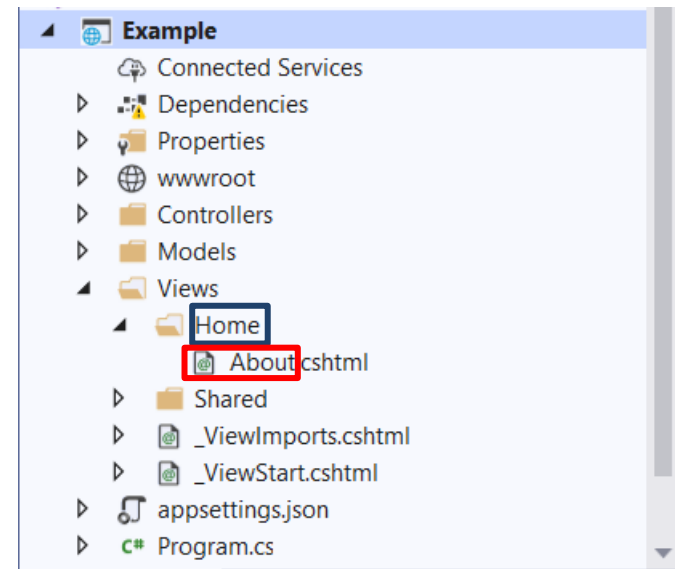


# View selectie

# Hoe selecteert Controller een view?

- Action-method retournt ViewResult
  - Gebruik maken van `View()` helper-methode
    - Zoekt template, voorziet deze (eventueel) van data en voert de template uit
  - Zoekt standaard in folder `Views/{naam controller}`
  - Indien geen argument voor `View()`-methode: standaard view met zelfde naam als Action-method

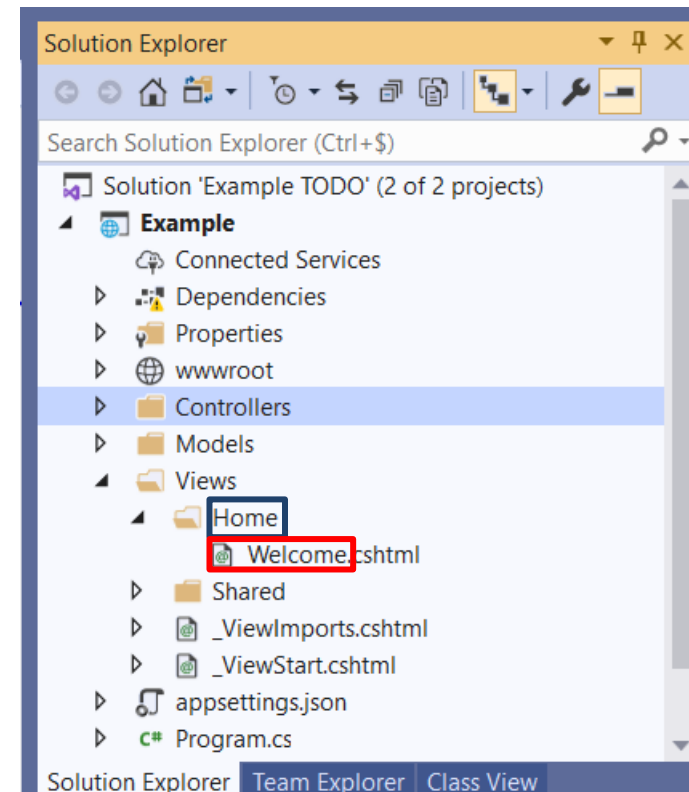
```
public class HomeController : Controller {  
    public IActionResult About() {  
  
        return View(); //default: zelfde naam als Action-method  
    }  
}
```



## Hoe selecteert Controller een view?

- Afwijken van conventie: extra argument meegeven aan View()-methode

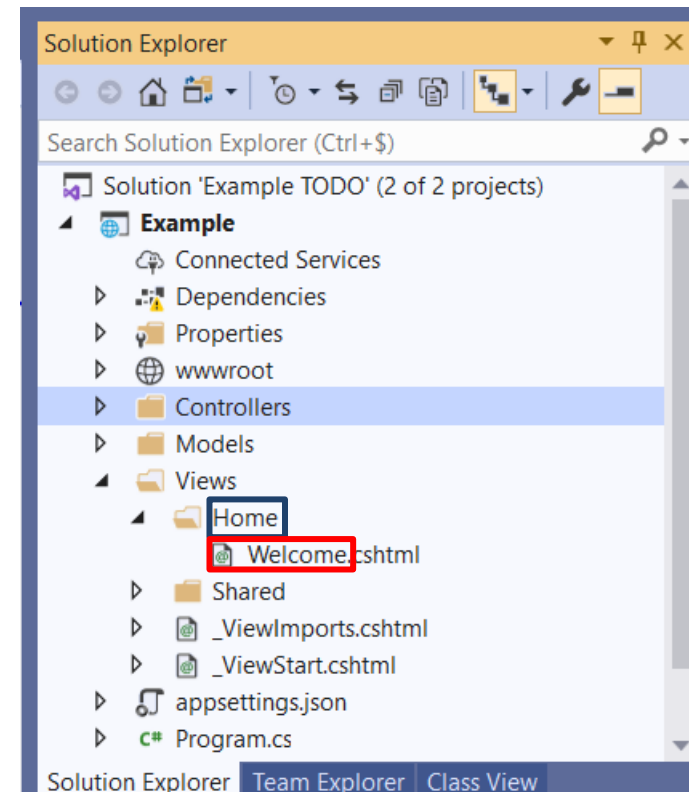
```
public class HomeController : Controller {  
    public IActionResult Index() {  
  
        return View("Welcome");  
    }  
}
```



## Hoe selecteert Controller een view?

- Ook mogelijk om **absoluut pad** op te geven, beginnend van de root van het project

```
public class HomeController : Controller {  
    public IActionResult Index() {  
        return View("Views/Home/Welcome.cshtml");  
    }  
}
```



## Hoe selecteert Controller een view?

- Opmerkingen:
  - Controller kan ook **redirects** uitvoeren via `RedirectToAction()` of `RedirectToRoute()` (zie hoofdstuk Routing)
  - View kan bepaald worden via logica (bv.: *if-else*):

```
public IActionResult Details(int id) {  
    Product product = _productRepository.FindById(id);  
  
    if(product == null) {  
        return View("NotFound"); //Views/Product/NotFound.cshtml  
    } else {  
        return View(product); //Views/Product/Details.cshtml  
    }  
}
```



# Data doorgeven aan View

## Data doorgeven aan View

- Een controller kan op verschillende manieren data doorgeven aan View:
  - **ViewData:** dictionary (key-value paren)
  - **ViewBag:** wrapper rond **ViewData**. Maakt gebruik van dynamic om casting te vermijden
  - **View model:** eigen C#-klasse die property's bevat voor alle data die aan view moet doorgegeven worden

# Data doorgeven aan View: ViewData

## Voorbeeld ViewData:

### HomeController.cs

```
public IActionResult Index() {  
    ViewData["Message"] = "Welcome to my website!";  
  
    return View();  
}
```

### Views/Home/Index.cshtml

```
<h1>Home</h1>  
<p>Message: @ViewData["Message"]</p>
```



## Home

Message: Welcome to my website!



# Data doorgeven aan View: ViewBag

## Voorbeeld ViewBag:

### HomeController.cs

```
public IActionResult Index() {  
    ViewBag.Title = "Welcome!";  
    ViewBag.Message = "Hello, World!";  
  
    return View();  
}
```

### Views/Home/Index.cshtml

```
<h1>@ViewBag.Title</h1>  
<p>Message: @ViewBag.Message</p>
```



Welcome!

Message: Hello, World!

*Opmerking: de property's Title en Message worden "at runtime" aangemaakt en aan het ViewBag-object toegevoegd. Je krijgt hierbij géén auto-completion!*

## Data doorgeven aan View

- **MAAR:** ViewData en ViewBag zijn minder geschikt voor complexe objecten:

### ViewData:

```
@{  
    Product product = (Product)ViewData["Product"];  
}  
  
<p>Naam: @product.Name</p>  
<p>Beschrijving: @product.Description</p>  
<p>Prijs: &euro; @product.Price</p>
```



*Cast nodig → gevoelig voor fouten!*

### ViewBag:

```
<p>Naam: @ViewBag.Product.Name</p>  
<p>Beschrijving: @ViewBag.Product.Description</p>  
<p>Prijs: &euro; @ViewBag.Product.Price</p>
```



*Dynamic → geen compile-time type-checking  
→ Gevoelig voor fouten (runtime-exceptions  
mogelijk!)*

## Data doorgeven aan View

- Gebruik `ViewData` en `ViewBag` enkel voor beperkte en eenvoudige data
- Veel gegevens en/of complexe objecten doorgeven → maak een **View model-klasse**
  - **View model** = C#-klasse die property's bevat voor alle gegevens die aan View moeten doorgegeven worden

# Voorbeeld ViewModel

## ProductDetailsViewModel.cs

```
public class ProductDetailsViewModel {  
    public Product Product { get; set; }  
    public int NumberInStock { get; set; }  
}
```

View model **combineert** gegevens van **verschillende modellen** om deze als één geheel weer te geven

## ProductController.cs

```
public class ProductController : Controller {  
    private ProductRepository _productRepository = new ProductRepository();  
    private InventoryRepository _inventoryRepository = new InventoryRepository();  
  
    public IActionResult Details(int id) {  
        ProductDetailsViewModel viewModel = new ProductDetailsViewModel()  
        {  
            Product = _productRepository.FindById(id),  
            NumberInStock = _inventoryRepository.GetStockForProduct(id)  
        };  
  
        return View(viewModel);  
    }  
}
```



# Razor

## Razor: HTML genereren

- Controller geeft C#-object (view model) door aan view om response (bv.: HTML-code) te genereren
- Object dat door controller werd doorgegeven is in Razor-template beschikbaar via Model-property
- Template bevat code om property's van Model op te vragen en weer te geven
- Indien model wordt doorgegeven, spreekt men van “strongly typed view”

# Razor: voorbeeld

## Views/Products/Details

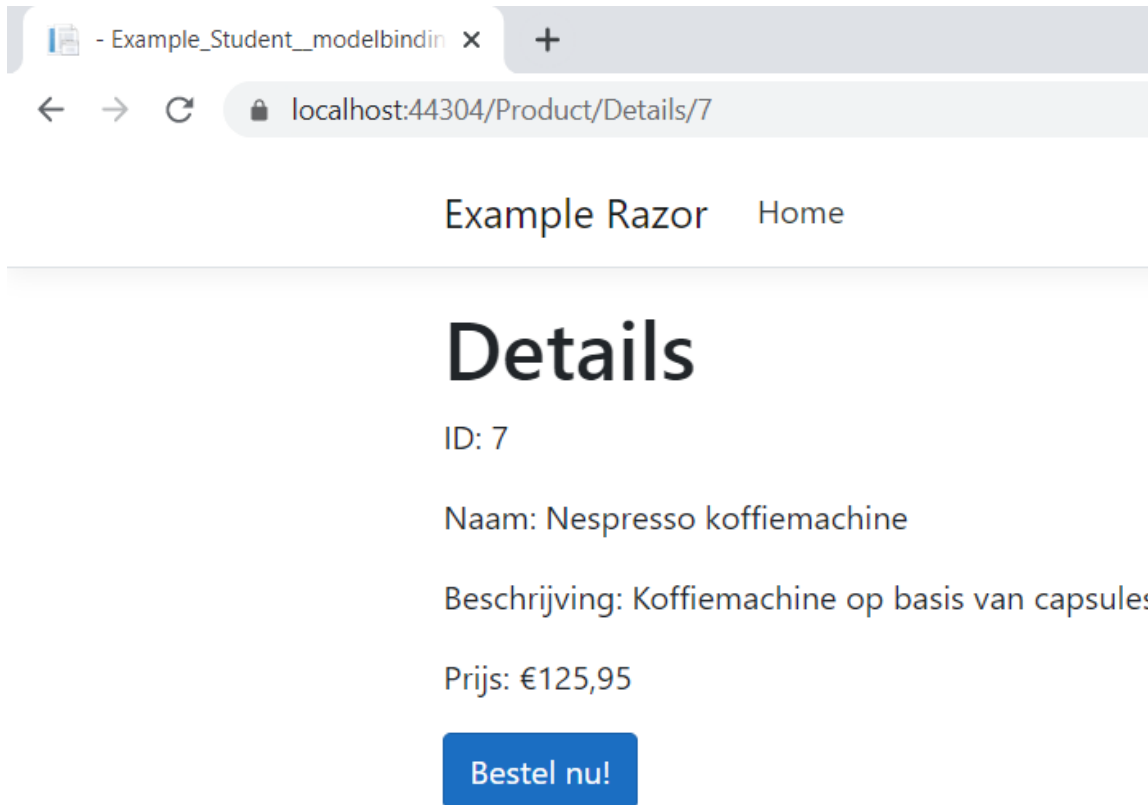
```
@model ProductDetailsViewModel

<h1>Details</h1>
<p>ID: @Model.Product.ID</p>
<p>Naam: @Model.Product.Name</p>
<p>Beschrijving: @Model.Product.Description</p>
<p>Prijs: &euro;@Model.Product.Price</p>

@if (Model.NumberInStock > 0)
{
    <button class="btn btn-primary">Bestel nu!</button>
}
else
{
    <p class="text-danger">Dit product is niet meer op voorraad!</p>
}
```

- **@model-directive:** geeft aan welk type object door controller werd doorgegeven
- **Model-property:** verwijst naar object dat door controller werd doorgegeven
- **If-else:** HTML-code genereren op basis van conditie

# Razor: voorbeeld





## Voorbeeld Razor-syntax



- Eenvoudige expressies:

```
<p>Copyright @DateTime.Now.Year &copy;</p>
```

- Expressie met evaluatie

```
<p>De som van 5 en 10 is: @(5 + 10)</p>
```

Meer voorbeelden: *zie inleiding MVC*

## Voorbeeld Razor-syntax



- Condities:

```
@model ProductDetailsViewModel

@if (Model.NumberInStock > 0)
{
    <button class="btn btn-primary">Bestel nu!</button>
}
else
{
    <p class="text-danger">Dit product is niet meer op voorraad!</p>
}
```



## Voorbeeld Razor-syntax

- Iteraties (bv.: foreach):

```
@model ProductDetailsViewModel

<h1>Product list</h1>
<table class="table table-striped">
  <thead>
    <tr>
      <th>#</th>
      <th>Naam</th>
      <th>Prijs</th>
    </tr>
  </thead>
  <tbody>
    @foreach (var product in Model.Products)
    {
      <tr>
        <td>@product.ID</td>
        <td>@product.Name</td>
        <td>@product.Price.ToString("c")</td>
      </tr>
    }
  </tbody>
</table>
```

} Voor elk item een nieuwe table-row (<tr>) genereren