



samen sterk voor werk

ASP.NET MVC

Webapplicaties maken met
ASP.NET MVC 5

Cursus

Deze cursus is eigendom van VDAB Competentiecentra ©

[Opmerkingen]

versie: 3/06/2015

INHOUD

1	Inleiding	9
1.1	Doelstelling.....	9
1.2	Vereiste voorkennis.....	9
1.3	Nodige software	9
2	Wat is ASP.NET MVC?.....	11
2.1	ASP.NET Webforms – ASP.NET Web pages – ASP.NET MVC.....	11
2.2	Het MVC pattern in ASP.NET.....	12
2.2.1	MVC – Model-View-Controller	12
2.2.2	Het verwerken van een request in een MVC applicatie	12
2.2.2.1	Een GET request verwerken.....	12
2.2.2.2	Een POST request verwerken.....	13
2.2.3	Routing.....	13
3	Een eerste ASP.NET MVC-webapplicatie	15
3.1	Een nieuw ASP.NET MVC Web Application project	15
3.2	Inhoud van het MVC project	16
3.3	De controllers	17
3.4	De views	17
3.5	De models.....	18
3.6	Zelf een controller toevoegen : de FiliaalController	18
3.7	Templates	20
3.8	Extra actions in de filiaalcontroller.....	21
3.9	Samenvatting.....	22
4	Meer over controllers.....	23
4.1	De Controller class.....	23
4.2	Controller methods	23
4.3	Returntypes	25

4.4	ViewResult	26
4.5	Samenvatting	27
5	Razor	29
5.1	De ASPX en Razor View Engines	29
5.2	Razor syntax.....	30
5.2.1	Het @-teken	30
5.2.2	Commentaar, character encoding, letterlijke tekst	31
5.3	Samenvatting	31
5.4	Oefening.....	31
6	Data doorgeven van controller naar view	33
6.1	Viewbag.....	33
6.2	Strongly typed data.....	34
6.3	Samenvatting	37
6.4	Oefening.....	37
7	Code hergebruiken met Layouts, Partial Views en @helper-functies ...	39
7.1	Layouts.....	39
7.1.1	Standaard opbouw van een view.....	39
7.1.2	Bundling en minification	40
7.1.3	Een andere layout instellen	41
7.1.4	Sections	42
7.2	Partial Views	43
7.3	@helper	47
7.4	Samenvatting	48
7.5	Oefening.....	48
8	Publiceren	49
8.1	Oefening.....	51
9	Opmaak van data	53
9.1	Opmaak in het cshtml-bestand.....	53

9.2	Opmaak gecentraliseerd in het model.....	54
9.3	UIHint.....	55
9.4	Opmaak voor enumeration-type.....	56
9.5	Samenvatting.....	57
9.6	Oefening	57
10	Gegevens over meerdere pagina's heen onthouden	59
10.1	Cookies	59
10.2	Session state.....	63
10.3	Application state	65
10.4	Samenvatting.....	68
11	Tempdata en redirects	69
11.1	Een verwijderoptie toevoegen	69
11.2	Samenvatting.....	71
11.3	Oefening	71
12	Hyperlinks	73
12.1	Verwijzingen naar views met Html.ActionLink en Url.Action	73
12.2	Redirection met Redirect en RedirectToAction	74
12.3	Samenvatting.....	75
12.4	Oefening	75
13	Html forms, html helpers en validatie	77
13.1	Voorbereiding.....	77
13.2	Een persoon-object verwijderen.....	80
13.3	Een POST-formulier samenstellen met html-helpers	81
13.3.1	De form-class	81
13.3.2	De form tonen met een GET request.....	82
13.3.3	De wijziging doorvoeren met een POST-request.....	84
13.3.4	Validatie van het formulier	84

13.4	Een GET-formulier samenstellen met html-helpers	88
13.4.1	De form-class	88
13.4.2	De form tonen met een GET request.....	89
13.4.3	De form verwerken met een GET-request.....	89
13.4.4	Bugfix voor decimale waarden.....	90
13.4.5	Meerdere velden t.o.v. elkaar valideren.....	91
13.5	Input elements genereren met Html.EditorFor.....	93
13.5.1	Een persoon toevoegen	93
13.5.2	De layout aanpassen	95
13.6	Meer over validatie.....	98
13.6.1	StringLength	98
13.6.2	Regularexpression	99
13.6.3	Compare	99
13.6.4	Een eigen validatie attribuut.....	99
13.6.5	Client side validation.....	100
13.7	Samenvatting	102
13.8	Oefening.....	102
14	Data uit een database lezen met het entity framework	103
14.1	De database voorbereiden	103
14.2	Scaffolding.....	103
14.3	Buddy classes	105
14.4	Een zoekform toevoegen.....	106
14.5	De applicatie publiceren	109
14.5.1	Een nieuwe SQL Server user	109
14.5.2	De webapplicatie publiceren	110
14.6	Samenvatting	111
14.7	Oefening.....	111
15	Upload.....	113
15.1	Verwijzing naar een uploadformulier	113
15.2	Het uploadformulier	113

15.3	Het formulier verwerken.....	114
15.4	Foto weergeven in detailpagina	114
16	Globalization	117
16.1	Taal en Locale	117
16.2	Resourcefiles	117
16.3	Verschillende teksten en datum- en getalnotaties gebruiken.....	118
16.4	Voorbeeld	118
16.4.1	Het navigatiemenu	118
16.4.2	Meertalige homepagina.....	120
16.4.3	De meertaligheid instellen via knoppen	121
16.4.4	Errormessages in verschillende talen	123
16.4.5	Labels in verschillende talen.....	124
16.5	Samenvatting.....	125
17	ActionFilters	127
17.1	ActionFilters voor alle controllers	127
17.2	ActionFilters voor één controller	129
17.3	Filter overrides	129
17.4	Samenvatting.....	130
18	Custom Routing.....	131
18.1	Routes gebaseerd op een URI template	131
18.2	Routes gebaseerd op een query string	134
18.3	Attribute routing	138
18.3.1	Attribute routing inschakelen en toepassen	138
18.3.2	Default en optionele parameters	139
18.3.3	Route prefixes en defaultroutes	140
18.3.4	Route Constraints	142
18.3.5	Custom route constraints	143
18.4	Samenvatting.....	146
19	Security	147

19.1	Authentication	147
19.2	Een beveiligde ASP.NET MVC webapplicatie	148
19.3	De database	148
19.4	Identity model.....	150
19.5	Security administratie.....	151
19.5.1	Gegevens zelf intikken	151
19.5.2	Een eigen user management systeem	152
19.5.3	Seeding.....	161
19.6	Actions beveiligen met de [Authorize] annotation	163
19.7	Een webapplicatie met authentication publiceren	165
19.8	Samenvatting	167
19.9	Oefening.....	167
20	Bootstrap	169
20.1	Bootstrap-classes in MVC web applicaties	169
20.2	Het bootstrap grid systeem	170
20.3	Bootstrap componenten.....	172
20.4	Bootstrap tekstopmaak	174
20.5	Bootstrap forms.....	175
20.6	Bootstrap tables.....	176
20.7	Samenvatting	176
21	Project Cultuurhuis.....	177
21.1	Opgave	177
21.1.1	Een voorstelling kiezen	177
21.1.2	Het aantal tickets ingeven.....	178
21.1.3	Overzicht van de gevraagde tickets	179
21.1.4	Identificatie van de klant.....	180
21.1.5	Overzicht gelukte/mislukte reservaties	182
21.2	Een oplossing	183

21.2.1	Het data model	183
21.2.2	De class CultuurService.....	183
21.2.3	De page Voorstellingen.....	183
21.2.4	De page Reserveren.....	189
21.2.5	Het winkelmandje.....	191
21.2.6	Bevestiging van de reservaties : klantgegevens opzoeken.....	195
21.2.7	Bevestiging van de reservaties : een nieuwe klant.....	198
21.2.8	Bevestiging van de reservaties : reservaties doorvoeren.....	203
21.2.9	Overzicht van de reservaties	204
22	Appendix A : Browsers en webservers.....	207
22.1	Algemeen.....	207
22.2	Requests, responses, HTTP, TCP/IP	207
22.3	URL.....	208
22.4	Statische pagina's – dynamische pagina's	209
22.4.1	Statische pagina	209
22.4.1.1	Algemeen.....	209
22.4.1.2	Voorbeeld.....	209
22.4.1.3	Overzicht van een request naar een statische pagina	209
22.4.2	Een dynamische pagina	210
22.4.2.1	Algemeen.....	210
22.4.2.2	Voorbeeld.....	210
22.4.2.3	Overzicht van een request naar een dynamische pagina	211
22.4.2.4	De IIS-webserver	211
22.5	Onderdelen van een request.....	211
22.5.1	HTTP method	211
22.5.2	Headers.....	212
22.5.3	Body	212
22.6	Query string.....	213
22.7	Onderdelen van een response	214
22.7.1	Status code	214
22.7.2	Headers.....	214
22.7.3	Body	215

22.8	Samenvatting van request en response en hun onderdelen	215
23	Appendix B : Tips bij gebruik van Windows 8	217
23.1	Gegevensopmaak.....	217
23.2	Geen afbeeldingen.....	217
23.3	Taalinstellingen	217
23.4	SQLServer instance instellen.....	218
24	Appendix C : Oplossing oefeningen	219
24.1	Fibonacci-reeks (p. 31).....	219
24.2	Bieren (p. 37)	219
24.3	Logo Biertempel en adresgegevens (p.48)	220
24.4	Extra opmaak (p.55).....	220
24.5	Een bier verwijderen (p.69)	221
24.6	Hyperlinks (p. 73).....	223
24.7	Nieuw bier toevoegen en layout verzorgen (p. 100).....	223
24.8	Bierendata uit databank (p. 107).....	226
24.9	Beveiligde bierenapplicatie (p. 161)	227

1 Inleiding

1.1 Doelstelling

In deze cursus leer je webapplicaties maken met de ASP.NET MVC technologie van Microsoft. Deze cursus is gebaseerd op ASP.NET MVC 5.

1.2 Vereiste voorkennis

Voor deze cursus is volgende voorkennis vereist :

- C#
- HTML5/CSS
- XML fundamentals
- Kennis van browsers en webservers en hoe een webpagina tot stand komt. Wie dit niet kent leest best **eerst** Appendix A achteraan deze cursus.

1.3 Nodige software

Bij de aanvang van de cursus veronderstellen we dat volgende software geïnstalleerd is :

- Visual Studio 2013 mét de recentste update
- SQL Server of SQL Server Express en eventueel SQL Server Management Studio

2 Wat is ASP.NET MVC?

2.1 ASP.NET Webforms – ASP.NET Web pages – ASP.NET MVC

Kies je voor C# om webapplicaties te maken, dan kan je dat op drie verschillende manieren doen. Microsoft biedt volgende technologieën aan :

- ASP.NET Webforms (<http://www.asp.net/web-forms>)

Met ASP.NET Webforms heeft Microsoft geprobeerd een ontwikkelmethode voor webapplicaties te maken die dezelfde 'look & feel' heeft als deze voor Winforms applicaties. Webforms ontwikkelaars gebruiken net als bij Winforms meestal een grafische ontwikkelomgeving waarbij ze controls op een formulier (form) slepen. Bijkomende code wordt toegevoegd door code te koppelen aan events die horen bij allerlei controls zoals buttons e.d. Alle informatie over ASP.NET Webforms lees je in de gelijknamige VDAB-cursus.

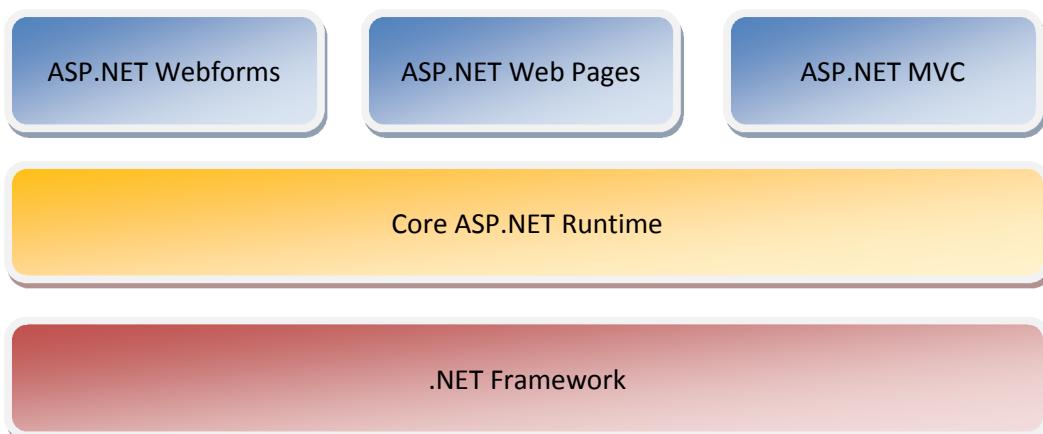
- ASP.NET Web Pages (<http://www.asp.net/web-pages>)

ASP.NET Web Pages is een minder gekende technologie voor het ontwikkelen van dynamische websites. Dat doe je via WebMatrix, een laagdrempelige ontwikkelomgeving die gebruikt wordt in combinatie met IIS Express als webserver en SQL Compact als databaseserver.

- ASP.NET MVC (<http://www.asp.net/mvc>)

Net als bij de twee voorgaande technologieën kan je met ASP.NET MVC webapplicaties maken. In dit geval gebeurt dit volgens het algemeen erkende MVC pattern. MVC staat voor Model-View-Controller. MVC is niet typisch .NET of Microsoft maar een algemeen gekend patroon in de software architectuur. Bij heel wat ontwikkelaars geniet ASP.NET MVC de voorkeur boven ASP.NET Webforms. We zien zo dadelijk waarom.

Elk van deze drie technologieën wordt ondersteund én verder ontwikkeld door Microsoft. Er wordt telkens gebruik gemaakt van de Core ASP.NET Runtime. De Core ASP.NET Runtime maakt op haar beurt dan weer gebruik van het .NET Framework.



2.2 Het MVC pattern in ASP.NET

2.2.1 MVC – Model-View-Controller

In de software architectuur doen zich vaak gelijkaardige problemen voor. Voor deze problemen zijn modeloplossingen gezocht én gevonden die we (*design*) patterns noemen. Een goed boek hierover is *Patterns of Enterprise Application Architecture* van Martin Fowler.

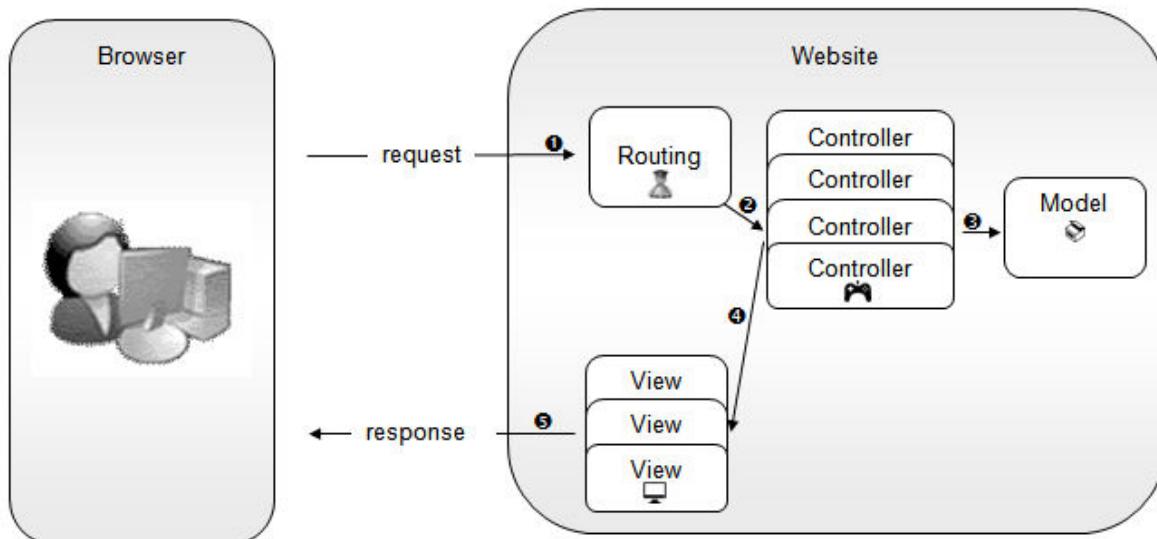
MVC of Model-View-Controller is zo één pattern. In een MVC applicatie verwerken de Controllers een vraag naar informatie en zoeken daartoe gegevens op in een Model (gegevens in tabellen of lijsten). De uitvoer wordt doorgegeven aan en gepresenteerd door een View. Dit systeem zorgt voor een mooie scheiding van verantwoordelijkheden. Bovendien is het ook veel makkelijker te unit-testen omdat je de code uit de achterliggende business logic gewoon kan testen zonder dat je daarvoor een volledig uitgewerkte user interface nodig hebt.

2.2.2 Het verwerken van een request in een MVC applicatie

Hoe ziet de programmaflow van een MVC-project er nu uit? Eén van de aanwezige controllers verwerkt de vraag van de client (browser). Als de request aan de kant van de website gewoon gegevens opvraagt en dus geen toevoegingen, wijzigingen of verwijderingen doet (een idempotent request) dan gebruik je een GET request. Wijzig, verwijder of voeg je wél iets toe dan gebruik je een POST request.

2.2.2.1 Een GET request verwerken

Bij een GET request ❶ beslist een routing-systeem ❷ welke controller de verwerking van de vraag doet. De keuze gebeurt op basis van de URL van de request. Zodadelijk meer hierover.



Vervolgens verwerkt één van de controllers de requestparameters en de request body. De controller kan tijdens dit verwerkingsproces het Model aanspreken (❸).

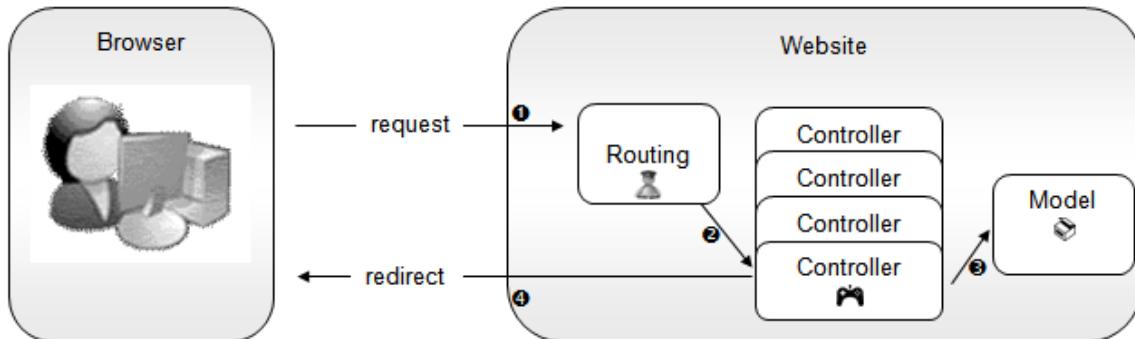
In het Model zitten classes die geen verband houden met de request of de response. Deze classes zijn entities, classes voor de databasetoegang, WCF of REST service clients,...

Wanneer de verwerking beëindigd is, kiest de controller een view **④** om het resultaat in weer te geven. Deze View bouwt de response **⑤** naar de browser op, op basis van modeldata die het van de controller heeft doorgekregen.

Meestal maken views HTML-code aan maar het kan even goed JSON of andere code zijn. Een view bevat in elk geval geen business logica of database access code. Deze horen thuis in het Model.

2.2.2.2 Een POST request verwerken

Wanneer je data in de database toevoegt, verwijdert of wijzigt gebruik je dus een POST request.



Na de verwerking van de request **①②③** gebeurt er een redirect **④** naar een andere pagina. De redirect bevat geen body met HTML, maar een status code 302 en een header met een Location parameter waarin de URL staat. De browser doet dan onmiddellijk een HTTP GET request naar die URL.

2.2.3 Routing

We hadden het daarnet al even over een routing-systeem dat beslist welke controller de request verwerkt. De routing is gebaseerd op een RouteTable en dat is zoals de naam zelf zegt een tabel met de te volgen routes.

Deze RouteTable wordt geïnitialiseerd in een bestand *RouteConfig.cs* in de folder *App_Start* en bevat reeds een default route. Later in deze cursus zullen we eigen routes toevoegen.

Om de werking van zo een route beter te begrijpen bekijken we even die default route. In het bestand *RouteConfig.cs* vinden we volgende code terug :

```

routes.MapRoute(
    name: "Default",
    url: "{controller}/{action}/{id}",
    defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
);

```

De method *MapRoute()* heeft als eerste parameter de naam van de route. In dit geval is dit "**Default**", de naam van de default route, logisch. De lijn eronder is belangrijker : zoals reeds aangegeven is het de URL die beslist welke controller er wordt uitgevoerd. Dus het eerste deel van de URL geeft de te kiezen controller aan. Vervolgens staat er een action aangegeven en een id. De action is de naam van een method in de opgegeven controller, de id is een mogelijke parameter. Wanneer er geen controller, action of id wordt opgegeven dan worden de defaultwaarden uit de laatste codelijn gebruikt. Dat is dus de 'Home'-controller en de action 'Index'.

In onderstaande tabel verduidelijken we dit met een paar voorbeelden.

URL	Controller	Controller method	Method parameter id
vdab.be/Filialen>Show/1	FilialenController	Show	1
vdab.be/Werknemers/Find/2	WerknemersController	Find	2
vdab.be/Werknemers/All	WerknemersController	All	
vdab.be/Werknemers	WerknemersController	Index	
vdab.be	HomeController	Index	

In het eerste voorbeeld zorgt de URL "vdab.be/Filialen>Show/1" ervoor dat in de **FilialenController** de action **Show** worden uitgevoerd met parameterwaarde "**1**". Het tweede voorbeeld is gelijkaardig. Daar wordt de Find method uitgevoerd met parameterwaarde "**2**" en dat binnen de Werknemers-Controller.

In het derde voorbeeld wordt duidelijk dat een parameter niet altijd verplicht is. De method All van de WerknemersController wordt uitgevoerd.

Wanneer we in het vierde voorbeeld de action weglaten, dan wordt de default action uitgevoerd en dat is de Index-method. Laten we alles weg (laatste voorbeeld) dan wordt de Index-method van de HomeController uitgevoerd.

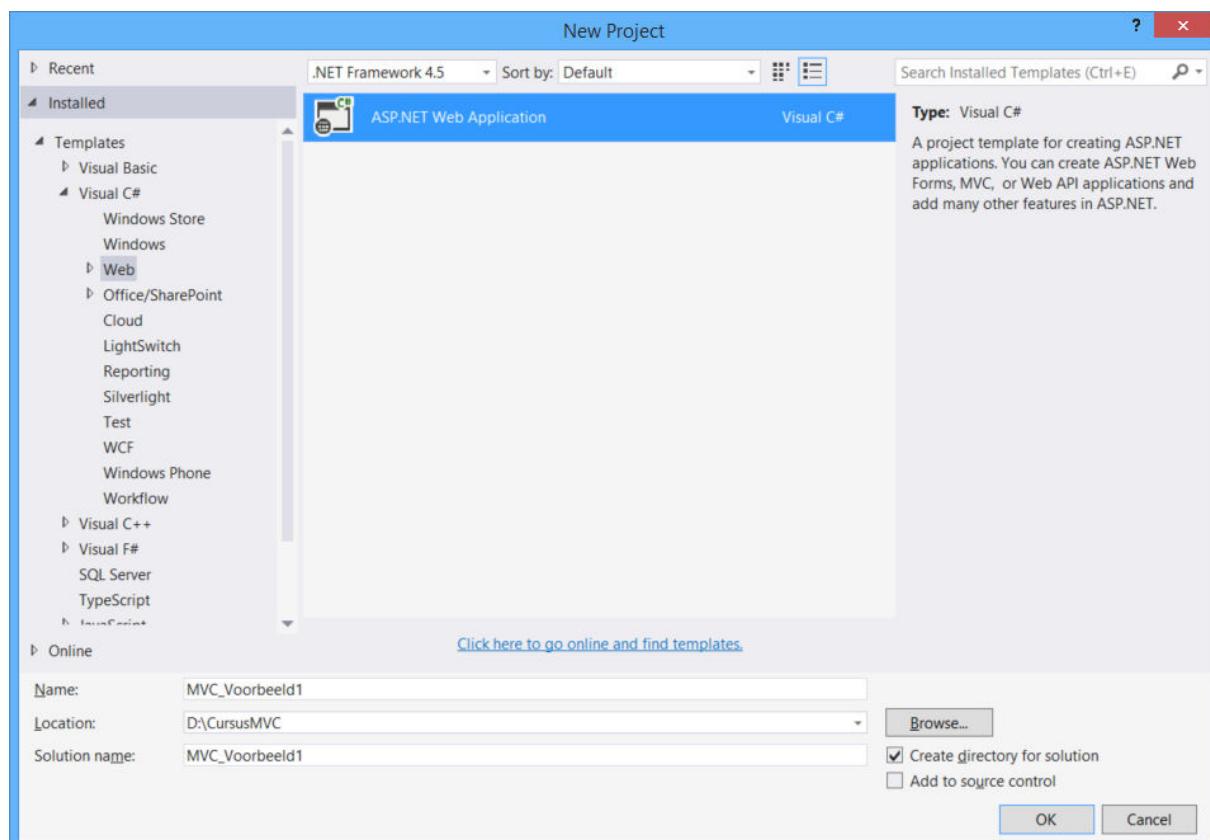
In het onze eerste MVC Webapplicatie zullen we dit effectief uitproberen.

3 Een eerste ASP.NET MVC-webapplicatie

Aan de slag nu. We brengen de theorie uit het vorige hoofdstuk in de praktijk met een eerste ASP.NET MVC Webapplicatie.

3.1 Een nieuw ASP.NET MVC Web Application project

Om een nieuwe ASP.NET MVC Web Application aan te maken kies je in Visual Studio voor *FILE – New – Project...*. Onder *Installed / Templates / Visual C# / Web* kies je aan de rechterkant voor *ASP.NET Web Application*.

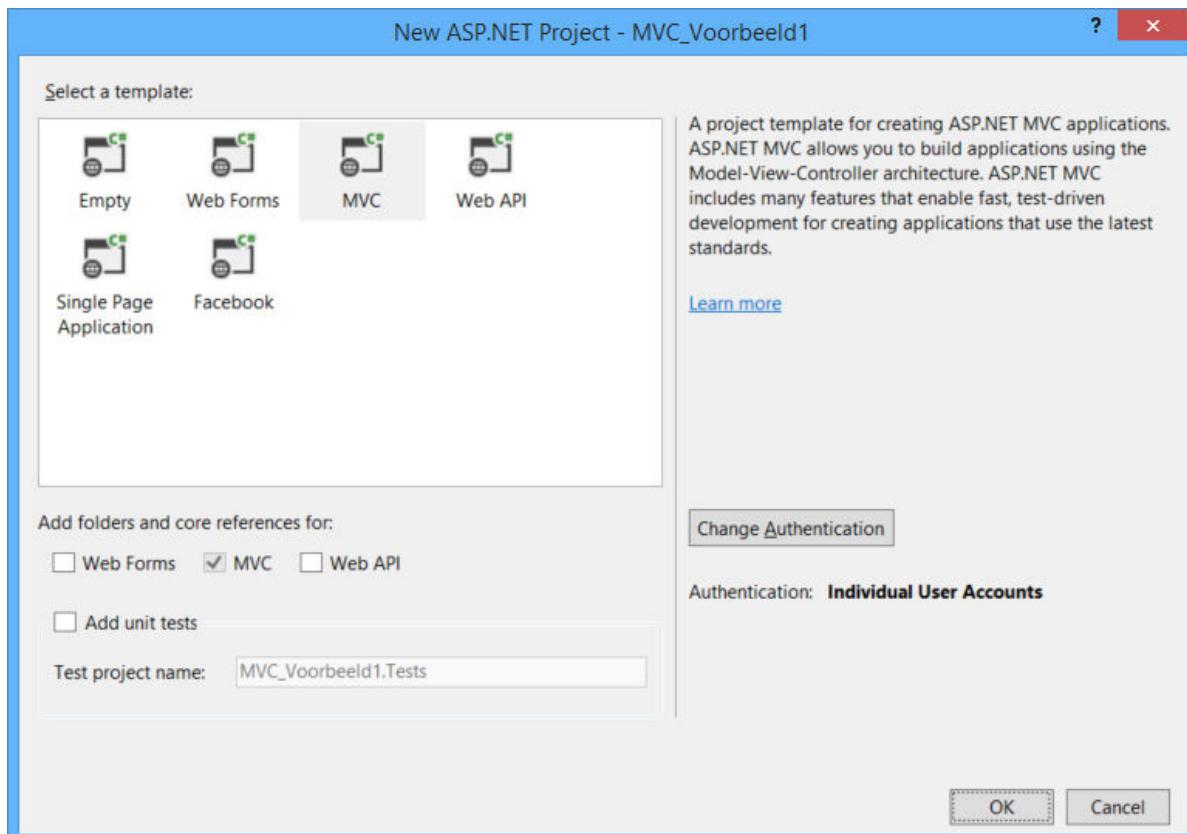


- Start Visual Studio en maak een nieuwe ASP.NET Web Application. Geef het project een zinvolle naam, bijvoorbeeld *MVC_Voorbeeld1*, kies een geschikte locatie en klik op OK.

We krijgen nu een dialoogvenster te zien waarin we een project template moeten kiezen.

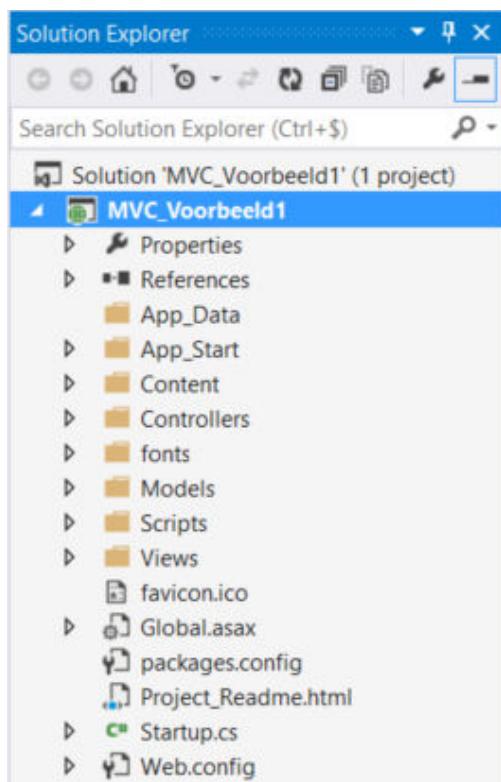
Mogelijkheden zijn : een leeg project, een Web Forms applicatie, een MVC web applicatie, een Web API project, een Single Page Application of een Facebook applicatie. Aan de rechterkant van het dialoogvenster lees je de omschrijving van deze mogelijke templates. Afhankelijk van de gekozen template worden in het dialoogvenster bepaalde opties in- of uitgeschakeld. Je kan namelijk een aantal technieken combineren. Zo kan je bijvoorbeeld MVC combineren met Web Forms. Voor de duidelijkheid gaan we dit in deze cursus niet doen.

- Kies de template *MVC* door op het icoontje MVC te klikken, de opties eronder laat je zoals ze standaard zijn aangevinkt.
- Klik op OK.



3.2 Inhoud van het MVC project

Visual Studio maakt nu een projectstructuur aan die je meteen kan bekijken in het Solution Explorer venster dat standaard aan de rechterkant te vinden is.



In de projectstructuur vinden we mappen voor onze **Models**, **Views** en **Controllers**.

Verder zie je ook mappen voor javascripts bestanden (Scripts), CSS-files (Content) en fonts.

In App_Start vind je o.a. de codefile RouteConfig.cs die we daarnet hebben besproken. De App_Data folder kan je gebruiken om een database in op te slaan.

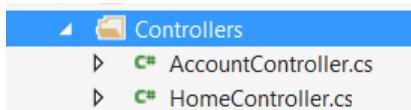
De file Global.asax heeft een bijhorende codefile waarin je o.a. code kan opnemen die uitgevoerd wordt wanneer de webapplicatie wordt opgestart op de webserver.

Web.config is een configuratiebestand.

Het project opent standaard in VS met een readme html-file (*Project_Readme.html*) die louter informatief is voor de ontwikkelaar.

3.3 De controllers

Zoals we in het vorige hoofdstuk hebben gezien begint alles bij een controller. De folder Controllers in ons project bevat reeds twee controllers : HomeController en AccountController.



De HomeController is de controller die wordt aangesproken wanneer we surfen naar de root van de applicatie. Dit staat zo standaard ingesteld in de routingtabel. De AccountController bevat een aantal actions die allemaal te maken hebben met een login-systeem dat je gratis en voor niks meegeleverd krijgt bij een nieuw MVC project.

In de praktijk maken we voor elke entiteit uit ons project een controller. Zo zullen we bijvoorbeeld een LeverancierController, een ArtikelController of een FactuurController maken.

3.4 De views

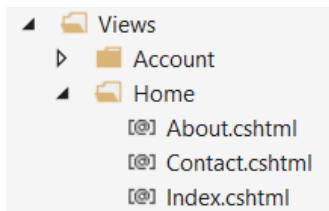
Wanneer je in de Solution Explorer dubbelklikt op het bestand HomeController.cs, dan zie je onderstaande code :

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

    public ActionResult About()
    {
        ViewBag.Message = "Your application description page.";
        return View();
    }

    public ActionResult Contact()
    {
        ViewBag.Message = "Your contact page.";
        return View();
    }
}
```

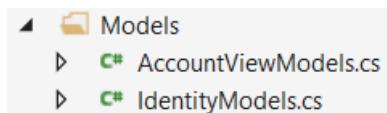
De HomeController die je aangeleverd krijgt bevat drie actions : Index(), About() en Contact(). Elk van deze actions leiden tot een view. Deze views vinden we terug in de folder Views. Die folder bevat een aantal subfolders waaronder de folder Home. Het is in deze folder dat we drie .cshtml-bestanden terugvinden : één voor elke view die hoort bij de actions uit de HomeController.



Wanneer je dubbelklikt op bijvoorbeeld de view Index.cshtml dan zie je dat dit bestand voornamelijk html-code bevat. Wie een beetje bekend is met Bootstrap zal enkel CSS-classes herkennen zoals o.a. 'jumbotron', 'col-md-4', 'btn' en 'btn-default'.

3.5 De models

De folder Models bevat reeds twee cs-bestanden : AccountViewModels.cs en IdentityModels.cs.

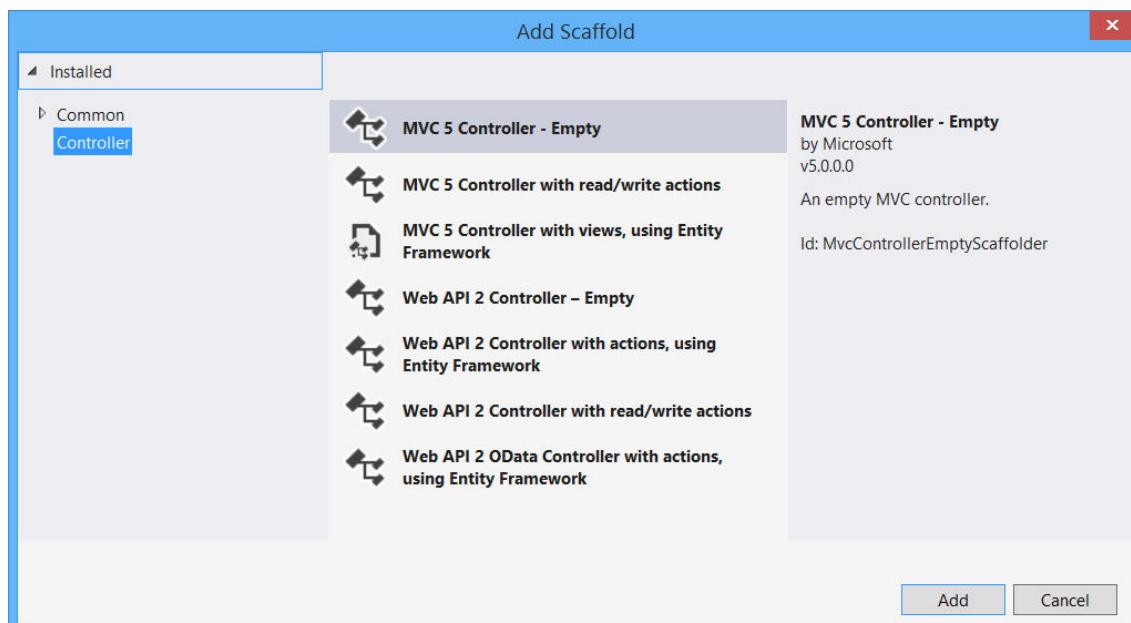


Deze bestanden worden gebruikt in de AccountController en bevatten een aantal classes voor ViewModels. In de WPF-cursus heb je wellicht al gewerkt met ViewModels.

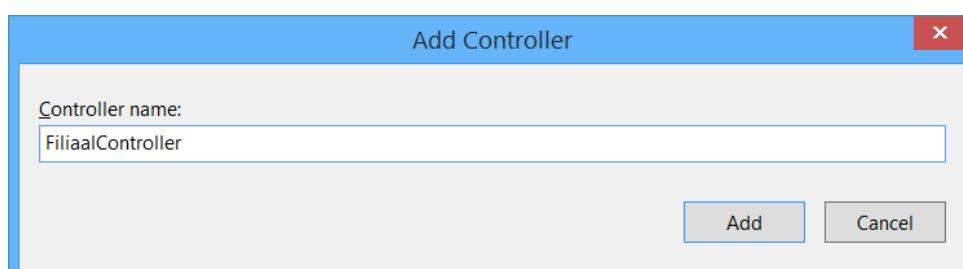
3.6 Zelf een controller toevoegen : de FiliaalController

Om de opbouw van een controller en bijhorende view(s) beter te begrijpen maken we zelf een nieuwe controller aan, de FiliaalController.

- Klik in de Solution Explorer met de rechtermuistoets op de map *Controllers* en kies *Add-Controller...*



- Kies voor een lege controller door te kiezen voor *MVC 5 Controller - Empty*.
- Klik op *Add*.
- Noem de nieuwe controller *FiliaalController* en klik op *Add*.



Het is aangewezen de naam van de controller te laten eindigen op ...Controller. Dit is niet verplicht maar je zal zo dadelijk zien dat je het je hiermee een stuk makkelijker maakt.

Visual Studio maakt nu een class aan met de naam *FiliaalController* die erft van Controller met daarin één method : *Index()*. Deze method wordt ook wel een action genoemd en heeft als resultaat een object van het type *ActionResult*. In een volgend hoofdstuk zal je zien dat actions ook andere resulttypes kunnen hebben.

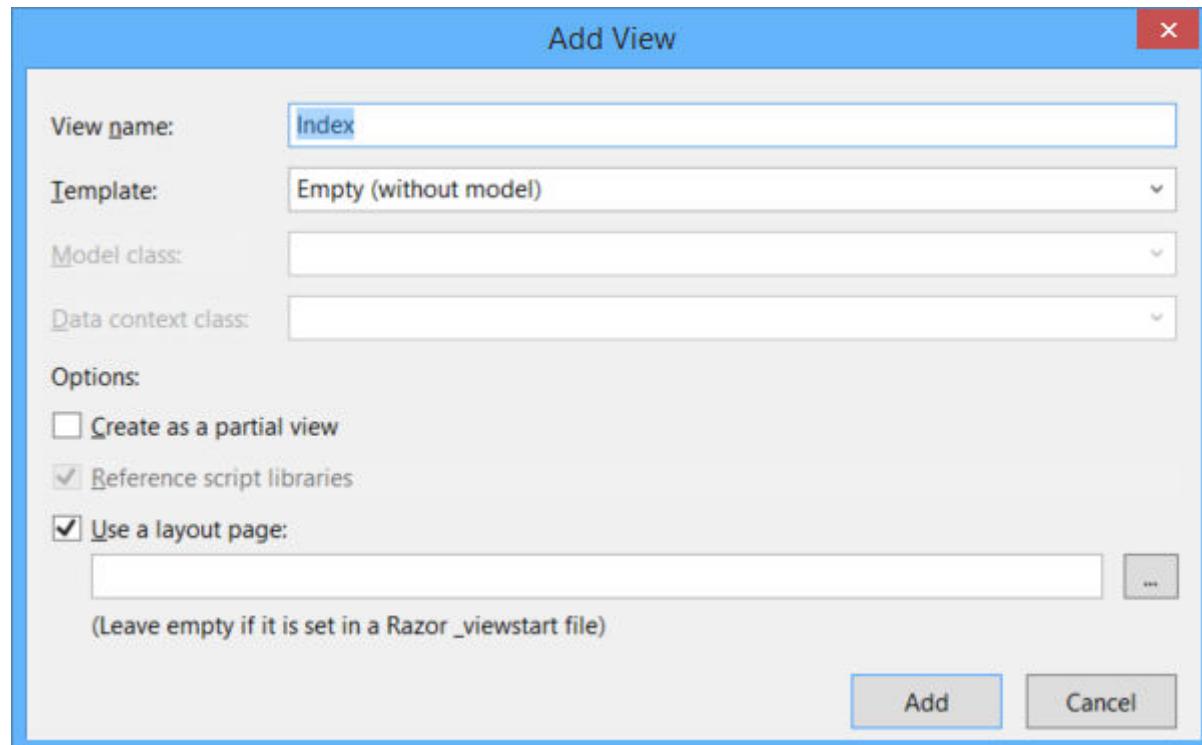
```
public class FiliaalController : Controller
{
    //
    // GET: /Filiaal/

    public ActionResult Index()
    {
        return View();
    }
}
```

Op dit moment beschikken we nog niet over een View of webpagina die het resultaat kan zijn van deze action. Dus voegen we een dergelijke View toe :

- Klik met de rechtermuistoets in de source van de *Index()*-method en kies *Add View...*

Het volgende dialoogvenster verschijnt :



We behouden alle standaardwaarden behalve bij *Use a layout page*:

- Vink de optie *Use a layout page*: **uit** en klik dan op *Add*.

Het resultaat is een pagina *Index.cshtml*. Bovenaan zie je een @-teken gevolgd door code tussen accolades, gevolgd door vertrouwde html-code.

```

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <div>
    </div>
</body>
</html>

```

- Tussen de `<div>`-tags tik je de html-code `<h1>Filialen</h1>`
- Probeer de webapplicatie uit door bijvoorbeeld F5 te drukken.
- Normaal wordt er nu `localhost:<poortnummer>/Filial/I`ndex weergegeven in de browser.
Mocht dit niet het geval zijn pas dan de url aan.

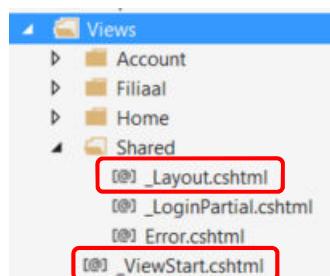
3.7 Templates

In het bovenstaande voorbeeld hebben we gevraagd om de optie *Use a layout page*: uit te vinken. We deden dit opdat je duidelijk zou zien dat het resultaat van de controller-action `Index()` een html-pagina is.

De pagina's die we zullen genereren bevatten dikwijls dezelfde begin- en eindcode. Het is dus aangewezen die stukjes code in een zogenaamde *layout page* te zetten. En dat is precies wat Visual Studio voorstelt met de optie '*Use a layout page*' in het venster *Add View*.

In de projectfolder *Views* vind je een bestand `_ViewStart.cshtml` en in de folder *Views/Shared* een bestand `_Layout.cshtml`. Als je die twee documenten eens opent zal je zien dat daarin de html-bouwstenen zitten die je ook terugvond in ons eerste voorbeeld.

Beide bestanden kan je naar eigen behoeftte aanpassen. Dit leer je in het hoofdstuk *Layouts* vanaf pagina 39.



3.8 Extra actions in de filiaalcontroller

Standaard krijg je bij het maken van een Controller een Index()-method. We voegen twee extra actions toe aan de FiliaalController : Show() en Read(). Via de method Show() gaan we alle filialen tonen, met de method Read() één enkel filiaal.

- Stop de uitvoering van de webapplicatie door Shift+F5 te drukken of door op de Stop Debugging-knop  te klikken in de werkbalk.
- Voeg in *FiliaalController.cs* onderstaande code toe :

```
public ActionResult Show() {
    return View();
}

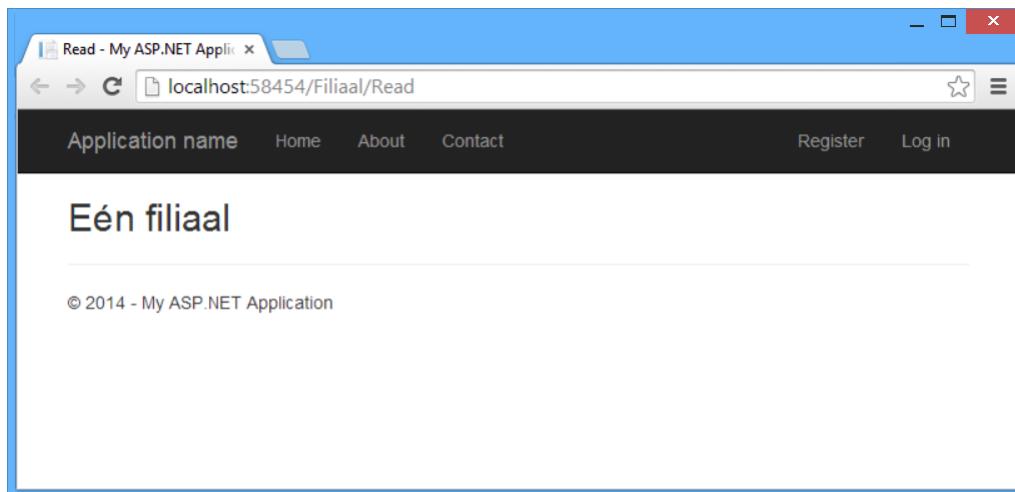
public ActionResult Read(int? id) {

    //lees in de database het filiaal waarvan de id gelijk is aan de waarde
    //van de parameter id
    return View();
}
```

- Klik met de rechtermuistoets in de method *Show()* en kies *Add View...*
- Je vinkt de optie *Use a layout page* terug aan.
- Klik op *Add*.
- Klik ook in de method *Read(int? id)* met de rechtermuistoets en kies opnieuw *Add View...*
- Kies opnieuw de standaardkeuzes en klik op *Add*.

Als resultaat krijg je twee vrij gelijkaardige sourcefiles : *Read.cshtml* en *Show.cshtml*.

- In de sourcefile *Show.cshtml* vervang je tussen de <h2>-tags de tekst 'Show' door 'Alle filialen'.
- In *Read.cshtml* tik je tussen de <h2>-tags 'Eén filiaal'.
- Start nu de webapplicatie. Je komt normaal gezien terecht op de pagina die je laatst hebt aangepast.

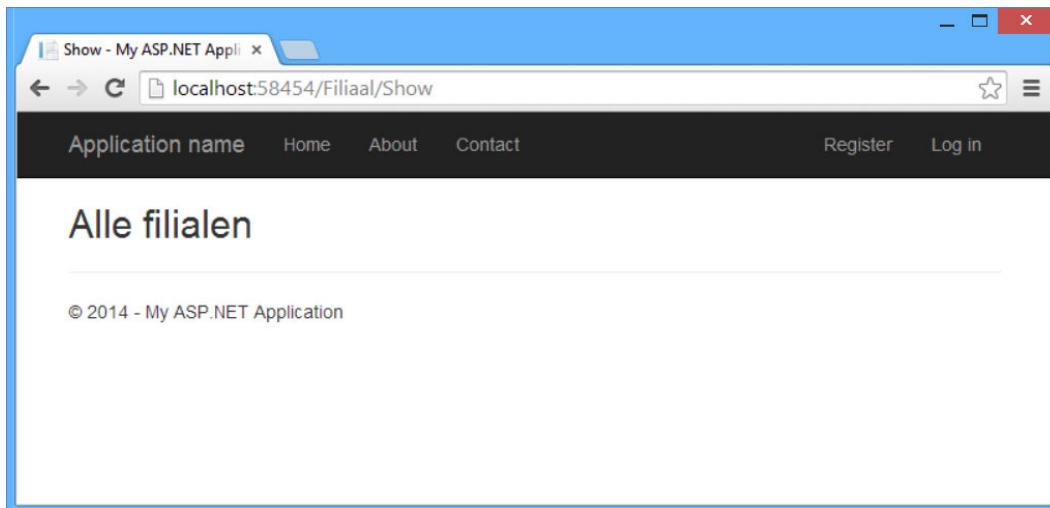


Je komt terecht in de Read-action van de FiliaalController. Een heel intuïtief ogende URL dus. Op een gelijkaardige manier kan je surfen naar .../Filiaal/Read/2 om bijvoorbeeld de gegevens van

een welbepaald filiaal, hier het tweede, te bekijken. Om effectief de gegevens van een tweede filiaal te tonen moeten we wel nog code toevoegen in de Read-action maar dat is voor later.

Je krijgt ook een menu bovenaan en login-opties rechts bovenaan. De html-code die hiervoor zorgt vind je terug in _Layout.cshtml.

- Wijzig nu de URL tot .../Filiaal>Show en bekijk het resultaat.



Je krijgt nu de webpagina voor alle filialen te zien.

3.9 Samenvatting

- Controllers zijn classes en bevatten actions geschreven in C#-code.
- Het resultaat van een action is een view. Deze bevat html-code. Wanneer je kiest voor *Use a layout page* dan wordt deze view samengesteld uit meerdere cshtml-bestanden. Op die manier krijgen alle pagina's van jouw applicatie dezelfde look-and-feel en moet je een aanpassing meestal slechts éénmaal doorvoeren i.p.v. op elke pagina apart.
- Je roept een action op door te browsen naar een url van de vorm **controller/action/id**. Bijvoorbeeld .../Home/Index of .../Filiaal/Read/2.

4 Meer over controllers

Het is ondertussen duidelijk dat de controllers de draaischijven zijn van onze MVC webapplicatie. In dit hoofdstuk bekijken we een controller in detail.

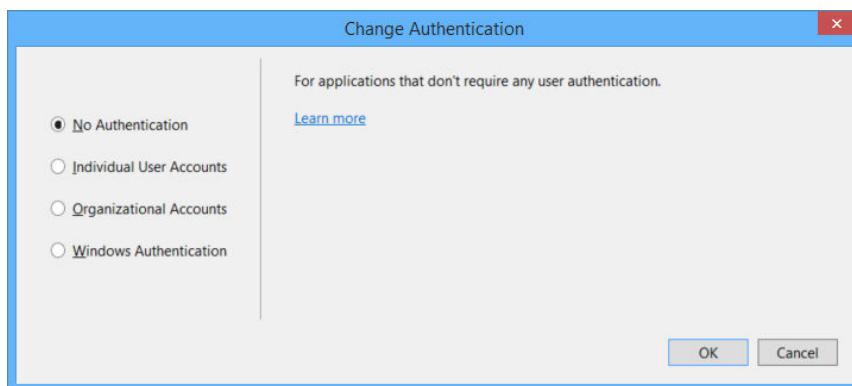
4.1 De Controller class

In het vorige voorbeeld kon je in de source van de HomeController zien dat deze is afgeleid van de class Controller. De meeste ontwikkelaars maken per entity class een controller waarvan de naam is samengesteld uit de naam van de entity (in enkelvoud), gevolgd door 'Controller'. Dus naast een HomeController hebben we bijvoorbeeld een WerknemerController of PersoneelController. Sommige ontwikkelaars gebruiken liever een meervoudsvorm zoals in FilialenController. De keuze is vrij.

Voor elke binnenkomende request maakt MVC een instance aan van de controller die hoort bij de opgegeven URL. Deze instance 'leeft' zolang de verwerking van de request duurt.

Voor dit hoofdstuk maken we een nieuwe *ASP.NET MVC Web Application* aan. We gebruiken opnieuw de MVC template.

- Sluit de huidige solution en kies *FILE-New-Project...*.
- Kies een *ASP.NET Web Application* en geef deze de naam *MVC_Voorbeeld2*.
- Kies de template *MVC* maar klik rechts op de knop *Change Authentication* en selecteer *No Authentication*. We hebben immers geen inlogssysteem nodig.



- Klik op OK.
- Voeg er nu een (empty) MVC 5 Controller aan toe met de naam *WerknemerController*.

4.2 Controller methods

De WerknemerController bevat standaard een Index-method. Het is een publieke method met als resultaattype een *ActionResult* en kan via een request naar .../Werknemer/Index opgeroepen worden. Dit soort methods noemen we ook wel een action.

```
public ActionResult Index()
{
    return View();
}
```

Publieke methods die niet bedoeld zijn om via een request op te roepen laten we voorafgaan door de annotation [NonAction].

```
[NonAction]
public void GeenAction()
{
    ...
}
```

- Voeg bovenstaande action toe aan de WerknemerController (zonder de puntjes uiteraard).
- Start het project op en probeer nu eens te browsen naar .../werknemer/geenaction.

Je krijgt de fout '*The resource cannot be found*' en dat is normaal want de method is geen action. Ook als je browsed naar .../werknemer/index krijg je een fout : '*The view 'Index' or its master was not found*' maar dit komt omdat je nog geen view hebt gemaakt voor deze action.

Aan de hand van de annotations [HttpGet], [HttpPost], [HttpPut] en [HttpDelete] kan je de gewenste Http-method aangeven. Daarover later meer. In het onderstaande voorbeeld wijzigen we de weddes dus gebruiken we een HttpPost-method.

```
[HttpPost]
public ActionResult VerdubbelDeWeddes()
{
    return View();
}
```

Verderop in deze cursus bespreken we het gebruik van de annotations [HttpGet] en [HttpPost] nog meer in detail. Je hoeft het nu nog niet uit te proberen.

Als een method parameters heeft, dan worden die automatisch gevuld met path variabelen uit de URL die dezelfde naam hebben als de method-parameters.

```
public ActionResult Read(int? id)
{
    ...
}
```

In bovenstaand voorbeeld wordt bij een request .../Werknemer/Read/2 de parameter *id* gevuld met de waarde 2.

Een methodparameter kan ook gevuld worden met een request-parameter. De request .../Werknemer>ShowAll?paginanr=3 zal in onderstaande controllermethod de parameter *paginaNr* opvullen met de waarde 3.

```
public ActionResult ShowAll(int? paginaNr)
{
    ...
}
```

In bovenstaande voorbeelden zijn de methodparameters telkens nullable. Indien de method-parameter een gewone int zou zijn en we surfen naar .../Werknemer/Read dan krijgt de parameter geen waarde en krijgen we een fout.

4.3 Returntypes

In de bovenstaande voorbeelden hadden de controllermethods telkens als returntype ActionResult. ActionResult is een abstracte class. De eigenlijke returnwaarde van de methods zullen dus een instance van een afgeleide class zijn van ActionResult.

Je kan zo een instance zelf aanmaken maar meestal gebruik je een helpermethod van de Controller class om dit te doen. De method View() is zo een helpermethod en die levert je bijvoorbeeld een instance op van de class ViewResult.

Dus dit :

```
public ActionResult Index()
{
    return View();
}
```

is een kortere schrijfwijze voor dit :

```
public ActionResult Index()
{
    var viewResult = new ViewResult();
    return viewResult;
}
```

We overlopen nu even de diverse mogelijke returntypes.

ViewResult

Een ViewResult zullen we vaak gebruiken en heb je al in actie gezien. Het bevat de response html-code. Deze code wordt aangemaakt door een bijhorend .cshtml-bestand in de map Views of in een submap van Views. Hoe deze view wordt teruggevonden lees je in de volgende paragraaf 4.4.

ContentResult

Bij een ContentResult bevat de response een stuk tekst (platte tekst, XML,...) die in de controller zelf wordt aangemaakt. Ook hier kunnen we opnieuw een korte schrijfwijze gebruiken door de helper-method Content() te gebruiken. Voorbeeld :

```
return Content("Kiekeboe");
```

FileResult

Voor een FileResult kan je de helper-method File() gebruiken. De response bevat dan (binaire) data uit een bestand op de server of een stream gemaakt op de server. Voorbeeld :

```
return File(@"e:\etiketten\Cezarken.jpg", "image/jpg");
```

JsonResult, JavaScriptResult

Bij een JsonResult bevat de response JSON-code, bij een JavaScriptResult is dit JavaScript-code. Voor de eerste kan je de method Json() gebruiken, voor de andere JavaScript().

RedirectResult

Hier is de response een Redirect-opdracht. Dit gebeurt ofwel via de helper-method `Redirect()` en een URL ofwel via een helper-method `RedirectToRoute()` en een route. Voorbeeld :

```
public ActionResult VerdubbelDeWeddes()
{
    //update in de database
    //....
    return Redirect("~/Werknemer/WeddesAangepast");
}

public ActionResult WeddesAangepast() {
    return View();
}
```

In bovenstaand voorbeeld zie je een method `VerdubbelDeWeddes()` waarin de weddes worden aangepast. Er wordt een `RedirectResult` gereturnd en de URL is `"~/Werknemer/WeddesAangepast"`.

HttpNotFoundResult, HttpUnauthorizedResult en HttpStatusCodeResult

Een `HttpNotFoundResult` levert een response op met HTTP-status 404. Je kan de helper-method `HttpNotFound()` gebruiken. Een `HttpUnauthorizedResult` levert een response op met HTTP-status 401. Je kan de helper-method `HttpUnauthorizedResult()` gebruiken. Bij een `HttpStatusCodeResult` bevat de response een eigen gekozen http-status. Je gebruikt een `HttpStatusCodeResult()` helper method.

4.4 ViewResult

Omdat we meestal een `ViewResult` zullen gebruiken staan we er even wat langer bij stil.

Wanneer het resultaat een `ViewResult` is, dan zoekt de helpermethod `View()` een cshtml-bestand met dezelfde naam als de controllermethod. Dus als jouw controllermethod de naam `Read` heeft dan wordt er gezocht naar een cshtml-bestand met de naam `Read.cshtml`. Standaard wordt er gezocht naar dit bestand in een submap van de map `Views`, die de naam van de controller draagt zonder het woord 'controller'. Dus in ons geval in de submap `Werknemer` van de map `Views`.

Wanneer het nodige cshtml-bestand niet gevonden wordt in de beoogde submap van de map `Views` dan wordt er in de map `Shared` gezocht naar een cshtml-bestand met dezelfde naam als de huidige Controller method.

Volledigheidshalve moeten we hier ook vermelden dat indien er geen cshtml-bestand wordt gevonden er ook wordt gezocht naar gelijknamige .aspx, .ascx en .vbhtml bestanden.

Laten we nu bij wijze van voorbeeld een dergelijke `ViewResult` aanmaken.

- Klik met de rechtermuistoets in de source van de controllermethod `Index()`.
- In het snelmenu dat verschijnt klik je op `Add View...`

Visual Studio stelt als naam voor de View dezelfde naam als de method voor. Je kan de voorgestelde naam `Index` kiezen maar ook een zelfgekozen naam geven aan jouw `ViewResult`.

- Geef de view de naam `AndereNaam` en klik op `Add`.

De kous is nu nog niet af. Indien je nu zou browsen naar .../Werknemer/Index dan zou MVC op zoek gaan naar een view met de naam Index.cshtml. Om aan te geven dat de view AndereNaam.cshtml moet getoond worden, geef je de naam mee als stringparameter van de helpermethod View().

- Pas de code van de Index()-method als volgt aan :

```
public ActionResult Index()
{
    return View("AndereNaam");
}
```

- Surf nu naar .../Werknemer/Index en bekijk het resultaat. Je ziet de view *AndereNaam*.

Tip

Wil je, bijvoorbeeld tijdens een testfase, bij het starten van de webapplicatie meteen een specifieke action laten uitvoeren dan kan je dit instellen in de properties van jouw project.

- Kies in het Visual Studio menu voor PROJECT- *Projectnaam* Properties.
- Selecteer de rubriek Web.
- Kies onder Start Action voor *Specific Page*.
- Vul de URL in waarnaar gebrowset moet worden, bijvoorbeeld *Werknemer/Index*.

Opgepast : vul bij *Specific Page* geen bestandsnaam in die hoort bij een View zoals bijvoorbeeld Index.cshtml. Een view is geen doel voor een browser maar wel het resultaat van een actionmethod !

4.5 Samenvatting

- Controllers bevatten methods, ook wel actions genoemd.
- Het resultaat van een action is meestal een view, een soort html-pagina, maar kan ook een redirect zijn naar een andere action.
- Wanneer het resultaat van de action een view is dan wordt de view gereturnd met een helperfunctie View(). Wanneer deze functie geen parameter heeft dan wordt er naar een view gezocht die dezelfde naam heeft als de actionmethod. Wil je verwijzen naar een view met een andere naam dan geef je de naam van die view mee als parameter.

5 Razor

We begrijpen nu al iets beter hoe een controller in elkaar zit. Maar voorlopig staat er nog niet veel in de Views. In dit hoofdstuk leer je hoe je dynamische content opneemt tussen statische tekst.

5.1 De ASPX en Razor View Engines

Een webpagina is meestal een mix van vaste tekst of gegevens en veranderlijke data, meestal afkomstig uit één of andere databasetabel. De veranderlijke data of dynamic content voeg je toe in een view via code. Wanneer een gebruiker via een request een pagina opvraagt, dan wordt de code voor de dynamische content op de server verwerkt en geïnterpreteerd door een View Engine. Deze produceert dan de nodige HTML-code. Op de webserver bevindt een view dus een mix van vaste tekst en code, het resultaat op de browser is het resultaat ervan.

Er bestaan verschillende View Engines. Er is de ASPX View Engine en de nieuwere Razor View Engine. De ASPX View Engine bestond reeds in de oudere Web Forms-technologie. Sinds VS2012 is er een nieuwere View Engine, namelijk de Razor View Engine. Elk van beide View Engines interpreert een code met een specifieke syntax. De code voor de Razor View Engine, Razor-code of kortweg Razor, gebruikt een syntax die koper is en intuïtiever oogt dan de oudere ASPX-syntax.

Om het verschil aan te tonen geven we van beide een klein voorbeeldje. Aan jou om vast te stellen of de razorsyntax inderdaad intuïtiever is. In onderstaande voorbeelden wordt na 17u en voor 9u de tekst ‘Relax...’ afgebeeld en anders ‘Work!’.

ASPX (of Web Forms View Engine)

```
<% if (DateTime.Now.Hour > 17 || DateTime.Now.Hour < 9)
    { %>
    <p>Relax...</p>
<% } else { %>
    <p>Work!</p>
<% } %>
```

Razor

```
@if (DateTime.Now.Hour > 17 || DateTime.Now.Hour < 9) {
    <p>Relax...</p>
} else {
    <p>Work!</p>
}
```

In de ASPX-code moet alle C#-code binnen <% en %> staan. Op deze manier is dergelijke code al snel onoverzichtelijk. Een <% of %> is bovendien al snel eens vergeten. De Razor View Engine is slimmer. Je plaatst een @-teken aan het begin van je code. Tussendoor plaats je waar nodig gewoon html-code. Razor is slim genoeg om te merken dat dit html-code is en geen C#.

5.2 Razor syntax

We geven hier enkele voorbeelden van razor syntax die je kan uitproberen in het huidige MVC-project, bijvoorbeeld in de view *AndereNaam.cshtml*.

5.2.1 Het @-teken

Om een lijn Razor-code te laten interpreteren en omzetten naar html plaats je een @-teken vlak vóór de lijn. Ook als je de waarde van een variabele of een expressie wil weergeven zet je er een @-teken voor.

een berekening : `<h1>@(1+2*5+0.5)</h1>`

een for-lus :

```
@for(var teller = 0;teller < 10;teller++)
{
    @teller<br />
}
```

Je plaatst meerdere codelijnen na een @-teken en tussen accolades. Omdat we tussen de `<p>`-tags de inhoud van de variabele *getal* willen tonen en niet de tekst "getal", zetten we er een @-teken voor.

een while-lus :

```
@{
    <h2>Countdown</h2>
    var getal = 10;
    while (getal > 0)
    {
        <p>@getal</p>
        getal--;
    }
}
```

- Voeg onderstaande code toe aan de view *AndereNaam.cshtml* :

Het resultaat van $(1+2*5+0.5)$ is `@(1+2*5+0.5)`

- Start het project en bekijk het resultaat door te browsen naar .../Werknemer/Index.
- Voeg nu ook onderstaande for-lus toe in de view *AndereNaam.cshtml* :

```
@for(var teller = 0;teller < 10;teller++)
{
    <br />@teller
}
```

- Bewaar de view *AndereNaam.cshtml* en refresh het resultaat in de browser. Je ziet het nieuwe resultaat zonder de webapplicatie te moeten heropstarten.
- En tenslotte proberen we ook de while-lus uit. Voeg het onderstaande toe aan de view :

```
@{
    <h2>Countdown</h2>
    var getal = 10;
    while (getal > 0) {
        <p>@getal</p>
        getal--;
    }
}
```

- Bewaar opnieuw de view en refresh de browser.

Opmerking : om wijzigingen aan te brengen in de controller moet je wél de applicatie stoppen en daarna herstarten.

5.2.2 Commentaar, character encoding, letterlijke tekst

Commentaar plaats je tussen @* en *@ :

```
@* dit is commentaar *@
```

Om het @-teken zelf te plaatsen noteer je het twee keer :

```
Dit is het @@-teken
```

Razor doet zelf automatisch aan character encoding :

de code @("Sien & Maria") stuurt Sien & Maria naar de browser.

Als je binnen accolades letterlijke tekst die niet omringd is door tags naar de browser wil sturen dan plaats je die tekst tussen <text>-tags of begin je de regel met @:

```
@if (new Random().Next(1,11) == 7)
{
    <text>Het is zeven</text>
    @: Je hebt geluk
}
else
{
    <text>Het is geen zeven</text>
    @: Een andere keer misschien
}
```

Met Random().Next(1,11) genereer je een random getal tussen 1 en 10.

- Probeer ook bovenstaande voorbeelden uit in de view.

5.3 Samenvatting

- In een view kan je dynamische content opnemen. Dit doe je door de nodige code te schrijven.
- Deze code wordt geïnterpreteerd door een view engine. Je gebruikt best code, geschreven in Razor.

5.4 Oefening

Schrijf de nodige code om de eerste getallen van de fibonacci-reeks weer te geven :

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89,...

6 Data doorgeven van controller naar view

6.1 Viewbag

Om data door te geven tussen de controller en de view kunnen we gebruik maken van een 'Viewbag', een zakje met data dus.

In de controller kunnen we gegevens in de viewbag stoppen, bijvoorbeeld :

```
ViewBag.naam = "Eddy";
```

In de view lezen we vervolgens dit gegeven als volgt : @ViewBag.naam

Bij wijze van voorbeeld maken we een method die controleert of een woord een palindroom is.

- Voeg in de *HomeController* een actionmethod *Palindroom* toe :

```
public ActionResult Palindroom(string woord) (1)
{
    char[] omgekeerd = woord.ToCharArray();
    Array.Reverse(omgekeerd);
    string achtersteveuren = new string(omgekeerd); (2)

    if (woord == achtersteveuren)
        ViewBag.palindroom = true;
    else
        ViewBag.palindroom = false; (3)

    ViewBag.ingetiktwoord = woord; (4)
    return View(); (5)
}
```

- Voeg een bijkomende View toe en vervang de aanwezige code door het volgende :

```
<h2>Palindroom</h2>
@ViewBag.ingetiktwoord<text> is </text> (6)
@if (ViewBag.palindroom) (7)
{ <text>een </text> }
else
{<text>geen </text>}
palindroom.
```

- Probeer dit nu uit door te surfen naar .../Home/Palindroom?woord=lepel

Een woordje uitleg. De URL .../Home/Palindroom?woord=lepel activeert de actionmethod *Palindroom* in de *HomeController* op basis van de default route (zie p. 14). De requestparameter *woord* wordt doorgegeven aan de parameter *woord* in de method *Palindroom* (1).

We draaien deze string via een array van chars om en stoppen het resultaat in een nieuwe string (2).

Als de oorspronkelijke string gelijk is aan de omgedraaide dan stoppen we een gegeven met de naam *palindroom* in de viewbag met waarde true. Als dat niet zo is dan heeft het de waarde false (3).

We stoppen ook het oorspronkelijke woord in de viewbag (4) en verwijzen naar een view die net zoals de method *Palindroom* heet (5).

In de view gebruiken we de razorcode `@ViewBag.ingetikwoord` om het oorspronkelijke woord weer te geven (6). De statische tekst 'is' staat tussen `<text>`-tags maar dit is niet strikt noodzakelijk. Het bevordert wel de leesbaarheid.

Het ViewBag-gegeven *palindroom* bevat ofwel true, ofwel false. Afhankelijk hiervan geven we de tekst *een* of *geen* weer (7).

6.2 Strongly typed data

De Viewbag-methode om data door te geven tussen controller en view is vrij eenvoudig maar niet echt overzichtelijk. We kunnen op een meer gestructureerde manier data doorgeven via 'strongly typed data'. Daarbij geven we een object, bijvoorbeeld van een eigen gedefinieerde class, door van Controller naar View. Dit doen we door dit object als parameter van de helpermethod *View()* mee te geven.

Als voorbeeld voeg je in de map *Models* een class *Persoon* toe. Deze geef je twee properties : Voornaam en Familienaam, beide strings.

- Klik met de rechtermuistoets op de map *Models*.
- Kies *Add-Class...*
- Geef de class de naam *Persoon* en klik op *Add*.
- Voeg onderstaande code toe aan de class :

```
public class Persoon
{
    public string Voornaam { get; set; }
    public string Familienaam { get; set; }
}
```

- Wijzig in de *HomeController* de code van de *Index()*-method als volgt :

```
return View(new Persoon { Voornaam = "Eddy", Familienaam = "Wally" });
```

- Bovenaan de source voeg je best nog een using-instructie toe :

```
using MVC_Voorbeeld2.Models;
```

- Open de bijhorende View : *Index.cshtml*

In deze view kunnen we dan de data als volgt gebruiken :

```
@model MVC_Voorbeeld2.Models.Persoon
Doorgegeven persoon : @Model.Voornaam @Model.Familienaam
```

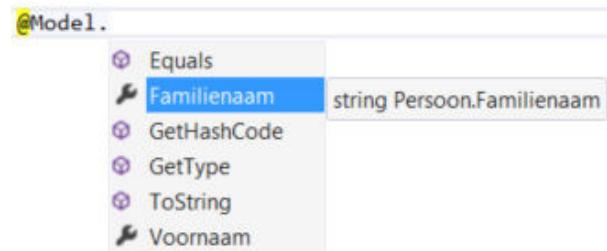
- Vervang de code in de view *Index.cshtml* door bovenstaande code en start het project.

Bovenaan vermelden we dus `@model` (met kleine beginletter), gevolgd door het type van de doorgegeven data mét vermelding van de namespace. Vervolgens kunnen we waar nodig de

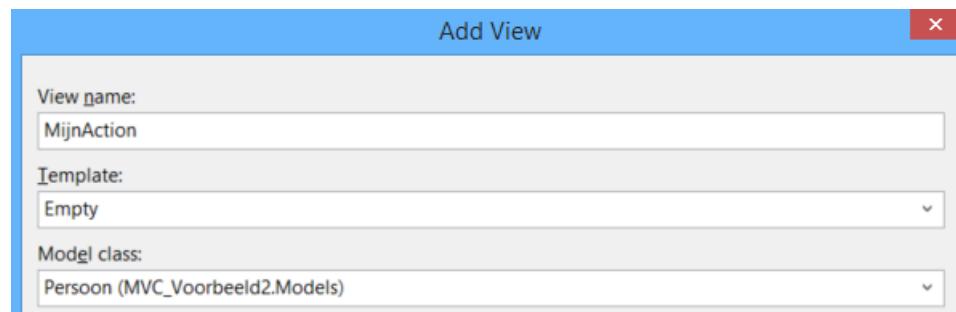
properties van het doorgegeven object in de html-code aanspreken door @Model (deze keer met hoofdletter!) te noteren, een punt en dan de property.

Willen we de properties aanspreken in C#-code dan noteren we de @ niet.

Een groot voordeel van deze werkwijze t.o.v. de ViewBag-methode is dat we hier kunnen rekenen op code-completion en ook syntax-controle. ViewBag heeft dit niet. Van zodra we na @Model de punt tikken kunnen we kiezen tussen de mogelijke properties van het doorgegeven object :



Als je bij het aanmaken van een View reeds weet welke strongly-typed data je wil doorgeven dan kan je dat al aangeven in het dialoogvenster *Add View* :



Als je in het dialoogvenster Add View kiest voor de template Empty, dan kan je een Model class intikken. Hierboven kozen we voor de class Persoon uit de namespace MVC_Voorbeeld2.Models. Visual Studio voegt dan in de view bovenaan de lijn @model MVC_Voorbeeld2.Models.Persoon toe.

We hoeven ons niet te beperken tot één enkel object om door te geven. We kunnen bijvoorbeeld ook een List doorgeven.

- Voeg in de map *Models* een *Werknemer* class toe.
- Voeg de properties Voornaam (string), Wedde (decimal) en Indienst (DateTime) toe :

```
public class Werknemer
{
    public string Voornaam { get; set; }
    public decimal Wedde { get; set; }
    public DateTime Indienst { get; set; }
}
```

- In de WerknemerController voeg je een method *AlleWerknemers()* toe :

```
public ActionResult AlleWerknemers()
{
    var werknenmers = new List<Werknemer>();
```

```

        werknemers.Add(new Werknemer { Voornaam = "Steven", Wedde = 1000,
                                      Indienst = DateTime.Today });
        werknemers.Add(new Werknemer { Voornaam = "Prosper", Wedde = 2000,
                                      Indienst = DateTime.Today.AddDays(2) });
        return View(werknemers);
    }

```

- Voeg een bijhorende view *AlleWerknemers* toe. Als template kies je Empty en als Model class het Werknemer objecttype.
- In de view wijzig je nu op de bovenste regel het Werknemer-object naar een List van Werknemer-objecten :

```
@model List<MVC_Voorbeeld2.Models.Werknemer>
```

- We kunnen nu in deze view de werknemers uitlijsten. Voeg onderstaande code toe :

```

<h2>Alle werknemers</h2>
<ul>
@foreach (var werknemer in Model)
{
    <li>@werknemer.Voornaam verdient @werknemer.Wedde</li>
}
</ul>

```

- Je kan dit nu uitproberen door naar .../Werknemer/AlleWerknemers te surfen.

Deze keer geven we dus een List van Werknemers als strongly-typed data door. Met een foreach lussen we doorheen deze lijst (Model).

Tip :

In een view kan je bovenaan een using-statement opnemen. Op deze manier moet je niet telkens de volledige naam van een object noteren in de razor-code maar volstaat het de korte naam (zonder namespace) te gebruiken.

Een voorbeeld :

```

@using MVC_Voorbeeld2.Models
@model List<Werknemer>

<h2>Alle werknemers</h2>
<ul>
    @foreach (Werknemer° werknemer in Model)
    {
        <li>@werknemer.Voornaam verdient @werknemer.Wedde</li>
    }
</ul>

```

° : we gebruiken hier een Werknemer-object als iteratievariabele, enkel en alleen om aan te tonen dat de korte naam van het object volstaat. In de praktijk gebruiken we eerder var.

6.3 Samenvatting

- Je kan heel eenvoudig data doorgeven van controller naar view via een dynamisch object, genaamd ViewBag. In de view spreek je dan de properties van de ViewBag aan.
- Wil je op een meer gestructureerde manier te werk gaan, dan gebruik je strongly typed data. Je geeft de data door als parameter van de helperfunctie View(). In de view spreek je de data aan via Model. Bovenaan de view noteer je @model, gevolgd door het type van de data die je doorgeeft.

6.4 Oefening

Maak een nieuwe ASP.NET MVC Webapplication met naam *MVCBierenApplication*.

Voeg aan de folder *Models* een class *Bier.cs* toe. Deze class heeft properties ID* (int), Naam (string) en Alcohol (float).

Maak een BierController. In een actionmethod *Index* maak je een List<Bier> aan met daarin enkele bieren. Geef deze list door naar een bijhorende view. In deze view geef je de doorgegeven bieren weer.

* We kiezen hier bewust voor een property *ID* en niet *Biernr* bijvoorbeeld omdat *ID* de defaultnaam is voor een requestparameter.

7 Code hergebruiken met Layouts, Partial Views en @helper-functies

Een heel belangrijke eigenschap van een goede website is dat alle pagina's een consequente layout hebben. Om dit te verwezenlijken kunnen we in ASP.NET MVC gebruik maken van layouts.

Een ideaal middel om code te hergebruiken op verschillende plaatsen zijn partial views. Dit zijn 'deel-views' of 'view-fragmenten' die je in verschillende views kan opnemen.

Ook in meerdere views bruikbaar zijn de @helper-functies.

7.1 Layouts

Een layout is eigenlijk een template waarvan jouw cshtml-bestanden gebruik maken. Deze templates vind je terug in de map *Views/Shared*. Je bent niet verplicht je te houden aan één enkele layout voor alle webpagina's. Een webapplicatie kan meerdere layouts hebben.

7.1.1 Standaard opbouw van een view

Vooraleer we zelf een eigen layout gaan instellen kijken we even hoe het er standaard aan toe gaat. Om een view samen te stellen wordt eerst de code uit *_ViewStart.cshtml* – deze vind je in de map *Views* – uitgevoerd. Hierin staat standaard het volgende :

```
@{  
    Layout = "~/Views/Shared/_Layout.cshtml";  
}
```

Deze code geeft aan welke layout er wordt gebruikt. Als je dus een andere layout wil gebruiken dan zal je dit hier moeten instellen.

De code uit de layout *_Layout.cshtml* uit de map *Views/Shared* ziet er bij een MVC ASP.NET webapplication als volgt uit:

```
<!DOCTYPE html>  
<html>  
<head>  
    <meta charset="utf-8" />  
    <meta name="viewport" content="width=device-width" />  
    <title>@ViewBag.Title - My ASP.NET Application</title>  
    @Styles.Render("~/Content/css")  
    @Scripts.Render("~/bundles/modernizr")  
</head>  
<body>  
    <div class="navbar navbar-inverse navbar-fixed-top">  
        ...  
    </div>  
    <div class="container body-content">  
        @RenderBody()  
        ...  
    </div>  
    @Scripts.Render("~/bundles/jquery")  
    @Scripts.Render("~/bundles/bootstrap")  
    @RenderSection("scripts", required: false)  
</body>  
</html>
```

Je ziet hierboven de standaard opbouw van een HTML5-pagina met een head en een body. In de head zie je drie zaken :

- ✓ Meta-informatie over de character-set en de breedte van de viewport.
- ✓ De titel van de pagina wordt uit de ViewBag gehaald. Wellicht heb je al gemerkt dat deze titel standaard wordt ingesteld helemaal bovenaan de views die je creëert. Bijvoorbeeld door :

```
@{
    ViewBag.Title = "Index";
}
```

- ✓ Tenslotte worden ook nog stylesheets en scripts opgenomen. Hoe dit technisch in elkaar zit lees je in de volgende paragraaf : *Bundling en minification*. Concreet wordt hier verwezen naar de style sheets *Site.css* en *bootstrap.css* en naar de scriptfile *modernizr-2.6.2.js*.

In de body vind je onder de code voor de navigationbar de code `@RenderBody()`. Deze code zorgt ervoor dat de html uit de view wordt opgenomen. Daaronder wordt opnieuw scriptcode opgenomen, meer bepaald alle scriptfiles die starten met 'jquery-' en een versienummer en ook de scriptfiles *bootstrap.js* en *respond.js*. Tenslotte wordt ook nog een section opgenomen met de naam *scripts*, indien deze bestaat.

Meer over het opnemen van scripts en over sections in de volgende paragrafen.

7.1.2 Bundling en minification

Moderne websites bevatten veel javascript- en css-bestanden. Deze bestanden zijn soms groot en de browser doet per bestand een request. Dit is tijdrovend en daar heeft MVC gelukkig iets aan gedaan. MVC bundelt enerzijds css- of script-bestanden door in één oproep alle scripts in te voegen uit een folder. Dit heet bundling. Anderzijds kunnen we minified versies van scripts ophalen. Dat zijn scripts waaruit alle white space en commentaar is weggelaten. Geen pretje (en ook niet bedoeld) om als ontwikkelaar zelf in te lezen maar wel zeer compact en dus snel op te halen door de browser.

Het bundelen van stylesheets en javascriptfiles gebeurt in het bestand *BundleConfig.cs* in de folder *App_Start*. In dat bestand zie je volgende code :

```
public static void RegisterBundles(BundleCollection bundles)
{
    bundles.Add(new ScriptBundle("~/bundles/jquery").Include(
        "~/Scripts/jquery-{version}.js");

    bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(
        "~/Scripts/jquery.validate*"));

    bundles.Add(new ScriptBundle("~/bundles/modernizr").Include(
        "~/Scripts/modernizr-*"));

    bundles.Add(new ScriptBundle("~/bundles/bootstrap").Include(
        "~/Scripts/bootstrap.js",
        "~/Scripts/respond.js"));

    bundles.Add(new StyleBundle("~/Content/css").Include(
        "~/Content/bootstrap.css",
        "~/Content/site.css"));
}
```

In bovenstaande code worden 4 scriptbundles ("~/bundles/jquery", "~/bundles/jqueryval", "~/bundles/modernizr" en "~/bundles/bootstrap") gedefinieerd en 1 stylebundle ("~/Content/css"). De namen van deze bundels geven al enigszins aan welk soort scripts ze bevatten. De bundel met naam " ~/bundles/jquery" bevat bijvoorbeeld alle .js bestanden uit de folder Scripts waarvan de bestandsnaam begint met "jquery-" en vervolgens een versienummer. De laatste bundel, de bundel voor de css-bestanden bevat dan weer de css-bestanden bootstrap.css en site.css uit de folder Content.

Eens de bundels gedefinieerd zijn kunnen ze aangesproken worden in de .cshtml-bestanden. Dat hebben we al gezien in het bestand _Layout.cshtml. Daar wordt de stylesheetsbundel ingelast met het commando `@Scripts.Render("~/bundles/jquery")`.

Wil je zelf eigen scripts en styles bundelen tot script- en stylebundles en ze inlassen in jouw .cshtml-bestanden dan weet je nu hoe dat moet : je definieert ze in *BundleConfig.cs* en je voegt ze in met het commando `@Scripts.Render("<naam van de scriptbundel>")` voor scripts en het commando `@Styles.Render("<naam van de stylebundel>")` voor styles.

Standaard is bundling en minification evenwel niet geactiveerd. Om het te activeren moet je in het bestand *Web.config* (in de root van het project) de debugmodus op *false* zetten :

```
...  
<system.web>  
...  
  <compilation debug="true" targetFramework="4.5" />  
  <httpRuntime targetFramework="4.5" />  
</system.web>  
...
```

- Start het project *MVC_Voorbeeld2* en bekijk de source van de pagina in de browser. Je zal zien dat er van enige bundling & minification geen sprake is : je ziet de gewone scripts onder de source.
- Stel in *Web.config* de debugmodus in op *false*.
- Start het project via *Run without debugging* en bekijk de source opnieuw.
- Klik eens op één van de script-links en bewonder de minified javascriptcode.

7.1.3 Een andere layout instellen

Daarnet heb je gezien dat de layout van een view bepaald wordt in *_ViewStart.cshtml*. Wil je dat jouw views er anders uitzien dan wijzig je dit dus in *_ViewStart.cshtml*. Bedenk wel dat je op deze manier het uitzicht van al jouw views verandert.

Wil je enkel voor één view een andere layout instellen dan kan je dit doen door volgende code (bovenaan) op te nemen in het cshtml-bestand :

```
@{  
    Layout = "~/Views/Shared/_AndereLayout.cshtml";  
}
```

Een voorbeeld :

- Voeg bovenstaande code toe helemaal bovenaan de view *Views/Home/Index.cshtml*.
- Maak in de map *Views/Shared* een kopie van de layout *_Layout.cshtml* en noem deze *_AndereLayout.cshtml*.
- Voeg onderaan in de footer van deze layout jouw naam toe naast het copyrightteken.
- Start het project en vergelijk de footer van *Home/Index* met deze op een andere pagina.

Je kan er ook voor kiezen geen layout te gebruiken en jouw html-pagina helemaal in de view zelf op te bouwen. Je geeft dan via de code `@{Layout = null}` aan dat er geen enkele layout moet gebruikt worden. Let in dat geval wel op dat jouw view een geldige html-pagina produceert met een volledige html-structuur !

Opmerking : eerder in deze cursus hebben we er al eens voor gekozen geen layout te gebruiken om de samenstelling van een view te demonstreren. Op pagina 20 hebben we de code `Layout = null` bovenaan de view laten invoegen door in het dialoogvenster *Add View* het vinkje naast *Use a layout page* uit te vinken.

7.1.4 Sections

Hierboven hebben we gezien dat je met de instructie `@RenderBody()` bepaalt waar in jouw layout de inhoud van de view wordt geplaatst. Ook op andere plaatsen in de layout kan je dynamisch aanpassbare stukken code opnemen via sections. De inhoud van deze sections bepaal je in de view.

In een layout geef je met `@RenderSection("naamvandesection", false)` aan dat er op die plaats de inhoud van een section met als naam *naamvandesection* komt te staan. De tweede parameter, hier met waarde *false*, geeft aan of de section verplicht is of niet. Dus als er *false* staat dan zal er gezocht worden naar een section met de opgegeven naam en als deze niet wordt gevonden wordt er niks weergegeven.

In de view noteren we dan de inhoud van de section via :

```
@section naamvandesection { inhoud van de section... }
```

In het bestand *_Layout.cshtml* hebben we reeds een voorbeeld gezien van een dergelijke instructie :

```
@RenderSection("scripts", required: false)
```

We zullen nu in een view een section opnemen met de naam *scripts* zodat we kunnen vaststellen dat deze sectie effectief wordt opgenomen. De plaats waar je deze section definieert in de view bepaalt niet waar deze wordt weergegeven. Het is de instructie `@RenderSection` die bepaalt waar de section wordt opgenomen.

- Wijzig de view *AlleWerknemers.cshtml* in de map *../Views/Werknemer* als volgt :

```
@model List<MVC_Voorbeeld2.Models.Werknemer>
<h2>Alle werknemers</h2>
<ul>
@foreach (var werknemer in Model) {
    <li>@werknemer.Voornaam verdient @werknemer.Wedde</li>
}
</ul>
@section scripts {
    <p>Hier komen nog allerlei scripts...</p>
}
```

- Navigeer nu naar/Werknemer/AlleWerknemers. Aangezien in `_Layout.cshtml` helemaal onderaan de pagina, vlak voor `</body>` de instructie `@RenderSection("scripts", required: false)` staat, zie je de tekst "Hier komen..." helemaal onderaan verschijnen.

Komen in al jouw views telkens gelijkaardige delen (sections) terug maar met een lichtjes andere inhoud dan zijn sections de aangewezen manier van werken.

7.2 Partial Views

Partial views zijn stukken html- en razorcode, opgeslagen in een cshtml-bestand, die je gebruikt als bouwstenen om jouw view op te bouwen. Een voorbeeldje : onderstaande code geeft eerst alle gegevens weer van de hoofdzetel en vervolgens de gegevens van alle filialen.

```
@{
    ViewBag.Title = "Filialen";
}
@Html.Partial("HoofdZetel")
@Html.Partial("Allefilialen")
```

De instructie `@Html.Partial("HoofdZetel")` voegt dus de volledige inhoud van het bestand `HoofdZetel.cshtml` in. Met `@Html.Partial("Allefilialen")` plaats je de inhoud van `Allefilialen.cshtml` eronder.

Indien de partial view niet wordt gevonden in dezelfde map als de view waarin deze wordt opgeroepen, dan wordt de partial view gezocht in de map `Views/Shared`.

In de partial view kan je ook gebruik maken van de viewbag van de view waarin de partial view is opgenomen en ook van het model. We proberen het eens uit :

- Maak een nieuwe ASP.NET Web Application met de naam `MVC_Voorbeeld3` en kies de MVC template zonder authentication.

We voegen zo meteen twee entities toe, een object `Filiaal` en een object `HoofdZetel`. Voor elk van deze objecten maken we een serviceclass waarin we code toevoegen die de nodige gegevens voorziet.

- In de folder `Models` maak je een class `Filiaal` :

```
public class Filiaal
{
    public int ID { get; set; }
    public string Naam { get; set; }
    public DateTime Gebouwd { get; set; }
    public decimal Waarde { get; set; }
}
```

- Maak nu ook in dezelfde folder een class `HoofdZetel` :

```
public class HoofdZetel
{
    public string Straat { get; set; }
    public string HuisNr { get; set; }
    public string PostCode { get; set; }
    public string Gemeente { get; set; }
}
```

- Maak onder de root van het project een folder *Services* en voeg aan deze folder een class *HoofdZetelService* toe :

```
using MVC_Voorbeeld3.Models;
...
public class HoofdZetelService
{
    public HoofdZetel Read() {
        return new HoofdZetel
        {
            Straat = "Keizerslaan",
            HuisNr = "11",
            PostCode = "1000",
            Gemeente = "Brussel"
        };
    }
}
```

In deze class voorzien we een method *Read()* die ons een *HoofdZetel*-object retourneert. Deze gegevens zijn nu hard gecodeerd maar zouden bijvoorbeeld uit een database kunnen komen.

- Eveneens in de folder *Services* voegen we een class *FiliaalService* toe :

```
using MVC_Voorbeeld3.Models;
...
public class FiliaalService
{
    private static Dictionary<int, Filiaal> filialen =
        new Dictionary<int, Filiaal>(); (1)

    static FiliaalService()
    {
        filialen[1] = new Filiaal { ID = 1, Naam = "Antwerpen",
            Gebouwd = new DateTime(2003, 1, 1),
            Waarde = 2000000 };
        filialen[2] = new Filiaal { ID = 2, Naam = "Dendermonde",
            Gebouwd = new DateTime(1979, 1, 1),
            Waarde = 2500000 };
        filialen[3] = new Filiaal { ID = 3, Naam = "Haasrode",
            Gebouwd = new DateTime(1976, 1, 1),
            Waarde = 1000000 };
        filialen[4] = new Filiaal { ID = 4, Naam = "Wevelgem",
            Gebouwd = new DateTime(1981, 1, 1),
            Waarde = 1600000 };
        filialen[5] = new Filiaal { ID = 5, Naam = "Genk",
            Gebouwd = new DateTime(1990, 1, 1),
            Waarde = 4000000 };
    }

    public List<Filiaal> FindAll() { (2)
        //levert enkel de values op van de dictionary
        return filialen.Values.ToList();
    }
}
```

De *FiliaalService* voorzien we van een static *Dictionary<int, Filiaal>* met de naam *filialen* (1). In de static constructor voegen we vijf *Filiaal*-objecten toe (2). Tenslotte schrijven we een method *FindAll()* die de filialenlijst retourneert (3).

We voegen nu een *FiliaalController* toe :

- Klik in de Solution Explorer met de rechtermuistoets op *Controllers* en kies *Add- Controller...*
- Geef deze de naam *FiliaalController*. In deze FiliaalController voeg je twee private variabelen toe. Voeg ook het nodige using-statement toe :

```
using MVC_Voorbeeld3.Services;
...
private FiliaalService filiaalService = new FiliaalService();
private HoofdZetelService hoofdZetelService = new HoofdZetelService();
```

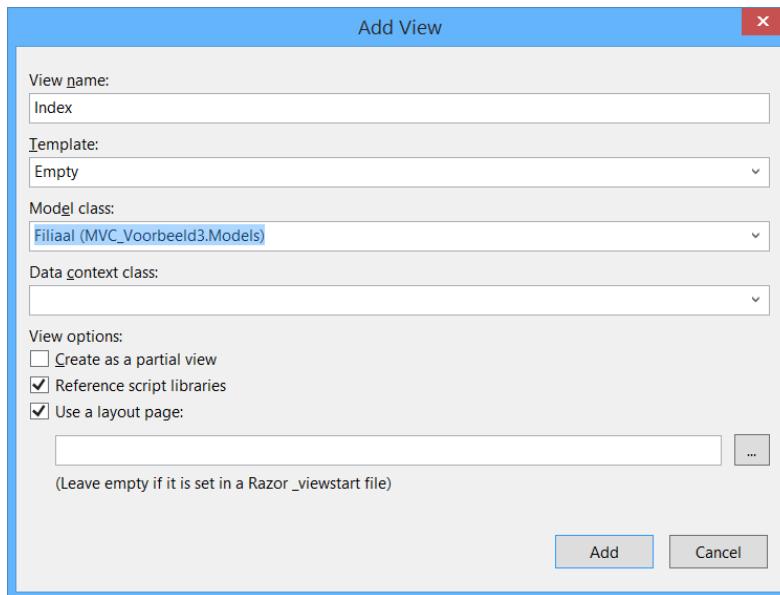
- Vervang de reeds aanwezige method *Index()* door onderstaande :

```
public ActionResult Index()
{
    var hoofdZetel = hoofdZetelService.Read(); (1)
    ViewBag.hoofdZetel = hoofdZetel; (2)
    var filialen = filiaalService.FindAll(); (3)
    return View(filialen); (4)
}
```

In *Index()* roep je de method *Read()* uit de *HoofdZetelService* op (1) en je stopt het resultaat in de *ViewBag* (2). Vervolgens roep je de method *FindAll()* uit de *FiliaalService* op (3). Dit levert je de lijst met Filialen op, dewelke je meegeeft als parameter van de *View()*-helperfunctie (4).

En dan komen we nu aan het uiteindelijke doel van dit voorbeeld : een view met daarin een partial view.

- Voeg via *Add View...* een View toe aan de *Index()*-method.
- Bij *Template* kies je *Empty* en bij *Model class* kies je de class *Filiaal*.



- Klik op *Add*. We moeten echter nog een kleine wijziging aanbrengen. We geven een *List<Filiaal>* door en geen *Filiaal*. Wijzig de eerste codelijn dus naar :

```
@model List<MVC_Voorbeeld3.Models.Filiaal>
```

In deze view zullen we twee partial views opnemen : Filialen en HoofdZetel.

- Wijzig de code in *Index.cshtml* als volgt :

```
@model List<MVC_Voorbeeld3.Models.Filiaal>
@{
    ViewBag.Title = "Index";
}
@Html.Partial("HoofdZetel")
@Html.Partial("Filialen")
```

- Klik in de Solution Explorer met de rechtermuistoets op de folder Views/Filiaal en kies Add – View... Voeg een view *HoofdZetel* toe (empty, without model). Doe dit nogmaals voor een view *Filialen*. De inhoud van de folder Views/Filiaal ziet er dan zo uit :



De code voor de view *Filialen.cshtml* :

```
<h2>Filialen</h2>
<ul>
    @foreach (var filiaal in Model) {
        <li>@filiaal.ID @filiaal.Naam @filiaal.Gebouwd @filiaal.Waarde</li>
    }
</ul>
```

En deze voor de view *HoofdZetel.cshtml* :

```
<h2>Onze hoofdzetel</h2>
<p>Straat: @ViewBag.HoofdZetel.Straat</p>
<p>Huisnr: @ViewBag.HoofdZetel.HuisNr</p>
<p>Postcode: @ViewBag.HoofdZetel.PostCode</p>
<p>Gemeente: @ViewBag.HoofdZetel.Gemeente</p>
```

Bemerk dat je in *Filialen.cshtml* het Model gebruikt ook al staat deze niet vermeld in de partial view.

Je kan dit nu uittesten door te browsen naar .../Filiaal/Index

De uitvoer :

Onze hoofdzetel

Straat: Keizerslaan
Huisnr: 11
Postcode: 1000
Gemeente: Brussel

Filialen

- 1 Antwerpen 1/01/2003 0:00:00 2000000
- 2 Dendermonde 1/01/1979 0:00:00 2500000
- 3 Haasrode 1/01/1976 0:00:00 1000000
- 4 Wevelgem 1/01/1981 0:00:00 1600000
- 5 Genk 1/01/1990 0:00:00 4000000

© 2014 - My ASP.NET Application

7.3 @helper

Een andere vorm van hergebruik van code zijn helper-functies. Je kan in een apart cshtml-bestand, bijvoorbeeld met de naam *MijnHelpers.cshtml*, een helperfunctie *MijnFunctie* schrijven. Deze functie produceert html-code en je kan de functie in jouw view oproepen via *@MijnHelpers.MijnFunctie(...)*.

Nadeel is wel dat je in een dergelijke functie geen gebruik kan maken van de viewbag of van het model. Je kan wel parameters doorgeven.

Cshtml-bestanden met helperfuncties bewaar je in een map *App_Code* onder de root van jouw project.

- Maak onder de root van jouw project een map *App_Code*. Dit kan snel door in de Solution Explorer met de rechtermuistoets te klikken op de naam van het project. Kies dan *Add – Add ASP.NET Folder – App_Code*.
- In deze map voeg je nu een MVC 5 View Page (Razor) toe (via *Add – New Item...*) met de naam *MyHelpers.cshtml*.
- In dit bestand neem je volgende code op (de reeds aanwezige code mag weg) :

```
@helper Milieuscore(DateTime datum)
{
    if (datum.Year < 1980) {
        <span>**</span>
    }
    else {
        <span>***</span>
    }
}
```

Een gebouw van vóór 1980 krijgt milieuscore 2, de andere 3.

We gebruiken deze @helper-functie nu in het bestand *FiliaLEN.cshtml* :

```
<li>
    @filiaal.ID @filiaal.Naam @MyHelpers.Milieuscore(@filiaal.Gebouwd) @filiaal.Waarde
</li>
```

Je kan dit nu uittesten door te browsen naar .../Filiaal/Index

Het resultaat :

The screenshot shows a list of filialen (branches) in a table. The columns are labeled 'Naam' (Name), 'ID', 'Gebouwd' (Built), and 'Waarde' (Value). The data is as follows:

Naam	ID	Gebouwd	Waarde
1 Antwerpen	1	***	2000000
2 Dendermonde	2	**	2500000
3 Haasrode	3	**	1000000
4 Wevelgem	4	***	1600000
5 Genk	5	***	4000000

7.4 Samenvatting

- Via layouts kan je jouw views éénzelfde look and feel geven. In een layout plaats je die content die de verschillende views gemeen hebben, zoals header, footers,...
- Met de instructie

```
@{
```

Layout = "~/<pad>/<viewnaam>.cshtml";
`}`

geef je in een view aan welke layout er moet gebruikt worden.
- Met de code `@RenderBody()` geef je in de layout aan waar de specifieke content van de view moet opgenomen worden.
- In een view kan je ook een section definiëren. In de layout bepaal je met `@RenderSection("NaamVanDeSection")` waar deze section moet staan.
- Stukken view-code die regelmatig terugkeren in meerdere views kan je apart opslaan als een Partial View. Dit is een gewoon .cshtml bestand, net zoals een gewone view. Een partial view neem je op in een view via de code `@Html.Partial("naamPartialView")`.
- Viewcode kan ook het resultaat zijn van een functie. We noemen een dergelijke functie een helperfunctie. Je slaat een helperfunctie op in een bestand in een folder App_Code. De functie roep je op met de instructie `@BestandsNaam.Functienaam(parameter)`.

7.5 Oefening

Pas in de webapplication MVCBierenApplication het bestand `_Layout.cshtml` aan zodat op alle pagina's bovenaan het logo `biertempel.png` te zien is. Onderaan alle pagina's staan de adresgegevens van Biertempel. In `_Layout.cshtml` verwijder je alle code voor de navigationbar (de eerste `<div>` binnen de body).

Tip : om te verwijzen naar een figuur gebruik je een tilde (~) en vervolgens het pad binnen de webapplicatie. Bijvoorbeeld : ``

Voorbeeld van .../Bier/Index :



The screenshot shows the Biertempel website. At the top, there is a yellow header bar with the logo 'De Biertempel Online bierenshop' and a mug icon. Below the header, the main content area has a red banner with the text '- lekker genieten van schuimend gerstenat -'. The main content lists 'Alle bieren' with two items: '15. Felix (7 %)' and '17. Roman (7,5 %)'. At the bottom of the page, there is a footer with the text '- De Biertempel - Hoppestraat 103 - 3360 Bierbeek - tel. 016123456 - email: info@biertempel.be -'.

8 Publiceren

In de praktijk testen we de werking van onze MVC Webapplications door ze te starten vanuit Visual Studio (al dan niet in debugmodus). Visual Studio publiceert de webapplication voor ons op een webserver genaamd *IIS Express*. Rechts onderaan het scherm in de system tray vind je het bijhorende icoontje .

Werkt onze webapplicatie perfect dan kunnen we deze publiceren op IIS. Is IIS nog niet geïnstalleerd op jouw toestel dan open je in het Windows *configuratiescherm* (*Control Panel*) het onderdeel *Programma's en onderdelen* (*Programs and Features*). Je kiest verder de optie *Windows-onderdelen in- of uitschakelen* (*Turn Windows features on or off*). Wanneer de optie *Internet Information Services* niet is aangevinkt (het vierkantje is leeg) dan klik je deze optie aan. Normaal worden niet alle onderdelen van IIS aangevinkt en dat is ook niet nodig. Klik daarna op OK en je start best Windows eens opnieuw op.



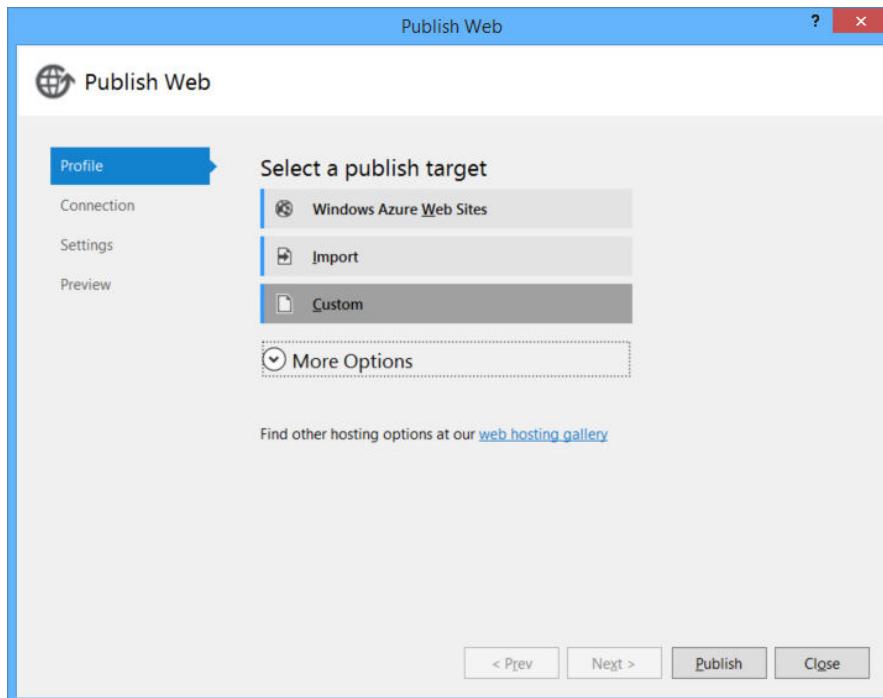
Was IIS wél al geïnstalleerd en heb je pas nadien Visual Studio geïnstalleerd dan 'kent' IIS mogelijk het nieuwste .NET Framework nog niet. In dat geval kan je best onderstaand commando uitvoeren vanaf de Developer Command Prompt for VS2013. Je vindt een shortcut naar deze command prompt in de folder C:\Program Files (x86)\Microsoft Visual Studio 12.0\Common7\Tools\Shortcuts.

Start de developer command prompt als administrator en tik het commando **aspnet_regiis.exe -i**

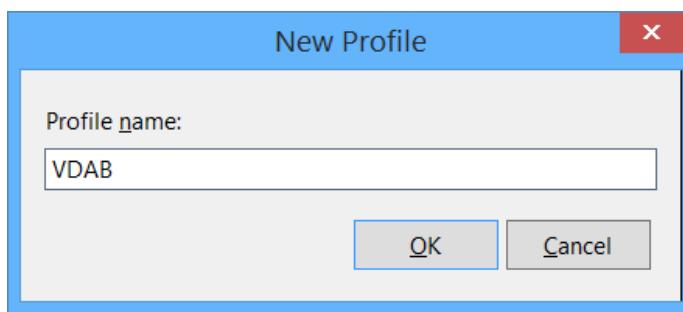
Nu zou het ASP.NET Framework juist moeten geregistreerd zijn in IIS.

Om te kunnen publiceren vanuit Visual Studio moeten we VS opstarten met Administrator rechten. In Windows 7 klik je in het startmenu met de rechtermuistoets op *Visual Studio 2013* en kies je *Run as administrator*. In het User Account Control venster klik je op *Yes*. In Windows 8 tikt je bijvoorbeeld op het startscherm de tekst ‘Visual Studio’. In het resultaatvenster klik je met de rechtermuistoets op *Visual Studio 2013* en kies je *Run as administrator*.

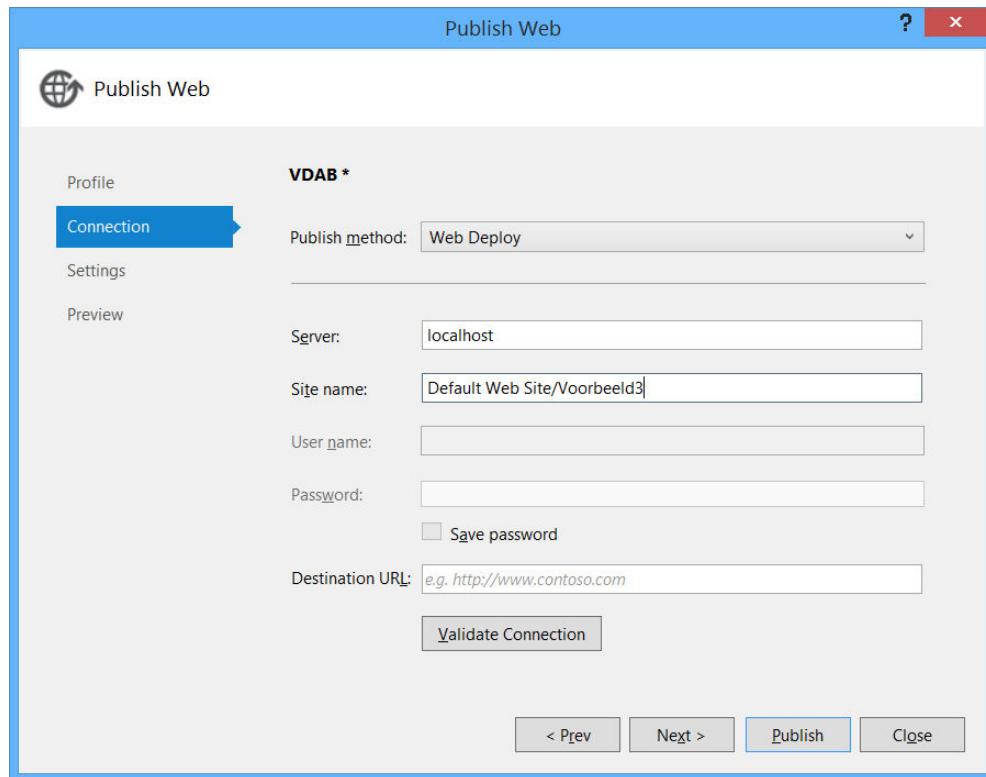
- Open opnieuw jouw project *MVC_Voorbeeld3*.
- Om te publiceren klik je in de Solution Explorer met de rechtermuistoets op de naam van het project en je kiest *Publish...* We moeten nu eerst een profiel aanmaken :



- Kies *Custom* en vul een profielnaam in :



- Klik op OK en vul in het volgende dialoogvenster de gegevens in naast Server en Site name.



Bij Server kiezen we voor localhost. Dit is de IIS op jouw toestel. Als Site name hebben we Default Web Site/Voorbeeld3 ingetikt. Standaard is Default Web Site op jouw toestel terug te vinden in de folder C:\inetpub\wwwroot. Onder die folder komen dan jouw websites elk een aparte subfolder. In dit geval is dit de subfolder Voorbeeld3.

- Klik nu op Publish. Wanneer dit gelukt is kan je het resultaat bekijken door te navigeren naar <http://localhost/Voorbeeld3/Filiaal>.



Windows 8 gebruikers lezen best even de tip op pagina 217.



Mogelijk krijg je bij het uittesten in de browser een foutbericht i.v.m. een OwinStartupAttribute. Je lost dit op door in het bestand Web.config (in de root van de web applicatie) een lijntje toe te voegen binnen de tag `<appSettings>`:

```
<appSettings>
  ...
  <add key="owin:AutomaticAppStartup" value="true"/>
</appSettings>
```

Daarna publiceer je de applicatie opnieuw en je foutbericht zou moeten verdwenen zijn.

8.1 Oefening

Publiceer nu ook de Bieren-webapplicatie. Hou rekening met de opmerkingen hierboven.

9 Opmaak van data

In een view zal je af en toe data, zoals getallen en datums, moeten voorzien van de nodige opmaak. Je kan dit op twee manieren doen. Ofwel doe je dit telkens opnieuw op elke plaats waar je een dergelijk gegeven weergeeft, ofwel breng je de opmaak aan bij de basis, namelijk gecentraliseerd in het model.

9.1 Opmaak in het cshtml-bestand

Je kan getallen en datums opmaken met de method `String.Format()`.

Enkele voorbeelden :

```
@(String.Format("{0:d}", Model.Indienst))  
@(String.Format("{0:#,##0.00}", Model.Wedde))
```

In het eerste voorbeeld staat de `d` in `{0:d}` voor `ShortDatePattern`. De wedde in het tweede voorbeeld wordt weergegeven met een decimaal teken en twee cijfers na de komma en een cijfergroeperingsteken tussen elke drie cijfers voor de komma.

We passen dit even toe in ons voorbeeld. In het bestand *FiliaLEN.cshtml* geven we de bouwdatum opnieuw weer maar deze keer in een ander formaat, namelijk enkel het jaartal. De waarde van het gebouw voorzien we eveneens van opmaak.

- Wijzig de code als volgt :

```
<li>@filiaal.ID @filiaal.Naam @MyHelpers.Milieuscore(@filiaal.Gebouwd)  
@{String.Format("{0:yyyy}", filiaal.Gebouwd)} @{String.Format("{0:#,##0.00}",  
filiaal.Waarde)}</li>
```

- Test dit uit door te browsen naar .../Filiaal/Index

Het resultaat :

FiliaLEN	
• 1	Antwerpen *** 2003 2.000.000,00
• 2	Dendermonde ** 1979 2.500.000,00
• 3	Haasrode ** 1976 1.000.000,00
• 4	Wevelgem *** 1981 1.600.000,00
• 5	Genk *** 1990 4.000.000,00



Windows 8 gebruikers lezen best even de tip op pagina 217.

9.2 Opmaak gecentraliseerd in het model

Omdat onze data toch meestal uit een object in het model komt, kunnen we beter meteen daar de juiste opmaak voorzien. Op die manier moeten we slechts één keer, op een centrale plaats, de correcte opmaak voorzien.

In onderstaand voorbeeld geven we in een class *Persoon* de properties *Indienst* en *Wedde* de nodige opmaak:

```
public class Persoon {
    ...
    [DisplayFormat(DataFormatString = "{0:d}")]
    public DateTime Indienst { get; set; }
    [DisplayFormat(DataFormatString = "{0:#,##0.00}")]
    public decimal Wedde { get; set; }
}
```

We kennen de standaard opmaak toe met een annotatie *DisplayFormat* en een attribuut *DataFormatString* (uit de namespace *System.ComponentModel.DataAnnotations*). Daarna kunnen we in een view deze opmaak effectief toepassen met de functie `@Html.DisplayFor(...)`.

Voorbeeldcode in een view :

```
@Html.DisplayFor(m=>m.Indienst)
@Html.DisplayFor(m=>m.Wedde)
```

In dit voorbeeld verwijst m naar Model.

We passen dit nu toe in ons Filialenvoorbeeld (MVC_Voorbeeld3) :

- In de class *Filiaal* voeg je de annotation *DisplayFormat* toe voor de properties *Gebouwd* en *Waarde* :

```
using System.ComponentModel.DataAnnotations;
...
[DisplayFormat(DataFormatString = "{0:yyyy}")]
public DateTime Gebouwd { get; set; }
[DisplayFormat(DataFormatString = "{0:#,##0.00}")]
public decimal Waarde { get; set; }
...
```

- In de partial view *Filialen.cshtml* kan je nu de opmaak toepassen. Wijzig de code als volgt :

```
@model List<MVC_Voorbeeld3.Models.Filiaal>
<h2>Filialen</h2>
<ul>
    @foreach (var filiaal in Model) {
        <li>@filiaal.ID @filiaal.Naam @MyHelpers.Milieuscore(@filiaal.Gebouwd)
            @Html.DisplayFor(m=>filiaal.Gebouwd) @Html.DisplayFor(m=>filiaal.Waarde)</li>
    }
</ul>
```

Opgepast : de functie *DisplayFor()* krijgt als parameter een lambda-functie mee. Deze werkt enkel als je bovenaan de partial view nog eens het type van het gebruikte model vermeldt. Doe je dit niet dan krijg je een foutberichtschap "...An expression tree may not contain..." .

9.3 UIHint

Met de annotation [DisplayFormat()] uit de vorige paragraaf zijn de opmaakmogelijkheden eerder beperkt. Met een andere annotation, UIHint, zijn de mogelijkheden oneindig. Je kan verschillende stukken html-code laten weergeven afhankelijk van de waarde van een model-property.

In ons voorbeeld zouden we i.p.v. de bouwdatum en een milieuscore, de leeftijd van het gebouw kunnen weergeven aan de hand van afbeeldingen van sterretjes . Twee sterretjes betekent gebouwd vóór 1980, anders drie sterretjes. Om dit niet telkens in elke view te moeten programmeren gaan we de code in één enkele cshtml-file opnemen en in het model deze manier van opmaken toepassen via een UIHint-annotation.

- In het bestand *Filiaal.cs* vervang je de annotation [DisplayFormat(DataFormatString = "{0:yyyy}")] door [UIHint("sterretjes")].
- In de folder *Views/Shared* voeg je een folder *DisplayTemplates* toe.
- In deze nieuwe map voeg je een View toe met de naam 'sterretjes'. In deze miniview komt er een parameter binnen van het type DateTime. Afhankelijk van de waarde van die parameter worden twee of drie sterretjes afgebeeld.
- Vervang de code in sterretjes.cshtml door onderstaande code :

```
@model DateTime
@{int aantal = 0;}
@if (Model.Year < 1980) {
    aantal = 2;
}
else {
    aantal = 3;
}
@for (int i = 0;i<aantal;i++)
{
    
}
```

Bemerk dat je de initialisatie van de lokale int-variabele tussen accolades moet plaatsen.

- Maak in de root van het project een folder *Images* en kopieer er de afbeelding *ster.png* in.
- Browse opnieuw naar .../Filiaal/Index om de nieuwe opmaak uit te proberen.

Het resultaat :

FiliaLEN

- 1 Antwerpen *** 2.000.000,00
- 2 Dendermonde ** 2.500.000,00
- 3 Haasrode ** 1.000.000,00
- 4 Wevelgem *** 1.600.000,00
- 5 Genk *** 4.000.000,00

9.4 Opmaak voor enumeration-type

Als laatste opmaakmogelijkheid demonstreren we even hoe je voor enumeratie-types een opmaak kan definiëren. We beginnen met een extra property *Eigenaar* toe te voegen aan een filiaal. Deze property is van een enumeratie-type met mogelijke waarden : Eigendom en Gehuurd.

- Voeg aan de folder *Models* een code file *Eigenaar.cs* toe :

```
namespace MVC_Voorbeeld3.Models {
    public enum Eigenaar {
        Eigendom,
        Gehuurd
    }
}
```

- In de class *Filiaal.cs* voeg je een extra property *Eigenaar* toe :

```
public Eigenaar Eigenaar { get; set; }
```

- In de folder *Views/Shared/DisplayTemplates* voeg je nu een view *Eigenaar.cshtml* toe :

```
@model MVC_Voorbeeld3.Models.Eigenaar
![@Model.ToString()](~/Images/@Model.ToString().png)

```

Deze view krijgt als parameter een enum-waarde die we gebruiken om de juiste afbeelding weer te geven.

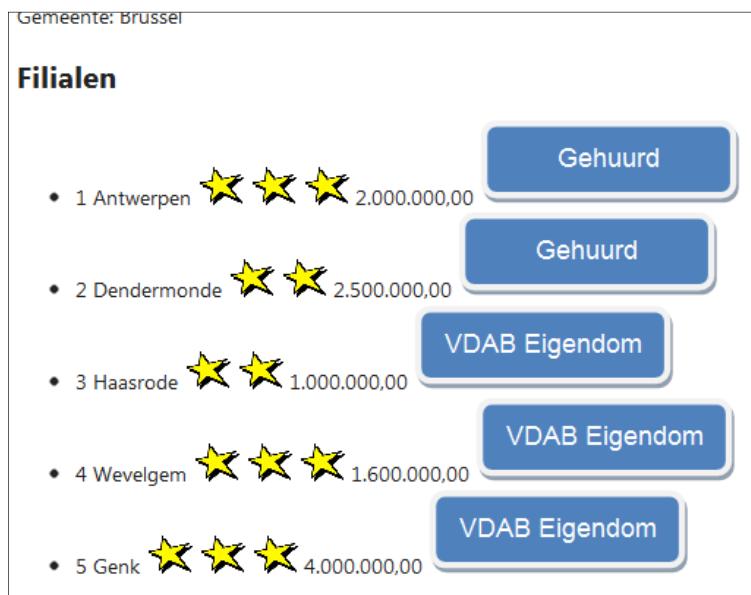
- Voeg de afbeeldingen *eigendom.png* en *gehuurd.png* toe aan de folder *Images*.
- In de view *Filiaal.cshtml* beeld je nu ook de property *Filiaal.Eigenaar* af :

```
@model List<MVC_Voorbeeld3.Models.Filiaal>
<h2>Filialen</h2>
<ul>
    @foreach (var filiaal in Model)
    {
        <li>@filiaal.ID @filiaal.Naam @Html.DisplayFor(m=>filiaal.Gebouwd)
            @Html.DisplayFor(m => filiaal.Waarde) @Html.DisplayFor(
                m => filiaal.Eigenaar)
        </li>
    }
</ul>
```

- Vooraleer je dit uitprobeert moet je in het bestand *FiliaalService.cs* wel nog waarden toekennen aan de property *Eigenaar* voor de diverse filialen. Bijvoorbeeld :

```
...
filialen[1] = new Filiaal
{
    ID = 1,
    Naam = "Antwerpen",
    Gebouwd = new DateTime(2003, 1, 1),
    Waarde = 2000000,
    Eigenaar = Eigenaar.Gehuurd
};
...
```

Het resultaat :



9.5 Samenvatting

- Met de method String.Format() kan je data voorzien van de nodige opmaak.
- Je kan opmaak ook gecentraliseerd in het model aanbrengen. Dit heeft als voordeel dat je de opmaak maar één keer moet toepassen. Wanneer de opmaak moet veranderd worden moet je het ook maar op één plaats veranderen.
- Met UIHint kan je zelfs nog een stukje verder gaan dan enkel opmaak toepassen. Je kan afhankelijk van de waarde van een property zelfs ganse stukken viewcode produceren.
- Ook voor enumerationtypes kan je specifieke opmaak definiëren.

9.6 Oefening

Pas de opmaak van een bier in de MVCBierenApplication aan :

- De ID wordt steeds met 3 digits weergegeven, dus met voorloopnullen.
- Via een UIHint geef je het alcoholpercentage weer in kleur :
 - alcoholpercentage < 2 : groen
 - alcoholpercentage <= 5 : geel
 - alcoholpercentage <= 8 : oranje
 - overige : rood

Voeg eventueel extra bieren toe zodat je de opmaak kan controleren.

 **De Biertempel**
Online bierenshop

- lekker genieten van schuimend gerstenat -

Alle bieren

001 Cnudde kriek	4,7 %
002 Liefmans goudenband	8 %
003 Sloeber	7,5 %
004 Felix kriekbier	5 %

- De Biertempel - Hoppestraat 103 - 3360 Bierbeek - tel. 016123456 - email: info@biertempel.be -

10 Gegevens over meerdere pagina's heen onthouden

In de voorbije hoofdstukken heb je geleerd hoe je gegevens kan doorgeven van een actionmethod naar de bijhorende view. Vraag je evenwel een totaal andere pagina op dan gaan deze gegevens verloren. We gebruiken immers het HTTP-protocol en dat is stateless. De 'staat' van een webpagina gaat verloren van zodra we een andere pagina opvragen.

Er zijn verschillende manieren om dit op te lossen. Ofwel bewaar je de gegevens op de browser. We hebben het dan over de alom gekende cookies. Een andere mogelijkheid is de gegevens op de server te bewaren. Daar hebben we dan de keuze of we gegevens opslaan die per session anders kunnen zijn (sessionvariabelen) of gegevens die eigen zijn aan de applicatie en die dus voor alle gebruikers van de webapplicatie dezelfde zijn (applicationvariabelen).

10.1 Cookies

Om het gebruik van cookies te demonstreren gaan we het tijdstip waarop de gebruiker het laatst onze webapplicatie heeft bezocht bewaren in een cookie.

- Wijzig de Index-action in de HomeController als volgt :

```
public ActionResult Index()
{
    string resultaat = "Dit is jouw eerste bezoek";                                (1)
    //zijn er cookies?
    if (Request.Cookies != null)                                                     (2)
    {
        //is er een cookie met de naam "user"?
        if (Request.Cookies["lastvisit"] != null)                                     (3)
        {
            //dan lezen we het tijdstip van het laatste bezoek uit de cookie
            resultaat = "Welkom terug. Jouw laatste bezoek was op " +
                Request.Cookies["lastvisit"].Value;                                    (4)
        }
        //we slaan het huidige tijdstip op als laatste bezoek
        string laatsteBezoek = DateTime.Now.ToString();
        var userCookie = new HttpCookie("lastvisit", laatsteBezoek);
        userCookie.Expires = DateTime.Now.AddDays(365);
        Response.Cookies.Add(userCookie);                                         (5)
    }
    ViewBag.Tijdstip = resultaat;                                                 (6)
    return View();
}
```

We initialiseren eerst de resultaatstring **(1)**. Daarna controleren we of er wel cookies zijn **(2)**. Is dat het geval dan zoeken we een cookie met de naam `lastvisit` **(3)**. Hebben we die gevonden dan is dit niet het eerste bezoek en kunnen we de gebruiken welkom heten en het tijdstip van zijn laatste bezoek weergeven. Daarvoor gebruiken we de Value-property van de cookie **(4)**.

Of dit nu het eerste bezoek is of niet, het huidige tijdstip moet nu opgeslagen worden als laatste bezoek. We gebruiken hiervoor een object van het type `HttpCookie`. Deze geef je een naam en een waarde. We geven een expirationdate op in de toekomst zodat de cookie ook bewaard blijft wanneer onze session eindigt. Zo wordt het een persistent cookie. Doen we dit niet dan gaat de cookie

verloren bij het afsluiten van de session (session cookie). Tenslotte slaan we de cookie op in de Response (**5**) en geven we de resultaatstring door aan de view via de ViewBag (**6**).

We passen nu de bijhorende view aan.

- Klik met de rechtermuistoets in de Index()-action en kies *Go To View*.
- Wijzig de div met class "jumbotron" zoals hieronder weergegeven.

```
<div class="jumbotron">
    <h1>Cookies</h1>
    <p class="lead">@ViewBag.Tijdstip</p>
    <p>
        <a href("~/Home/Wissen" class="btn btn-primary btn-lg">Cookie wissen &raquo;</a>
    </p>
</div>
```

De bootstrap-class *jumbotron* geeft de informatie weer in een lichtgrijs kader met lichtjes afgewonde hoeken. Via de ViewBag geven we de te tonen informatie weer. Daaronder wijzigen we de ASP.NET knop in een knop die onze cookie wist. Voorlopig doet deze knop nog niks behalve een nog niet bestaande *Wissen()*-action proberen uitvoeren. We vullen de code zo meteen verder aan.

We proberen het aanmaken van de cookie nu even uit. Om geen uren te moeten zoeken naar de cookie in de lijst wissen we eerst alle cookies in de browser. In onderstaand kader zie je hoe je in de meest courante browsers de lijst met cookies bekijkt/wist.

Internet Explorer

In IE kies je in het menu voor *Extra – Internetopties*. Op het tabblad *Algemeen* vind je onderaan in de rubriek *Browsegeschiedenis* een knop *Verwijderen...* Wanneer je die knop aanklikt dan kan je een vinkje plaatsen naast *Cookies en websitegegevens* en vervolgens klikken op *Verwijderen* om de cookies te wissen. Op het tabblad *Algemeen* is er ook een knop *Instellingen*. Via deze knop krijg je een dialoogvenster met o.a. een tabblad *Tijdelijke internetbestanden*. Daarop staat een knop *Bestanden weergeven*. Als je daarop klikt toont de Explorer je een map met o.a. bestanden waarvan de bestandsnamen beginnen met *cookie:<gebruikersnaam>@...* Dat zijn dus de IE-cookies.

In IE kan je ervoor kiezen de browsegeschiedenis automatisch te laten wissen na het afsluiten van IE. Dit kan door bij het Internetopties op het tabblad *Algemeen* het keuzevakje *Browsegeschiedenis verwijderen bij afsluiten* aan te vinken. Als je zo meteen wil controleren of er wel degelijk een cookie wordt aangemaakt kan je dit keuzevakje dus beter niet aanvinken.

Chrome

Gebruik je Chrome als browser dan kies je *Instellingen – Geavanceerde instellingen weergeven...* Bij *Privacy* klik je op de knop *Instellingen voor inhoud...* In het venster dat dan verschijnt klik je op *Alle cookies en sitegegevens...* Je ziet dan de lijst met cookies. Met de knop *Alles verwijderen* wis je alle cookies.

Firefox

In Firefox bekijk je de cookies als volgt. Kies in de *Settings voor Options*. Op het tabblad *Privacy* kies je onder *History* in de keuzelijst *Firefox will* de optie *Use custom settings for history*. Via een knop *Show Cookies* kan je dan de cookies bekijken/wissen.

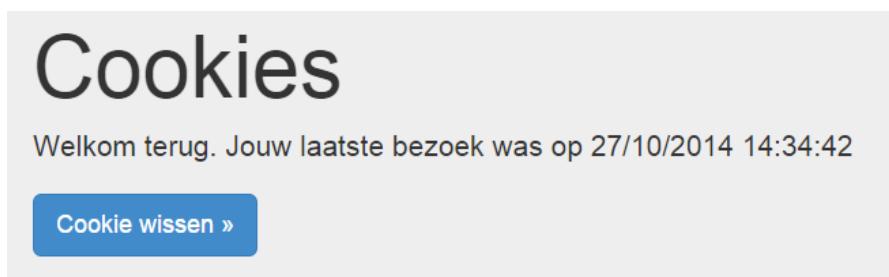
- Wis in jouw favoriete browser alle cookies.
- Start jouw webapplicatie in diezelfde browser. Normaal word je nu voor het eerst welkom geheten.



- Bekijk nu jouw cookielijst.
- Klik op *localhost* en vervolgens op *lastvisit*.
- Hieronder zie je het resultaat in Chrome.

Site	Lokaal opgeslagen gegevens	Alles verwijderen	Cookies zoeken
localhost	1 cookie, Lokale opslag	<input type="button" value="lastvisit"/>	<input type="button" value="Lokale opslag"/>
	Naam: lastvisit Inhoud: 27/10/2014 14:33:00 Domein: localhost Pad: / Verzenden voor: Elk soort verbinding Toegankelijk voor script: Ja Gemaakt: maandag 27 oktober 2014 14:33:04 Verloopt op: dinsdag 27 oktober 2015 14:33:00		
		<input type="button" value="Verwijderen"/>	

- Refresh de pagina en deze keer zou je het tijdstip van jouw vorige bezoek moeten zien :



- Sluit de browser nu volledig af alsook de webapplicatie in Visual Studio.
- Start de webapplicatie opnieuw en je zou het tijdstip van het vorige bezoek moeten zien.

We voegen nu de code toe om via de webapplicatie de cookie te wissen.

- Voeg onderstaande code toe in de *HomeController*.

```
public ActionResult Wissen()
{
    //zijn er cookies?
    if (Request.Cookies != null)
    {
        //is er een cookies met de naam "user"?
        if (Request.Cookies["lastvisit"] != null) (1)
        {
            Request.Cookies["lastvisit"].Expires = DateTime.Now.AddDays(-1);
            Response.Cookies.Add(Request.Cookies["lastvisit"]); (2)
        }
    }
    return View();
}
```

Op de index-pagina hebben we reeds een knop voorzien met het opschrift *Cookie wissen*. Deze knop activeert een *Wissen*-actionmethod. In die method controleren we eerst of er wel cookies zijn en als dat zo is of er een cookie bestaat met de naam 'lastvisit' **(1)**. Een cookie wissen we door explicet de Expires-property op een tijdstip uit het verleden te zetten. Vervolgens voeg je dan de gewijzigde cookie toe aan de response **(2)**.

- Voeg nu onderstaande view toe aan deze wissen-action.

```
<div class="jumbotron">
    <h1>Cookies</h1>
    <p class="lead">Cookie werd gewist.</p>
    <p><a href="/Home/Index" class="btn btn-primary btn-lg">Terug naar indexpagina &raquo;</a></p>
</div>
```

- Probeer de applicatie nu uit. Refresh eventueel de indexpagina een keer zodat je zeker een tijdstip te zien krijgt.
- Klik dan op *Cookie wissen*.
- Keer terug naar de indexpagina en stel vast dat je verwelkomd wordt alsof je de website voor het eerst bezoekt.

Tot nu toe hebben we per cookie slechts één waarde opgeslagen. Maar je kan er ook voor kiezen om meerdere name/value-paren op te slaan in één cookie. Men spreekt dan van subkeys.

We passen ons voorbeeld een beetje aan. I.p.v. enkel het tijdstip van ons laatste bezoek op te slaan, bewaren we ook de voorkeurtaal van de browser.

- Wijzig de code in de *Index()*-action als volgt :

```
public ActionResult Index()
{
    string resultaat = "Dit is jouw eerste bezoek";
```

```
if (Request.Cookies != null)
{
    if (Request.Cookies["lastvisit"] != null)
    {
        resultaat = "Welkom terug. Jouw laatste bezoek was op " +
            Request.Cookies["lastvisit"]["tijdstip"] +
            ". Jouw voorkeurtaal is " +
            Request.Cookies["lastvisit"]["taal"] + ".";
    }

    string laatsteBezoek = DateTime.Now.ToString();
    var userCookie = new HttpCookie("lastvisit");
    userCookie["tijdstip"] = laatsteBezoek;
    userCookie["taal"] = Request.UserLanguages[0];
    userCookie.Expires = DateTime.Now.AddDays(365);
    Response.Cookies.Add(userCookie);
}

ViewBag.Tijdstip = resultaat;
return View();
}
```

Deze keer maken we de cookie aan met een constructor met één parameter : de naam van de cookie (2). In deze cookie slaan we twee waarden op : het tijdstip en de voorkeurtaal van de browser (3). Om één van deze waarden op te vragen gebruiken we een speciale notatie met twee steken rechte haken (1).

Het resultaat met en-US als voorkeurtaal :



10.2 Session state

Wanneer we cookies gebruiken, dan bewaren we de informatie die over de pagina's heen moet onthouden worden op de client (=de browser). Je kan in ASP.NET MVC er ook voor kiezen deze informatie te bewaren op de server. Je bent dan niet afhankelijk van wat de gebruiker allemaal uitspookt. De gebruiker zal maar net je zorgvuldig opgebouwde cookie wissen.

Je kan de informatie op twee verschillende niveaus bijhouden : session en application.

Gegevens bijgehouden op session-niveau behouden hun waarde zolang de session duurt en zijn enkel toegankelijk vanuit dezelfde session. Elke gebruiker heeft zijn eigen set waarden.

Gegevens bijgehouden op application-niveau behouden hun waarde zolang de webapplicatie draait op de server en zijn toegankelijk vanuit alle sessions. Alle gebruikers delen dus dezelfde set waarden.

In deze paragraaf bekijken we eerst de session-state.

In een controller method bewaar je een waarde in een sessionvariabele via :

```
Session["naamsessionvariabele"] = waarde;
```

Vervolgens kan je deze waarde in de view ophalen via :

```
@Session["naamsessionvariabele"]
```

Een voorbeeld : we houden in een sessionvariabele bij hoeveel keer we de home-pagina van onze webapplicatie bekijken binnen eenzelfde session.

- We werken verder in de webapplicatie MVC_Voorbeeld3. Vervang de code in de Index-method van de HomeController door het onderstaande :

```
public ActionResult Index()
{
    if (this.Session["aantalBezoeken"] == null)
        this.Session["aantalBezoeken"] = 1;
    else
        this.Session["aantalBezoeken"] = (int)this.Session["aantalBezoeken"] + 1;
    return View();
}
```

Wanneer je voor het eerst, binnen dezelfde session, naar de homepagina surft bestaat er nog geen sessionvariabele met de naam *aantalBezoeken* en is deze variabele null. We zetten ze in dat geval op 1. Is er wel reeds een dergelijke variabele dan verhogen we deze met 1.

- Ga nu naar de bijhorende view. In deze view geven we de waarde van de sessionvariabele weer. Vervang de code binnen de 'jumbotron-div' door onderstaande code :

```
<div class="jumbotron">
    <h1>Session state</h1>
    <p class="lead">Welkom, dit is jouw @Session["aantalBezoeken"]e bezoek.</p>
</div>
```

- Probeer deze view uit in de browser.
- Refresh de pagina nu een aantal keer en je zal de teller zien verhogen. Surf je naar dezelfde URL in een andere session dan begint de telling opnieuw vanaf 1.

Opmerking : de meeste browsers beschouwen het browsen in een nieuw tabblad van de browser als browsen in dezelfde session. Wil je een ganse nieuwe session dan kopieer je best de URL, sluit je de browser volledig af en start je de browser opnieuw. Plak de URL in de browser en je hebt een nieuwe session.

Ook deze keer gaan we een wis-functie toevoegen.

- In de view Index.cshtml voorzien we terug een knop om de bewaarde informatie te wissen. Voeg onderstaande vetgedrukte code toe onderaan de 'jumbotron-div'.

```
<div class="jumbotron">
    ...
    <p><a href="~/Home/Wissen"
        class="btn btn-primary btn-lg">Sessionvariabele wissen &raquo;</a>
    </p>
</div>
```

- Wijzig de code in de Wissen-action als volgt :

```
public ActionResult Wissen()
{
    if (this.Session["aantalBezoeken"] != null)
        this.Session["aantalBezoeken"] = null;
    return View();
}
```

- Pas nu ook de code in *Wissen.cshtml* aan :

```
<div class="jumbotron">
    <h1>Sessionstate</h1>
    <p class="lead">Sessionvariabele werd gewist.</p>
    <p><a href="~/Home/Index"
        class="btn btn-primary btn-lg">Terug naar indexpagina &raquo;</a>
    </p>
</div>
```

Je kan de applicatie terug uitproberen : ververs de homepagina een paar keer om de teller te laten oplopen en klik dan op de wis-knop. Keer terug naar de indexpagina en de teller staat terug op 1.

Opmerking :

In de bovenstaande oefening hebben we één session-variabele gewist door deze op null te zetten. Je kan met de instructie **Session.Clear()**; alle session-variabelen in één keer wissen.

10.3 Application state

Bij Session state beschikt elke session over zijn eigen set variabelen. Zo komen gebruikers van de webapplicatie niet in elkaars vaarwater. Bij application state daarentegen worden de application-variabelen gedeeld door alle gebruikers van de webapplicatie.

We passen ons voorbeeld een beetje aan. Deze keer gaan we niet alleen het aantal bezoeken binnen eenzelfde sessie tellen maar ook het totaal aantal bezoeken sinds het starten van de applicatie.

Beide tellers gaan we initialiseren via code in een bestand *Global.asax*. Een dergelijke file vind je in de root van de webapplicatie.

- Dubbelklik op de file Global.asax en je ziet het volgende in het bijhorende Global.asax.cs bestand :

```
public class MvcApplication : System.Web.HttpApplication
{
    protected void Application_Start()
    {
        AreaRegistration.RegisterAllAreas();
        FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
        RouteConfig.RegisterRoutes(RouteTable.Routes);
        BundleConfig.RegisterBundles(BundleTable.Bundles);
    }
}
```

In deze codefile is er reeds een method Application_Start() aanwezig. Zoals de naam al doet vermoeden wordt deze method uitgevoerd bij de opstart van de application. Het is dan ook in deze method dat we extra code gaan noteren die een teller initialiseert.

- Voeg in de method Application_Start() onderstaande vetgedrukte regels code toe :

```
protected void Application_Start()
{
    ...
    Application.Lock();
    Application.Add("aantalBezoeken", 0);
    Application.UnLock();
}
```

Onze webapplicatie beschikt over een object Application. Via de method Add kunnen we een application variabele toevoegen met een naam en een waarde. Om dit op een veilige manier te laten gebeuren locken we de applicationstate voor het toevoegen van de variabele en unlocken we deze achteraf. Dit is niet strikt noodzakelijk maar wordt wel aanbevolen.

We zouden ook een tweede teller bishouden in een session variabele. Deze kunnen we initialiseren in een method Session_Start(). Deze is standaard niet aanwezig in Global.asax dus voegen we die zelf toe.

- Voeg onderstaande method toe in Global.asax.cs :

```
protected void Session_Start()
{
    Session["aantalBezoeken"] = 0;
}
```

Bij de start van een nieuwe sessie zal dus een session variabele met de naam *aantalBezoeken* worden aangemaakt en op 0 geïnitialiseerd worden.

We passen nu de code in de HomeController aan.

- Vervang de code in de Index-action als volgt :

```
public ActionResult Index()
{
    //sessionvariabele aanpassen
    this.Session["aantalBezoeken"] = (int)this.Session["aantalBezoeken"] + 1;

    //applicationvariabele aanpassen
    System.Web.HttpContext.Current.Application.Lock();
    System.Web.HttpContext.Current.Application["aantalBezoeken"] =
        (int)System.Web.HttpContext.Current.Application["aantalBezoeken"] + 1;
    System.Web.HttpContext.Current.Application.UnLock();

    return View();
}
```

De code voor de sessionvariabele komt ons al bekend voor. We hoeven deze keer niet te testen of er een dergelijke variabele bestaat want deze wordt in global.asax aangemaakt én geïnitialiseerd. De code voor de applicationvariabele lijkt op die in global.asax. We locken eerst de application en passen dan de applicationvariabele aan. Daarna voeren we een UnLock door.

- Wijzig nu in de bijhorende view de 'jumbotron-div' als volgt :

```
<div class="jumbotron">
    <h1>Session en application state</h1>
    <p class="lead">Welkom, dit is jouw @Session["aantalBezoeken"]e bezoek
    binnen deze sessie.</p>
    <p class="lead">Deze pagina werd in totaal reeds
    @HttpContext.Current.Application["aantalBezoeken"] keer bezocht.</p>
    <p><a href("~/Home/Wissen" class="btn btn-primary btn-lg">Session en
    applicationvariabele wissen &raquo;</a></p>
</div>
```

Om de waarde van een applicationvariabele aan te spreken gebruiken we

@HttpContext.Current.Application["naamvandevariabele"].

- Vervang nu de code in de Wissen-action door onderstaande code :

```
public ActionResult Wissen()
{
    //reset sessionvariabele
    this.Session["aantalBezoeken"] = 0;

    //reset applicationvariabele
    System.Web.HttpContext.Current.Application.Lock();
    System.Web.HttpContext.Current.Application["aantalBezoeken"] = 0;
    System.Web.HttpContext.Current.Application.UnLock();

    return View();
}
```

Aangezien de sessionvariabele zeker bestaat kunnen we deze gewoon de waarde 0 geven. Daarna zetten we ook de applicationvariabele op 0. We gebruiken zowel voor de sessionvariabele als voor de applicationvariabele dezelfde naam (aantalBezoeken) maar dat moet uiteraard niet.

Tenslotte pas je nog de view Wissen.cshtml aan.

- Vervang de code in Wissen.cshtml door onderstaande code :

```
<div class="jumbotron">
    <h1>Session en application state</h1>
    <p class="lead">Session- en applicationvariabele werden gewist.</p>
    <p><a href "~/Home/Index"
        class="btn btn-primary btn-lg">Terug naar indexpagina &raquo;</a>
    </p>
</div>
```

Je kan dit nu uitproberen.

- Start de applicatie en refresh de pagina enkele keren. Beide tellers worden verhoogd.
- Kopieer de URL in de browser.
- Sluit de browser.
- Start de browser opnieuw en plak de URL in de browser. Deze keer begint het tellen van de application variabele niet meer vanaf 0.
- Refresh de pagina opnieuw enkele keren daarna kan je eventueel de wisknop eens uitproberen.

Het resultaat na enkele keren refreshen in een nieuwe session :

Session en application state

Welkom, dit is jouw 5e bezoek binnen deze sessie.

Deze pagina werd in totaal reeds 10 keer bezocht.

[Session en applicationvariabele wissen »](#)

Enkele opmerkingen :

- In de Index-view hebben we de waarde van een session én een application variabele opgevraagd. Dit wordt ook wel information pulling genoemd. Dit wordt als een bad practice beschouwd. Beter is deze informatie via de ViewBag of het Model door te geven. Deze technieken heb je eerder in deze cursus geleerd. Een view hoort enkel informatie weer te geven die het aangeleverd krijgt en niet zelf om informatie te vragen.
- Het gebruik van session variabelen wordt afgeraden omdat de webapplicatie in dat geval moeilijker te unit testen is. Een beter alternatief is het gebruik van static classes.

10.4 Samenvatting

- Om gegevens over verschillende pagina's (views) heen te onthouden kan je gebruik maken van cookies, session- en applicationvariabelen.
- Er zijn sessioncookies die na de session vanzelf verdwijnen en ook persistent cookies die een vervaldatum hebben.
- Je maakt een cookie aan door een instance van de class HttpCookie aan te maken. Je geeft het een naam en eventueel een waarde. Een cookie opvragen doe je via het Request object, de waarde invullen via het Response object.
- Een waarde die voor iedere gebruiker anders is en binnen dezelfde session moet onthouden worden kan je in een sessionvariabele bewaren. Dit gebeurt via Session["variablename"].
- Een applicationvariabele wordt door alle gebruikers gedeeld. Je definieert deze in het bestand global.asax(.cs). Het gebruik van applicationvariabelen wordt afgeraden.

11 Tempdata en redirects

In deze cursus vertelden we eerder al dat als we een toevoeging, wijziging of verwijdering uitvoeren we dit doen met een POST request. Op pagina 26 kon je lezen dat de response van deze actie dan een redirectopdracht is naar een andere webpagina. Dit veroorzaakt een volledig nieuwe request waarin je totaal geen idee hebt van de toegevoegde, verwijderde of gewijzigde waarde. Dus een geruststellende boodschap "De gegevens van user Xyz uit Abc werden succesvol opgeslagen." is niet zomaar mogelijk.

Maar gelukkig is er Tempdata. Dit zal ervoor zorgen dat je gegevens kan onthouden over redirections heen. Tempdata maakt achter de schermen gebruik van Sessionvariabelen. Het verschil is dat wanneer je de gegevens uit Tempdata inleest (na een redirect) deze gegevens automatisch weer worden gewist.

11.1 Een verwijderoptie toevoegen

In ons voorbeeld MVC_Voorbeeld3 zullen we een filiaal verwijderen. Na het verwijderen krijgen we dan de boodschap dat het filiaal (met vermelding van de naam) succesvol is verwijderd.

We gaan eerst wat voorbereidend werk doen.

- Voeg in de class *FiliaalService* een *Read()*-method en een *Delete()*-method toe :

```
public Filiaal Read(int id)
{
    return filialen[id];
}

public void Delete(int id)
{
    filialen.Remove(id);
}
```

Met deze methods kunnen we aan de hand van het id, het filiaal opzoeken om de gegevens tijdelijk te bewaren en vervolgens het filiaal verwijderen.

We voegen nu in onze filialenlijst een hyperlink toe naar een pagina *verwijderen/<filiaalnummer>*.

- Pas de code van *FiliaLEN.cshtml* als volgt aan :

```
<h2>Filialen</h2>
<ul>
    @foreach (var filiaal in Model)
    {
        var url = "Verwijderen/" + filiaal.ID;

        <li>@filiaal.ID @filiaal.Naam @Html.DisplayFor(m=>filiaal.Gebouwd)
            @Html.DisplayFor(m => filiaal.Waarde) @Html.DisplayFor(m => filiaal.Eigenaar)
            <a href="@url">Verwijderen</a>
        </li>
    }
</ul>
```

De hyperlink zou nu van de vorm .../Filiaal/Verwijderen/2 moeten zijn.

- In de FiliaalController voeg je een action *Verwijderen* toe :

```
public ActionResult Verwijderen(int id)
{
    var filiaal = filiaalService.Read(id);
    return View(filiaal);
}
```

- Voeg nu een bijhorende view *Verwijderen* toe. Kies een Empty template met als Model class de class *Filiaal* en voeg er onderstaande code aan toe :

```
@model MVC_Voorbeeld3.Models.Filiaal
 @{
    ViewBag.Title = "Verwijderen";
}
<h2>@Model.Naam verwijderen?</h2>
<form method="post" action("~/Filiaal/Delete/@Model.ID">
    <input type="submit" value="Delete" />
</form>
```

- In bovenstaande view roept een klik op de submitknop een actionmethod *Delete* op met het filiaalid als parameter. Daarom voeg je in de FiliaalController nu een action *Delete* toe die het filiaal effectief verwijdert :

```
[HttpPost]
public ActionResult Delete(int id)
{
    var filiaal = filiaalService.Read(id); (1)
    this.TempData["filiaal"] = filiaal; (2)
    filiaalService.Delete(id); (3)
    return Redirect("~/Filiaal/Verwijderd"); (4)
}
```

Vooraleer we met de *Delete*-method het filiaal effectief verwijderen (3), lezen we de gegevens van het filiaal in (1) en stoppen we het in TempData (2). Daarna doen we een Redirect naar een nog te maken action *Verwijderd* (4).

Bemerk dat we de annotation `[HttpPost]` gebruiken. Voor actions die gegevens wijzigen, toevoegen of verwijderen gebruiken we een Post-request. In hoofdstuk 13 Html forms gaan we meer in detail in op Post- en Get-requests. Hier bekijken we voornamelijk de werking van TempData.

- Voeg een action *Verwijderd* toe :

```
using MVC_Voorbeeld3.Models;
...
public ActionResult Verwijderd()
{
    var filiaal = (Filiaal)this.TempData["filiaal"];
    return View(filiaal);
}
```

In deze action halen we de filiaalgegevens terug uit TempData. We geven deze dan door aan de bijhorende view.

- Voeg een view toe aan de action Verwijderd. Kies opnieuw een empty view met model class Filiaal :

```
@model MVC_Voorbeeld3.Models.Filiaal
@{
    ViewBag.Title = "Verwijderd";
}
<h2>Verwijdering van @Model.Naam is doorgevoerd.</h2>
<h3>Terug naar de <a href="Index">lijst</a></h3>
```

Bemerkt dat de hyperlink naar de lijst met filialen gewoon *Index* is en niet *Filiaal/Index*. Dit komt omdat je reeds vanuit een pagina binnen de map *Filiaal* een relatieve verwijzing maakt naar een andere pagina binnen dezelfde map.

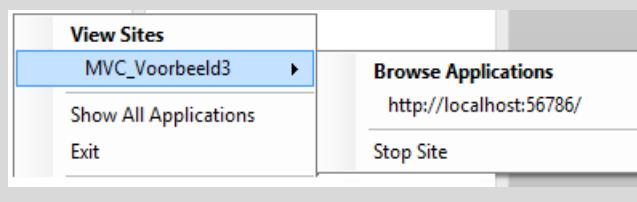
- Je kan nu uitproberen of je een filiaal kan verwijderen.

Bij het uittesten van deze webapplicatie kan je een resultaat krijgen dat je niet meteen verwacht. Stel : je start de webapplicatie op in debugmodus, verwijdert een filiaal en je sluit de browser.

Vervolgens stop je de debugsessie. Wanneer je daarna de webapplicatie opnieuw start is het mogelijk dat het filiaal nog steeds geschrapt is en dat had je misschien niet verwacht.

De filiaalgegevens zijn statische data. Dit betekent dat deze data blijft bewaard zolang de applicatie op de IIS Express draait. Wanneer je de browser sluit of stopt met debuggen blijft de applicatie actief op IIS Express. Het is pas wanneer je de Site expliciet stopt of IIS Express afsluit dat de statische data verloren gaat.

Om te testen welke Site(s) actief zijn op IIS Express klik je met de rechtermuistoets op het IIS Express icoontje. Klik dan op de Sitenaam en dan kan je ernaar browsen of de Site stoppen.



11.2 Samenvatting

- Na het updaten of verwijderen van gegevens wil je meestal een boodschap tonen aan de gebruiker. In die boodschap zal je dikwijls een stuk van de geüpdateerde of verwijderde gegevens willen tonen. Om dit in een stateless protocol als http te kunnen doen heb je een hulpmiddel nodig en dit bestaat in de vorm van TempData.
- Je stopt gegevens in TempData door een waarde toe te kennen aan TempData["variabelenaam"]. Deze waarde wordt na gebruik automatisch gewist.

11.3 Oefening

Voeg in de webapplicatie MVCBierenApplication naast ieder bier een kruisje (figuur *delete.png*) toe. Een klik op deze figuur verwijdert het bier. Zorg ook voor een toepasselijke melding welk bier er succesvol is verwijderd.

12 Hyperlinks

Tijdens het ontwikkelen van een ASP.NET MVC webapplication testen we vaak onze webapplicatie via de ingebouwde webserver van Visual Studio. Het publiceren van onze applicatie gebeurt telkens in de root van deze IIS Express. In ‘real life’ zullen webapplicaties niet direct in de root van een webserver gepubliceerd worden. Het is dan heel belangrijk dat je goed uitkijkt waar er wél of geen tilde (~) vlak vóór een url moet staan. Bij testing via de ingebouwde webserver komen dergelijke fouten zelden aan het licht.

12.1 Verwijzingen naar views met `Html.ActionLink` en `Url.Action`

Om bovenstaande problemen te vermijden kan je waar mogelijk best gebruik maken van de helperfuncties `Html.ActionLink` en `Url.Action`. Wanneer je deze functies gebruikt, dan denk je eerder in termen van verwijzingen naar actions binnen controllers i.p.v. verwijzingen naar urls.

Als voorbeeld pakken we de code in de (partial) view *FiliaLEN.cshtml* aan.

- Open *FiliaLEN.cshtml* uit de webapplication *MVC_Voorbeeld3* en wijzig de code zoals hieronder weergegeven :

```
@model List<MVC_Voorbeeld3.Models.Filiaal>
<h2>Filialen</h2>
<ul>
    @foreach (var filiaal in Model)
    {
        <li>@filiaal.ID @filiaal.Naam @Html.DisplayFor(m=>filiaal.Gebouwd)
            @Html.DisplayFor(m=>filiaal.Waarde) @Html.DisplayFor(m=>filiaal.Eigenaar)
            @Html.ActionLink("Verwijderen", "Verwijderen", new { ID = filiaal.ID })
        </li>
    }
</ul>
```

We gebruiken een zogenaamde Html-helperfunctie `ActionLink()` om te verwijzen naar de action *Verwijderen*. De eerste parameter van de `Html.ActionLink()`-functie is de zichtbare tekst van de hyperlink. De tweede parameter is de naam van de uit te voeren action. Met de laatste parameter geven we het filiaalnummer mee.

In het bovenstaande geval verwijzen we naar een action uit dezelfde controller. Willen we naar een action uit een andere controller verwijzen dan moeten we extra parameters vermelden : de naam van de controller (als 3^e parameter) en eventuele html-attributen als laatste parameter. Onze functieoproep zou er dan als volgt uitzien :

```
@Html.ActionLink("Verwijderen", "Verwijderen", "Filiaal", new { ID = filiaal.ID },
null)
```

In bovenstaand voorbeeld zijn er geen html-attributen opgegeven. De parameterwaarde is null. Je zou bijvoorbeeld een hint kunnen meegeven voor de gebruiker via het title-attribuut :

```
@Html.ActionLink("Verwijderen", "Verwijderen", "Filiaal", new { ID = filiaal.ID },
new { title="Filiaal " + filiaal.ID + " verwijderen" })
```

- Je kan het verwijderen opnieuw uitproberen.

Als we willen gebruik maken van een afbeelding om de verwijdering door te voeren dan kunnen we geen gebruik maken van `Html.ActionLink()` maar gebruiken we de functie `Url.Action()`.

- Voeg eerst het figuurtje `delete.png` toe aan de map `Images`.
- Wijzig in de (partial) view `FiliaLEN.cshtml` de code naar :

```
@model List<MVC_Voorbeeld3.Models.Filiaal>
<h2>Filialen</h2>
<ul>
    @foreach (var filiaal in Model)
    {
        var url = Url.Action("Verwijderen", "Filiaal", new { ID=filiaal.ID });
        <li>
            ...
            <a href="@url"><img src "~/Images/delete.png" alt="delete"/></a>
        </li>
    }
</ul>
```

Om de code een beetje leesbaar te houden stellen we de url samen en stoppen we deze in een variabele `url`. De `Url.Action()`-functie krijgt als parameters eerst de actionmethod mee, dan de controller – deze zou je eventueel mogen weglaten aangezien de gevraagde action in dezelfde controller zit – en vervolgens een object met daarin het filiaalnummer van het te verwijderen filiaal. Op het einde van ieder listitem plaatsen we dan de `<a>`-tag naar de samengestelde url rond het figuurtje.

- Je kan dit opnieuw uitproberen, zowel via de ingebouwde webserver in VS als na publicatie op IIS.

12.2 Redirection met Redirect en RedirectToAction

In de Filiaalcontroller zien we in de method `Delete` dat er na de verwijdering een redirection gebeurt naar de method `Verwijderd`. In ons voorbeeld gebeurt dit met de functie `Redirect` die als parameter een url meekrijgt.

```
return Redirect("~/Filiaal/Verwijderd");
```

Er bestaat ook een `RedirectToAction()`-functie die meer de voorkeur geniet boven `Redirect()`. In het kader hieronder leer je waarom. De `RedirectToAction()`-functie krijgt als parameter de naam van de method mee. Een tweede parameter kan eventueel de naam van de controller zijn indien de method zich in een andere controller bevindt.

```
return RedirectToAction("Verwijderd");
```

- Pas de code in de delete-method aan en probeer opnieuw uit.

Als om één of andere reden de directory-structuur van jouw website verandert dan moet je alle url's in jouw code veranderen als je gebruik maakt van `Redirect()`. Gebruik je `RedirectToAction()` dan is dit niet het geval.

12.3 Samenvatting

- Om problemen met de directorystructuur van de webserver te vermijden maak je best gebruik van de helperfuncties `Html.ActionLink()` en `Url.Action()` om te verwijzen naar een andere pagina.
- Wil je een redirection toepassen, gebruik dan `RedirectToAction()`.

12.4 Oefening

Pas de webapplicatie `MVCBierenApplication` aan zodat de hyperlinks en verwijzingen 'publiceer-safe' zijn.

13 Html forms, html helpers en validatie

In paragraaf 2.2.2 *Het verwerken van een request in een MVC applicatie* vermeldden we reeds dat als we gegevens willen wijzigen, toevoegen of verwijderen we steeds een POST request gebruiken. Het opvragen van gegevens gebeurt met een GET request. In dit hoofdstuk bekijken we beide in detail.

13.1 Voorbereiding

- Voeg eerst in de applicatie *MVC_Voorbeeld3* aan de folder *Models* een class *Persoon* toe :

```
using System.ComponentModel.DataAnnotations;
...
public class Persoon
{
    public int ID { get; set; }
    public string Voornaam { get; set; }
    public string Familienaam { get; set; }
    public int Score { get; set; }
    [DisplayFormat(DataFormatString = "{0:#,##0.00}")]
    public decimal Wedde { get; set; }
    public string Paswoord { get; set; }
    [DisplayFormat(DataFormatString = "{0:d}")]
    public DateTime Geboren { get; set; }
    public bool Gehuwd { get; set; }
    public string Opmerkingen { get; set; }
    public Geslacht Geslacht { get; set; }
}
```

- Voor het geslacht voeg je in dezelfde folder een enumeratie *Geslacht* toe:

```
public enum Geslacht {
    Man,
    Vrouw
}
```

In een nieuwe class *PersoonService* voegen we een statische lijst Persoon-objecten toe met eveneens een aantal basishandelingen : *FindAll()* om de volledige lijst op te vragen en *FindByID* om een persoon op te zoeken op ID.

- Voeg een class *PersoonService* toe in de folder *Services* :

```
using MVC_Voorbeeld3.Models;
...
public class PersoonService
{
    private static Dictionary<int, Persoon> personen = new Dictionary<int, Persoon>
    {
        {
            1, new Persoon {ID=1, Voornaam = "Jesse", Familienaam = "James", Score=5, Wedde=1000, Paswoord="123", Geboren=new DateTime(1966,1,1), Gehuwd=false, Opmerkingen="Schurk van het Wilde Westen", Geslacht=Geslacht.Man}
        },
        {
            2, new Persoon {ID=2, Voornaam = "Jane", Familienaam = "Calamity", Score=4, Wedde=2000, Paswoord="123", Geboren=new DateTime(1966,2,2), Gehuwd=false, Opmerkingen="Martha Jane Cannary", Geslacht=Geslacht.Vrouw}
        },
    };
}
```

```

{
    3, new Persoon {ID=3, Voornaam="Billy",Familienaam="The Kid", Score=5,
    Wedde=3000, Paswoord="123", Geboren=new DateTime(1966,3,3), Gehuwd=false,
    Opmerkingen="Revolverheld", Geslacht=Geslacht.Man}
},
{
    4, new Persoon {ID=4, Voornaam="Sarah",Familienaam="Bernhardt", Score=3,
    Wedde=4000, Paswoord="123", Geboren=new DateTime(1966,4,4), Gehuwd=false,
    Opmerkingen="Rosine Bernhardt", Geslacht=Geslacht.Vrouw}
}
};

public List<Persoon> FindAll()
{
    return personen.Values.ToList();
}

public Persoon FindByID(int id)
{
    return personen[id];
}
}

```

- Voeg ook een PersoonController toe. In de PersoonController voegen we een private instance van het object *PersoonService* toe zodat we een method als *FindAll()* kunnen gebruiken in de action *Index()* :

```

using MVC_Voorbeeld3.Services;
...
public class PersoonController : Controller
{
    private PersoonService persoonService = new PersoonService();

    public ActionResult Index()
    {
        return View(persoonService.FindAll());
    }
}

```

- Voeg een bijhorende view toe aan de *Index()*-method :

```

@model List<MVC_Voorbeeld3.Models.Persoon>
 @{
    ViewBag.Title = "Personen";
}
<h2>Personen</h2>
<table>
<thead>
<tr>
<th>ID</th>
<th>Voornaam</th>
<th>Familienaam</th>
<th>Score</th>
<th>Wedde</th>
<th>Paswoord</th>
<th>Geboren</th>
<th>Gehuwd</th>
<th>Opmerkingen</th>
<th>Geslacht</th>
</tr>
</thead>

```

```

<tbody>
@foreach(var persoon in Model)
{
    <tr>
        <td>@persoon.ID</td>
        <td>@persoon.Voornaam</td>
        <td>@persoon.Familienaam</td>
        <td>@persoon.Score</td>
        <td>@Html.DisplayFor(m=>persoon.Wedde)</td>
        <td>@persoon.Paswoord</td>
        <td>@Html.DisplayFor(m=>persoon.Geboren)</td>
        <td>@persoon.Gehuwd</td>
        <td>@persoon.Opmerkingen</td>
        <td>@persoon.Geslacht</td>
    </tr>
}
</tbody>
</table>

```

- Als je wil kan je dit al even uitproberen door te surfen naar .../Persoon.
- We voegen nog wat css toe in het bestand *Site.css* uit de map *Content* om de tabel op te maken :

```

table.zebra, table.zebra td, table.zebra.th
{
    border: 1px solid silver;
    border-collapse: collapse;
}

table.zebra thead
{
    color: white;
    background-color: gray;
}

table.zebra tbody tr:nth-child(even)
{
    background-color: #DDDDDD;
}

```

- In de view voorzien we de <table>-tag van de class "zebra" :

```

...
<table class="zebra">
...

```

De view zou er nu als volgt moeten uitzien :

Personen

ID	Voornaam	Familienaam	Score	Wedde	Paswoord	Geboren	Gehuwd	Opmerkingen	Geslacht
1	Jesse	James	5	1.000,00	123	1/01/1966	False	Schurk van het Wilde Westen	Man
2	Jane	Calamity	4	2.000,00	123	2/02/1966	False	Martha Jane Cannary	Vrouw
3	Billy	The Kid	5	3.000,00	123	3/03/1966	False	Revolverheld	Man
4	Sarah	Bernhardt	3	4.000,00	123	4/04/1966	False	Rosine Bernardt	Vrouw

13.2 Een persoon-object verwijderen

In het hoofdstuk over TempData hebben we al eens de mogelijkheid om een item te verwijderen toegevoegd. Bij wijze van opwarmingje voegen we deze functionaliteit ook hier toe.

- We beginnen met in de PersoonService een method te voorzien zodat een Persoon-object kan verwijderd worden :

```
public void Delete(int id)
{
    personen.Remove(id);
}
```

- In de view *Persoon/Index.cshtml* voeg je een extra kolom toe. De tabel krijgt dus een extra header en een extra cel. In die cel stoppen we een hyperlink naar een verwijderformulier *VerwijderForm*. Voor de hyperlink gebruiken we opnieuw het figuurtje *delete.png* :

```
...
<th>Verwijderen</th>
</tr>
...
<td>
    <a href="@Url.Action("VerwijderForm", "Persoon", new { ID = persoon.ID })">
        <img src("~/Images/delete.png" alt="delete"/>
    </a>
</td>
</tr>
...

```

- We passen nu de PersoonController aan door er een actionmethod *VerwijderForm* aan toe te voegen :

```
[HttpGet]
public ActionResult VerwijderForm(int id)
{
    return View(persoonService.FindByID(id));
}
```

Deze method zoekt de gegevens op van de te verwijderen persoon en stuurt die mee naar de bijhorende view. De annotation `[HttpGet]` moet je niet per se noteren maar het geeft wel duidelijk aan dat we hier iets opvragen.

- Voeg nu de view toe. Je kan kiezen voor de Empty template en de class *Persoon* instellen als Model class. De code van de view :

```
@model MVC_Voorbeeld3.Models.Persoon
 @{
    ViewBag.Title = Model.Voornaam + ' ' + Model.Familienaam + " verwijderen?";
}
<h2>@Model.Voornaam @Model.Familienaam verwijderen?</h2>
<form method="post" action="@Url.Action("Verwijderen", new {ID=Model.ID} )">
    <input type="submit" value="Verwijderen" />
</form>
```

In het formulier vragen we bevestiging voor het verwijderen van de persoon. De method van het formulier is post want we gaan gegevens verwijderen.

- De action Verwijderen maken we aan in de PersoonController :

```
[HttpPost]
public ActionResult Verwijderen(int id)
{
    persoonService.Delete(id);
    return RedirectToAction("Index");
}
```

De persoon verwijderen gebeurt met de method *Delete* uit de *PersoonService*. Na de verwijderactie gaan we opnieuw naar de *Index* van *Persoon*.

- Probeer maar uit.

13.3 Een POST-formulier samenstellen met html-helpers

In het vorige voorbeeld hebben we een html-formulier gebruikt waarbij we zelf de nodige html-code hebben geschreven. MVC heeft een eigen manier om formulieren samen te stellen. Het maakt daarbij gebruik van html-helpers om de nodige html-form code te genereren.

De werkwijze is als volgt :

1. Je maakt een Model class die de form voorstelt
2. Een GET request toont de form
3. Een POST request verwerkt de gesubmitte form na validatie

We proberen dit even uit aan de hand van een voorbeeld waarin we aan personen met een wedde tussen twee grenzen een weddeverhoging toe kennen.

13.3.1 De form-class

We starten met het aanmaken van een class die de form voorstelt.

- Voeg aan de folder *Models* een class *OpslagForm.cs* toe :

```
using System.ComponentModel.DataAnnotations;
...
public class OpslagForm
{
    [Display(Name = "Van wedde")]
    public decimal VanWedde { get; set; }
    [Display(Name = "Tot wedde")]
    public decimal TotWedde { get; set; }
    [Display(Name = "Percentage")]
    public decimal Percentage { get; set; }
}
```

Per invulveld maken we een property aan. We voegen er een *[Display]* annotation aan toe. Deze heeft één parameter, namelijk de tekst voor een label bij het invulveld.

13.3.2 De form tonen met een GET request

- Voeg nu een method *Opslag()* toe aan de PersoonController :

```
using MVC_Voorbeeld3.Models;
...
public ActionResult Opslag()
{
    OpslagForm opslagForm = new OpslagForm();
    return View(opslagForm);
}
```

We maken eerst een instance aan van de class *OpslagForm*. Deze geven we mee als parameter naar de bijhorende view (die we nog moeten maken).

- Voeg een view toe met de naam *Opslag.cshtml*. Kies als Model class voor *OpslagForm* en klik op *Add*.

We zullen deze view stap voor stap samenstellen en de opbouw in de browser volgen.

- Voeg alvast onderstaande code toe aan de view :

```
@model MVC_Voorbeeld3.Models.OpslagForm
 @{
    ViewBag.Title = "Opslag";
}

<h2>Opslag</h2>
@using (Html.BeginForm())
{
}
```

In de code `@using (Html.BeginForm())` gebruiken we een Html-helperfunctie om de nodige form html-tags voor ons te genereren. Voorlopig laten we het hierbij.

Om dit formulier te kunnen bereiken voegen we in de view *Persoon/Index.cshtml* helemaal onderaan een hyperlink toe naar de action *Opslag* uit de PersoonController.

- Voeg onderstaande vetgedrukte code toe in de view *Index.cshtml* in de folder *Views/Persoon* :

```
...
</table>
@Html.ActionLink("Opslag", "Opslag", "Persoon")
```

- Je kan de applicatie al eens starten door te browsen naar .../Persoon/Index. Op de indexpagina is er een hyperlink naar de opslagform. Volg deze hyperlink en bekijk de source van deze pagina. Er is een html-form gemaakt met een POST-action.

- In de view *Opslag.cshtml* voeg je binnen de `@using`-opdracht volgende code toe :

```
@Html.LabelFor(m => m.VanWedde)
@Html.TextBoxFor(m => m.VanWedde)
@Html.LabelFor(m => m.TotWedde)
@Html.TextBoxFor(m => m.TotWedde)
@Html.LabelFor(m => m.Percentage)
@Html.TextBoxFor(m => m.Percentage)
<input type="submit" value="Weddeverhoging doorvoeren" />
```

Per invulveld hebben we een label toegevoegd met de html-helperfunctie `Html.LabelFor()` en een `textBox` met de functie `Html.TextBoxFor()`.

We voegen nog wat CSS toe om de opmaak te verbeteren.

- Voeg in `Site.css` in de folder `Content` het volgende toe en probeer daarna uit in de browser.

```
input {  
    display: block;  
    margin-bottom: 10px;  
}
```

Het resultaat :

The screenshot shows a simple web form with a title 'Opslag'. It contains three text input fields: 'Van wedde' with value '0', 'Tot wedde' with value '0', and 'Percentage' with value '0'. Below these fields is a blue button labeled 'Weddeverhoging doorvoeren'.

Zoals je ziet bevatten de invulvelden allemaal de waarde 0. Als je dit niet wil dan wijzig je in `OpslagForm.cs` het type van de properties naar nullable decimals :

```
...  
[Display(Name = "Van wedde:")]  
public decimal? VanWedde { get; set; }  
...
```

- Verander de properties `VanWedde` en `TotWedde` in `OpslagForm.cs` naar nullable types.

Wil je aan een property een defaultwaarde meegeven dan kan je dit doen in de actionmethod `Opslag()` in de `PersonenController` :

```
public ActionResult Opslag()  
{  
    OpslagForm opslagForm = new OpslagForm();  
    opslagForm.Percentage = 10;  
    return View(opslagForm);  
}
```

- Geef het opslagpercentage een defaultwaarde 10 door de code van `Opslag()` te wijzigen zoals hierboven weergegeven.

Het nieuwe resultaat :

The screenshot shows the same 'Opslag' form as before, but the 'Percentage' input field now contains the value '10'.

13.3.3 De wijziging doorvoeren met een POST-request

We voegen nu actie toe aan de knop *Weddeverhoging doorvoeren*.

- Voeg in de PersoonController een extra method *Opslag()* toe met een parameter van het type *OpslagForm*. Deze parameter bevat de ingevulde waarden van de form. De andere actionmethod die eveneens *Opslag()* heet maar geen parameters heeft blijft behouden!

```
[HttpPost]
public ActionResult Opslag(OpslagForm opslagForm) {
    //hier komt nog code die de opslag doorvoert
    return View();
}
```

Met de annotation `[HttpPost]` geven we aan dat er een POST-request moet gebruikt worden.

- Voorlopig doet de method nog niet veel dus maken we in de PersoonService een method aan die de gewenste opslag kan doorvoeren :

```
public void Opslag(decimal vanWedde, decimal totWedde, decimal percentage)
{
    foreach (var p in
        (from persoon in personen.Values
         where persoon.Wedde >= vanWedde && persoon.Wedde <= totWedde
         select persoon)) //linq-query selecteert personen met de juiste wedde
    {
        p.Wedde += p.Wedde * percentage / 100;
    }
}
```

- Vervolledig de method *Opslag* in de PersonenController :

```
[HttpPost]
public ActionResult Opslag(OpslagForm opslagForm)
{
    persoonService.Opslag(opslagForm.VanWedde.Value,
                          opslagForm.TotWedde.Value,
                          opslagForm.Percentage);
    return RedirectToAction("Index");
}
```

Omdat de invulwaarden voor VanWedde en TotWedde nullable zijn moeten we `.Value` gebruiken om de waarden door te geven.

- Je kan opnieuw uitproberen en een weddeverhoging doorvoeren.

13.3.4 Validatie van het formulier

Het formulier werkt nu, maar om zeker te zijn dat er voldoende en correcte gegevens worden ingevuld zullen we het formulier valideren.

Om aan te geven dat een veld verplicht moet ingevuld worden gebruik je de annotation `[Required]`. Deze annotation heeft een parameter *ErrorMessage*.

- Voeg in de class *OpslagForm.cs* de annotation `[Required]` toe voor de invulvelden *VanWedde*, *TotWedde* en *Percentage* zoals hieronder weergegeven.

Met de annotation [Range] kunnen we eisen dat de ingevulde waarde voor een veld tussen twee opgegeven waarden ligt. Deze annotation heeft parameters voor een onder- en een bovengrens en voor een ErrorMessage.

- Voeg in de class *OpslagForm.cs* de annotation [Range] toe voor het veld *Percentage*.

De volledige code ziet er nu als volgt uit :

```
public class OpslagForm
{
    [Display(Name = "Van wedde")]
    [Required(ErrorMessage = "Van wedde is een verplicht veld")]
    public decimal? VanWedde { get; set; }

    [Display(Name = "Tot wedde")]
    [Required(ErrorMessage = "Tot wedde is een verplicht veld")]
    public decimal? TotWedde { get; set; }

    [Display(Name = "Percentage")]
    [Required(ErrorMessage = "Percentage is een verplicht veld")]
    [Range(0, 100, ErrorMessage = "De minimum- en maximumwaarden zijn : {1} en {2}")]
    public decimal Percentage { get; set; }
}
```

Om de validatie te activeren testen we in de actionmethod de waarde van *ModelState.IsValid*. Indien deze true is dan is de forminvoer geldig en kunnen we effectief de wijziging doorvoeren. Is het false dan tonen we de form opnieuw zodat er correcte data kan ingevoerd worden.

- Wijzig de method *Opslag* in de PersoonController :

```
[HttpPost]
public ActionResult Opslag(OpslagForm opslagForm)
{
    if (this.ModelState.IsValid)
    {
        //geen validatiefouten
        persoonService.Opslag(opslagForm.VanWedde.Value,
                              opslagForm.TotWedde.Value,
                              opslagForm.Percentage);
        return RedirectToAction("Index");
    }
    else
    {
        //wel validatiefouten
        return View(opslagForm);
    }
}
```

- Om de validatiefout(en) te tonen pas je de opslagform (*Opslag.cshtml*) aan:

```
@model MVC_Voorbeeld3.Models.OpslagForm
 @{
    ViewBag.Title = "Opslag";
}
<h2>Opslag</h2>
@using (Html.BeginForm())
{
    @Html.LabelFor(m => m.VanWedde)
    @Html.ValidationMessageFor(m => m.VanWedde)
    @Html.TextBoxFor(m => m.VanWedde)
```

```

@Html.LabelFor(m => m.TotWedde)
@Html.ValidationMessageFor(m => m.TotWedde)
@Html.TextBoxFor(m => m.TotWedde)

@Html.LabelFor(m => m.Percentage)
@Html.ValidationMessageFor(m => m.Percentage)
@Html.TextBoxFor(m => m.Percentage)

<input type="submit" value="Weddeverhoging doorvoeren" />
}

```

Om de validatiefouten voldoende te doen opvallen voegen we nog wat css toe in Site.css.

- Voeg in Site.css onderstaande css toe :

```

/* Styles for validation helpers
-----
.field-validation-error
{
    color: #ff0000;
}

.field-validation-valid
{
    display: none;
}

.input-validation-error
{
    border: 1px solid #ff0000;
    background-color: #ffeeee;
}

.validation-summary-errors
{
    font-weight: bold;
    color: #ff0000;
}

.validation-summary-valid
{
    display: none;
}

```

- Probeer maar uit.

Het resultaat :

Opslag

Van wedde:Van wedde is een verplicht veld

Tot wedde:Tot wedde is een verplicht veld

Percentage:De minimum- en maximumwaarden zijn : 0 en 100
 1011

De foutmeldingen staan nu naast elk label. We kunnen deze foutmeldingen ook centraliseren. Dit doe je met de Html-helperfunctie `Html.ValidationSummary()`.

- Voeg helemaal onderaan het bestand `Opslag.cshtml` het volgende toe :

```
@Html.ValidationSummary("Er zijn validatiefouten:")
```

Resultaat :

The screenshot shows a form titled "Opslag". It contains three text input fields: "Van wedde", "Tot wedde", and "Percentage", each with a red border indicating an error. Below the inputs is a button labeled "Weddeverhoging doorvoeren". A red error message "Er zijn validatiefouten" is displayed, followed by a bulleted list of validation errors: "Van wedde is een verplicht veld", "Tot wedde is een verplicht veld", and "De minimum- en maximumwaarden zijn : 0 en 100".

Je ziet de validatiefouten nu op twee plaatsen staan en dat is misschien wat te veel. Daarom kan je in `Opslag.cshtml` de `@Html.ValidationMessageFor()`-opdrachten een extra parameter geven, bijvoorbeeld een sterretje :

```
@Html.ValidationMessageFor(m => m.VanWedde, "*")
```

- Breng de nodige aanpassingen aan in `Opslag.cshtml`.

De Errormessage voor Percentage in `OpslagForm.cs` kan je dan misschien beter als volgt veranderen :

```
[Range(0, 100, ErrorMessage = "De minimum- en maximumwaarden voor percentage zijn : {1} en {2}")]
```

Op die manier weten we perfect bij welk veld deze validatiefout hoort.

- Maak de aanpassing in `OpslagForm.cs` en probeer uit.

Wanneer we geen getal maar een letter of woord intikken in de velden krijgen we volgende validatiesamenvatting :

The screenshot shows the same form as before, but with non-numeric input ("a", "b", and "c") in the text fields. The validation errors now point directly to the labels: "The value 'a' is not valid for Van wedde:", "The value 'b' is not valid for Tot wedde:", and "The value 'c' is not valid for Percentage:".

De foutberichten zijn in het Engels, terwijl we toch liever alles in het Nederlands houden.

- Voeg in het project een folder *App_GlobalResources* toe. Dit gaat het snelst door in de Solution Explorer met de rechtermuistoets te klikken op het project en te kiezen voor *Add – Add ASP.NET Folder – App_GlobalResources*.
- In die folder voeg je via *Add – New Item...* een Resources File toe. Geef deze de naam *Messages.resx*.
- Onder *Name* tik je *PropertyValueInvalid* en onder *Value* tik je *{0} is een verkeerde waarde voor {1}*. Bewaar de resourcefile.

	Name	Value	Comment
.	PropertyValueInvalid	{0} is een verkeerde waarde voor {1}	
*			

- In het bestand *Global.asax.cs* voeg je in de method *Application_Start()* deze lijn toe :
`DefaultModelBinder.ResourceClassKey = "Messages";`
- Je kan het voorbeeld terug uitproberen. De foutberichten zijn nu in het Nederlands.

13.4 Een GET-formulier samenstellen met html-helpers

Voor requests die aan de kant van de server geen toevoegingen, wijzigingen of verwijderingen doen gebruiken we een GET-formulier. De werkwijze bij een GET-formulier is vrij gelijkaardig als deze bij een POST-formulier :

1. Je maakt een Model class die de form voorstelt en ook het resultaat van de Get
2. Een GET request toont de form
3. Een GET request verwerkt de gesubmitte form na validatie

We proberen dit even uit aan de hand van een voorbeeld waarin we personen met een wedde tussen twee grenzen opvragen.

13.4.1 De form-class

- In de folder *Models* voeg je onderstaande class *VanTotWeddeForm.cs* toe.

```
using System.ComponentModel.DataAnnotations;
...
public class VanTotWeddeForm
{
    [Display(Name = "Van wedde")]
    [Required(ErrorMessage = "Van wedde is verplicht")]
    public decimal? VanWedde { get; set; }

    [Display(Name = "Tot wedde")]
    [Required(ErrorMessage = "Tot wedde is verplicht")]
    public decimal? TotWedde { get; set; }

    public List<Persoon> Personen { get; set; }
}
```

Onder de twee invulvelden merken we ook een List van Personen om het resultaat in op te slaan.

13.4.2 De form tonen met een GET request

- In de PersoonController voeg je een method toe :

```
[HttpGet]
public ActionResult VanTotWedde()
{
    var form = new VanTotWeddeForm();
    return View(form);
}
```

- Je voegt een bijhorende view toe. Als template kies je ‘Empty’, de Modelclass wordt *VanTotWeddeForm*. De code :

```
@model MVC_Voorbeeld3.Models.VanTotWeddeForm
 @{
    ViewBag.Title = "VanTotWedde";
}
<h2>Weddes tussen twee grenzen</h2>
@using (Html.BeginForm("VanTotWeddeResultaat", "Persoon", FormMethod.Get))
{
    @Html.LabelFor(m=>m.VanWedde)
    @Html.ValidationMessageFor(m=>m.VanWedde)
    @Html.TextBoxFor(m=>m.VanWedde)
    @Html.LabelFor(m=>m.TotWedde)
    @Html.ValidationMessageFor(m=>m.TotWedde)
    @Html.TextBoxFor(m=>m.TotWedde)
    <input type="submit" value="Zoeken" />
}
```

Je bemerkt hier een verschil met de Opslagform. De *Html.BeginForm()*-functie krijgt nu extra parameters mee. De eerste twee parameters geven aan welke de action is bij het submitten en in welke controller deze te vinden is. De derde parameter geeft aan dat het een GET-request is. Doen we dit niet dan wordt opnieuw het VanTotWedde-formulier getoond met een POST-request.

13.4.3 De form verwerken met een GET-request

- Voeg nu in de PersoonController de action *VanTotWeddeResultaat* toe.

```
[HttpGet]
public ActionResult VanTotWeddeResultaat(VanTotWeddeForm form)
{
    if (this.ModelState.IsValid)
    {
        //hier komt nog code (even geduld...)
    }
}
```

In deze method zullen we, wanneer het formulier de validatie heeft doorstaan, de weddes opzoeken die tussen de grenzen liggen. Dit doen we aan de hand van een method die we in de PersoonService schrijven.

- Voeg onderstaande code toe in *PersoonService.cs* :

```
public List<Persoon> VanTotWedde(decimal van, decimal tot)
{
    return (from persoon in personen.Values
            where persoon.Wedde >= van && persoon.Wedde <= tot
            orderby persoon.Wedde
            select persoon).ToList();
}
```

We gebruiken deze method nu in de action *VanTotWeddeResultaat* :

- Wijzig de code van *VanTotWeddeResultaat* in *PersoonController.cs* :

```
[HttpGet]
public ActionResult VanTotWeddeResultaat(VanTotWeddeForm form)
{
    if (this.ModelState.IsValid) {
        form.Personen = persoonService.VanTotWedde(
            form.VanWedde.Value, form.TotWedde.Value);
    }
    return View(form);
}
```

Wanneer we nu een view toevoegen dan zouden we een view *VanTotWeddeResultaat.cshtml* krijgen terwijl we eigenlijk het resultaat op dezelfde webpagina willen zien : het formulier. Daarom passen we de laatste lijn aan.

- Vervang de laatste lijn van *VanTotWeddeResultaat()* als volgt :

```
return View("VanTotWedde", form);
```

- Voeg in de view *VanTotWedde.cshtml* helemaal onderaan onderstaande code toe die de personen uitlijst die aan het criterium voldoen :

```
@if (Model.Personen != null) {
    <ul>
        @foreach(var persoon in Model.Personen)
        {
            <li>@persoon.Voornaam @persoon.Familienaam @persoon.Wedde</li>
        }
    </ul>
}
```

Je kan dit nu uitproberen door te browsen naar .../Persoon/VanTotWedde. Bemerk dat de url er bijvoorbeeld als volgt uitziet :

[.../Persoon/VanTotWeddeResultaat?VanWedde=2500&TotWedde=3500](#)

Bij een GET-request zitten de request-parameters in de URL wat ons bovendien het voordeel geeft dat we deze zoekopdracht kunnen bewaren in onze favorieten.

13.4.4 Bugfix voor decimale waarden

Wanneer we in ons voorbeeld weddes opvragen tussen grenswaarden waarvan minstens één van beide grenswaarden een decimaal teken bevat, dan krijgen we een validatiefout :

Van wedde
2000

Tot wedde **5000,5 is een verkeerde waarde voor Tot wedde**

5000,5

Zoeken

De komma wordt plots niet meer herkend als decimaal teken. Dit is een probleem dat zich enkel voordoet bij een Get-request. In het vorige voorbeeld (Opslag) gebruikten we een Post-request en daar doet dit probleem zich niet voor.

De oplossing ligt in een modelbinder.

- Voeg in de root van het project een class DecimalModelBinder.cs toe :

```
using System.Globalization;
using System.Web.Mvc;

...
public class DecimalModelBinder : IModelBinder
{
    public object BindModel(ControllerContext controllerContext,
                           ModelBindingContext bindingContext)
    {
        ValueProviderResult valueResult = bindingContext.ValueProvider
            .GetValue(bindingContext.ModelName);
        ModelState ModelState = new ModelState { Value = valueResult };
        object actualValue = null;
        try {
            actualValue = Convert.ToDecimal(valueResult.AttemptedValue,
                                              CultureInfo.CurrentCulture);
        }
        catch (FormatException e) {
            ModelState.Errors.Add(e);
        }
        bindingContext.ModelState.Add(bindingContext.ModelName, ModelState);
        return actualValue;
    }
}
```

- Je voegt deze modelbinder nu toe in Global.asax.cs in de method Application_Start() voor objecten van het type decimal en nullable decimal :

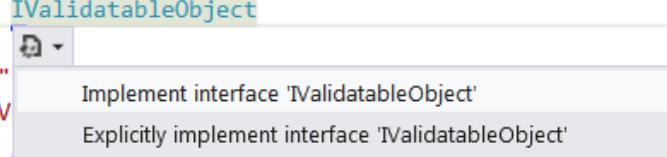
```
protected void Application_Start()
{
    ...
    ModelBinders.Binders.Add(typeof(decimal), new DecimalModelBinder());
    ModelBinders.Binders.Add(typeof(decimal?), new DecimalModelBinder());
}
```

- Probeer nu weddes op te vragen tussen waarden met een decimaal teken. Dit zou deze keer geen validatiefout meer mogen geven.

13.4.5 Meerdere velden t.o.v. elkaar valideren

Als kers op de taart gaan we nu eisen dat de ingevulde VanWedde kleiner is dan de TotWedde. Om velden op deze manier te kunnen valideren laten we de class *VanTotWeddeForm* de interface *IValidatableObject* implementeren.

```
public class VanTotWeddeForm : IValidatableObject
{
    [Display(Name = "Van wedde")]
    [Required(ErrorMessage = "V")]
    public decimal? VanWedde {
        [Display(Name = "Tot wedde")]
    }
}
```



The screenshot shows the code editor with the class definition. A tooltip appears over the 'VanWedde' property, providing two options: 'Implement interface 'IValidatableObject'' and 'Explicitly implement interface 'IValidatableObject''. This indicates that the developer is being prompted to implement the validation logic for this field.

Van zodra je een dubbele punt tikt en de naam van de interface, dan verschijnt er een blauw lijntje onder de I van IValidatableObject. Als je de cursor hier op laat rusten kan je via een 'smart tag' de nodige code laten toevoegen om deze interface te implementeren.

- Laat Visual Studio de interface implementeren via de smart tag en vul de code verder aan :

```
public IEnumerable<ValidationResult> Validate(ValidationContext validationContext)
{
    var validationResults = new List<ValidationResult>();
    if (VanWedde > TotWedde) {
        validationResults.Add(new ValidationResult(
            "TotWedde is kleiner dan VanWedde"));
    }
    return validationResults;
}
```

Je maakt dus een List van ValidationResult-objecten aan en voegt er één aan toe wanneer de VanWedde groter is dan de TotWedde. Deze List retourneer je dan.

In *VanTotWedde.cshtml* voegen we nog een validationsummary toe zodat de fout kan weergegeven worden.

- Wijzig de code in *VanTotWedde.cshtml* als volgt :

```
...
@Html.TextBoxFor(m=>m.TotWedde)
<input type="submit" value="Zoeken" />
@Html.ValidationSummary()
}
...
```

- Je kan de webapplicatie opnieuw uitproberen.

Indien je, om welke reden dan ook, zelf een fout wil weergeven dan kan je dat in de controller doen via de method *AddModelError()* van de *ModelState*.

Een voorbeeld : we willen de lijst beperken tot maximum 3 resultaten. Wanneer er meer dan 3 personen aan de criteria voldoen dan tonen we een fout.

```
[HttpGet]
public ActionResult VanTotWeddeResultaat(VanTotWeddeForm form) {
    if (this.ModelState.IsValid)
    {
        var lijst = persoonService.VanTotWedde(
            form.VanWedde.Value, form.TotWedde.Value);
        if (lijst.Count <= 3)
            // geen extra probleem
            form.Personen = lijst;
        else
            // wél een probleem
            this.ModelState.AddModelError("", "Te veel resultaten");
    }
    return View("VanTotWedde", form);
}
```

13.5 Input elements genereren met Html.EditorFor

In de voorbije voorbeelden hebben we de data telkens voorgesteld met een textbox. Daarvoor gebruikten we `Html.TextBoxFor()` en met `Html.LabelFor()` werd de textbox van een label voorzien. Er bestaan gelukkig nog andere input elementen zoals checkboxen, keuzelijsten,...

Met de functie `Html.EditorFor()` laten we MVC zelf beslissen welk element er wordt gekozen. MVC doet dit door naar het type van de parameter te kijken : voor een string-property zal er een textbox worden genomen, voor een boolean een checkbox, enzovoort.

13.5.1 Een persoon toevoegen

We proberen dit even uit. We maken een form om een persoon toe te voegen.

- Voeg aan de PersoonController volgende actionmethod toe :

```
[HttpGet]
public ActionResult Toevoegen() {
    var persoon = new Persoon();
    persoon.Geslacht = Geslacht.Vrouw; //defaultwaarde voor Geslacht
    return View(persoon);
}
```

We maken een nieuwe instantie aan van de class `Persoon` met eventueel reeds een defaultwaarde voor `Geslacht` of een ander veld. We geven deze persoon dan door aan de view.

- Voeg een bijhorende view toe. Kies een empty template en als model class kies je de class `Persoon`. De code :

```
@model MVC_Voorbeeld3.Models.Persoon
 @{
    ViewBag.Title = "Toevoegen";
}
<h2>Persoon toevoegen</h2>
@using (Html.BeginForm())
{
    @Html.LabelFor(m => m.Voornaam)
    @Html.EditorFor(m => m.Voornaam)
    @Html.LabelFor(m => m.Familienaam)
    @Html.EditorFor(m => m.Familienaam)
    @Html.LabelFor(m => m.Score)
    @Html.EditorFor(m => m.Score)
    @Html.LabelFor(m => m.Wedde)
    @Html.EditorFor(m => m.Wedde)
    @Html.LabelFor(m => m.Paswoord)
    @Html.EditorFor(m => m.Paswoord)
    @Html.LabelFor(m => m.Geboren)
    @Html.EditorFor(m => m.Geboren)
    @Html.LabelFor(m => m.Gehuwd)
    @Html.EditorFor(m => m.Gehuwd)
    @Html.LabelFor(m => m.Opmerkingen)
    @Html.EditorFor(m => m.Opmerkingen)
    @Html.LabelFor(m => m.Geslacht)
    @Html.EditorFor(m => m.Geslacht)
    <input type="submit" value="toevoegen" />
    @Html.ValidationSummary()
}
```

Browsen naar ..//Persoon/Toevoegen geeft het volgende resultaat :

Persoon toevoegen

Voornaam	<input type="text"/>
Familienaam	<input type="text"/>
Score	<input type="text" value="0"/> <input type="button" value=""/>
Wedde	<input type="text" value="0,00"/>
Paswoord	<input type="text"/>
Geboren	<input type="text" value="1/01/0001 0:00:00"/>
Gehuwd	<input checked="" type="checkbox"/>
Opmerkingen	<input type="text"/>
Geslacht	<input type="text" value="Vrouw"/>
<input type="button" value="toevoegen"/>	

We voegen nu een actionmethod toe die de toevoeging doorvoert.

- Voeg in de PersoonController een extra method toe :

```
[HttpPost]
public ActionResult Toevoegen(Persoon p)
{
    if (this.ModelState.IsValid) (1)
    {
        persoonService.Add(p); (2)
        return RedirectToAction("Index"); (3)
    }
    else
        return View(p); (4)
}
```

Deze method wordt uitgevoerd wanneer we het formulier submitten. Indien de validatie OK is (1) wordt er een persoonsobject toegevoegd via een nog te schrijven method *Add* uit *persoonService* (2). Daarna gaan we terug naar de personenlijst (3). Is er een validatiefout dan krijgt de gebruiker het toevoegformulier opnieuw te zien (4).

- Voeg in *PersoonService.cs* de method *Add()* toe :

```
public void Add(Persoon p)
{
    p.ID = personen.Keys.Max() + 1;
    personen.Add(p.ID, p);
}
```

Om geen dubbele keys te hebben in de dictionary zoeken we de hoogste waarde op met `.Keys.Max()` en voegen er 1 bij.

Je kan in de view `Index.cshtml` onderaan een link toevoegen naar de toevoegform:

```
@Html.ActionLink("Persoon toevoegen", "Toevoegen")
```

- Je kan het toevoegen nu uitproberen.

Opmerking :

In de viewcode in `Toevoegen.cshtml` gebruikten we zonet de Html-helper functie `Html.BeginForm()`. We hebben niet vermeld welke action er moet uitgevoerd worden bij een druk op de submit-knop. Hoe weet MVC dan welke action er moet uitgevoerd worden?

Als je niks vermeldt dan zoekt MVC een actionmethod met dezelfde naam als de view maar met een Post-method. Heb je jouw Post-actionmethod die de toevoeging doorvoert een andere naam gegeven, bijvoorbeeld `ToevoegingDoorvoeren` i.p.v. `Toevoegen`, dan moet je het volgende schrijven in de view :

```
@using (Html.BeginForm("ToevoegingDoorvoeren", "Persoon", FormMethod.Post))
```

13.5.2 De layout aanpassen

We brengen nu nog een aantal aanpassingen aan het formulier aan.

Ons formulier bevat een paswoord-veld.

- Om het paswoord onleesbaar te maken bij het intikken voeg je in de class `Persoon.cs` onderstaande annotation toe :

```
...
[DataType(DataType.Password)]
public string Paswoord { get; set; }
...
```

Afhankelijk van de gebruikte browser zullen er bolletjes of blokjes verschijnen bij het intikken van een wachtwoord.

Er bestaan nog meer DataTypes :

datum	<code>[DataType(DataType.Date)]</code>
textarea	<code>[DataType(DataType.MultilineText)]</code>
currency	<code>[DataType(DataType.Currency)]</code>
email	<code>[DataType(DataType.EmailAddress)]</code>
URL	<code>[DataType(DataType.Url)]</code>

- Pas een datatype `Date` toe op het veld `Geboren` en `MultilineText` op het veld `Opmerkingen`.

Opdat het veld `Geboren` ook in het formulier als datum zou weergegeven worden moet je in `Persoon.cs` de annotation `[DisplayFormat]` voorzien van een extra parameter `ApplyFormatInEditMode`.

- Pas de annotation `DisplayFormat` van de property `Geboren` als volgt aan :

```
[DisplayFormat(DataFormatString = "{0:d}", ApplyFormatInEditMode = true)]
```

Het resultaat :

Persoon toevoegen

Voornaam

Familienaam

Score

Wedde

Paswoord

Geboren

Gehuwd

Opmerkingen

Geslacht
 Vrouw
 Man

toevoegen

Zoals je ziet is het opmerkingenveld nu een textarea. Er is wel nog wat werk aan de schikking. Daarvoor gebruiken we css.

- In Site.css pas je ook voor het input element **textarea** dezelfde opmaak toe als voor **input** :

```
input, textarea
{
    display:block;
    margin-bottom:10px;
}
```

Bij checkboxen en radiobuttons staat het verklarende label gewoonlijk rechts van het input element, op dezelfde lijn.

- Voeg volgende css-code toe zodat label en input element op dezelfde lijn staan :

```
input[type='checkbox'], input[type='radio']
{
    display:inline;
}
```

- En wijzig in de view de volgorde :

```
...
@Html.EditorFor(m => m.Gehuwd)
@Html.LabelFor(m => m.Gehuwd)
...
```

Wanneer je het nu uitprobeert zal je zien dat het label van de Opmerkingen text area eveneens op dezelfde lijn als de checkbox komt te staan. Dit verhindert je door de checkbox en het bijhorende label in een div te plaatsen.

- Maak de nodige wijziging :

```
<div>@Html.EditorFor(m => m.Gehuwd)
@Html.LabelFor(m => m.Gehuwd)</div>
```

Het nieuwe resultaat :

The screenshot shows a 'Personen toevoegen' (Person Add) form. It includes fields for first name, last name, score (with a numeric up-down control), bet (with a decimal input field), password, birth date (with a dropdown menu), notes (with a text area), and a 'Married' checkbox. At the bottom left, there is a 'Gender' section with a dropdown menu set to 'Woman' and a 'Add' button.

Als laatste aanpassing pakken we het veld Geslacht aan. Hier kan de gebruiker om het even wat intikken terwijl we de keuze via een keuzelijst willen beperkt zien tot Man of vrouw.

Om dit te verwezenlijken moeten we een EditorTemplate toevoegen.

- Voeg in de map *Views/Shared* een folder *EditorTemplates* toe.
- Maak in deze folder een nieuwe view *Geslacht.cshtml*. Kies de template *Empty (without model)*.

```
@using MVC_Voorbeeld3.Models  
@model Geslacht  
@Html.DropDownList("",  
new SelectListItem[] {  
    new SelectListItem() {  
        (1)  
        (2)  
        (3)
```

```
        Text="Mannelijk",  
        Value=Geslacht.Man.ToString(),  
        Selected = Model==Geslacht.Man  
    },  
    new SelectListItem() {  
        Text="Vrouwelijk",  
        Value=Geslacht.Vrouw.ToString(),  
        Selected = Model==Geslacht.Vrouw  
    }  
}  
)
```

Op de eerste lijn voegen we een using-instructie toe zodat we gewoon Geslacht kunnen tikken i.p.v. MVC_Voorbeeld3.Models.Geslacht (1).

Via de Html-helperfunctie laten we een dropdownlist maken (2). Deze bevat een array van `SelectListItems` (3). De eerste eigenschap van een `SelectListItem` is de weer te geven tekst in de dropdownlist (4). De volgende is de bijhorende waarde die moet opgeslagen worden (5) en de laatste eigenschap bepaalt wanneer het keuzeitem al dan niet als geselecteerd moet weergegeven worden (6).

Nu er een `<select>`-tag wordt gebruikt om het geslacht te laten kiezen, worden het label en de lijst naast elkaar weergegeven. Om deze onder elkaar te plaatsen passen we *Site.css* nog een klein beetje aan.

- Voeg naast de selectors `input` en `textarea` ook `select` toe :

```
input , textarea , select {  
    display: block;  
    margin-bottom: 10px;  
}
```

Het uiteindelijke resultaat:

Geslacht	<input type="text" value="Vrouwelijk"/> 
<input type="button" value="toevoegen"/>	

13.6 Meer over validatie

In de voorbije paragrafen konden we al even kennismaken met validatie via de annotations [Required] en [Range]. Hieronder bekijken we nog meer validationmogelijkheden.

13.6.1 StringLength

Met de annotation `[StringLength]` kunnen we een maximum (en minimum) aantal karakters opleggen voor een veld. Enkele voorbeelden :

```
[StringLength(255, ErrorMessage = "Max. {1} tekens voor {0}")]
public string Familienaam { get; set; }

[StringLength(20, MinimumLength = 8, ErrorMessage =
    "Het paswoord bevat min. {2}, max. {1} tekens")]
public string Paswoord { get; set; }
```

In het eerste voorbeeld mogen maximaal 255 tekens ingegeven worden. We kunnen de naam van het veld weergeven met {0}. Bij het tweede voorbeeld geven we via een extra parameter aan dat er ook een minimumlengte is. Deze minimumlengte kunnen we afbeelden in de foutbericht met {2}.

- Voeg in *Persoon.cs* bovenstaande StringLength-annotation (minimum 8 tekens, maximum 20) toe aan het veld *Paswoord* en test het uit.

13.6.2 Regularexpression

Via een regularexpression kan je eisen dat de ingetikte waarde voldoet aan een bepaald patroon :

```
[RegularExpression(@"\d{3}-\d{7}-\d{2}", ErrorMessage = "Rekeningnummer verkeerd")]
public string rekeningNummer { get; set; }
```

Hier vragen we dat het ingetikte rekeningnummer bestaat uit 3 cijfers, een streepje, 7 cijfers, een streepje en 2 cijfers.

13.6.3 Compare

We kunnen twee velden met elkaar laten vergelijken met de annotation [Compare].

```
[Compare("Paswoord", ErrorMessage = "{0} verschilt van {1}")]
public string HerhaalPaswoord { get; set; }
```

Als eerste attribuut geven we het veld op waarmee moet worden vergeleken. In de foutbericht verwijst {0} naar het veld *HerhaalPaswoord*, {1} naar het eerste attribuut "Paswoord".

- Voeg aan de class *Persoon* een extra property *HerhaalPaswoord* toe.
- Zorg er met een annotation voor dat de inhoud van het veld *HerhaalPaswoord* hetzelfde moet zijn als de inhoud van *Paswoord*.
- Neem het veld *HerhaalPaswoord* ook op in de view *Toevoegen.cshtml*.

13.6.4 Een eigen validatie attribuut

Je kan ook zelf een validatie attribuut maken. In het volgende voorbeeld eisen we dat de ingevulde waarde voor een veld *Geboren* in het verleden ligt.

We leggen de voorwaarde op via een annotation :

```
[Verleden(ErrorMessage="Geboortedatum moet in het verleden liggen")]
public DateTime Geboren { get; set; }
```

- Voeg bovenstaande annotation toe in de class *Persoon.cs*.

Om dit te laten werken moeten we eerst een class *VerledenAttribute.cs* toevoegen die erft van ValidationAttribute. De naam van deze class is dus de naam van het validatie-attribuut, gevolgd door 'Attribute'. Je kan deze class bewaren in de root van het project. Als je de class in een folder bewaart dan zal het validatie-attribuut er als volgt uitzien :

```
[NaamVanFolder.Verleden(ErrorMessage="Geboortedatum moet in het verleden liggen")]
```

Opmerking : je mag ook de annotation [NaamVanFolder.VerledenAttribute(...)] gebruiken maar meestal laten we gewoon Attribute weg.

- Voeg een class *VerledenAttribute.cs* toe in de root van het project en geef deze volgende inhoud:

```
using System.ComponentModel.DataAnnotations;
...
public class VerledenAttribute : ValidationAttribute
{
    public override bool IsValid(object value) (1)
    {
        if (value == null) {
            return true; (2)
        }
        if (!(value is DateTime)) {
            return false; (2)
        }
        return ((DateTime)value) < DateTime.Today; (2)
    }
}
```

We overschrijven de method `IsValid()` (1). Deze method krijgt de te valideren waarde binnen als een parameter. Je retourneert `true` als de waarde geldig is, `false` indien niet (2).

- Je kan het nu uitproberen.

13.6.5 Client side validation

Tot nu toe werden de formulieren telkens gevalideerd op de server. We noemen dit server side validation. Dit heeft echter als nadeel dat wanneer de form niet geldig valideert er onnodig netwerkverkeer is. We kunnen dit opvangen door reeds op de client zelf (de browser) met javascript de velden te valideren. Gelukkig doet MVC dit voor ons en moeten we niet zelf javascript gaan schrijven.

Om client side validation te activeren moeten een aantal scriptbestanden opgenomen zijn. Dit gebeurt in twee stappen. De eerste heeft Visual Studio reeds voor ons gezet. In het bestand *BundleConfig.cs* in de map *App_Start* worden een aantal scriptbundles samengesteld. Vooral de eerste twee zijn hier van belang :

```
bundles.Add(new ScriptBundle("~/bundles/jquery").Include(
    "~/Scripts/jquery-{version}.js"));

bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(
    "~/Scripts/jquery.validate*"));
```

Eerst wordt een bundle `"~/bundles/jquery"` samengesteld met de scripts waarvan de naam start met `jquery-` en een versienummer, in het andere statement wordt een bundle `"~/bundles/jqueryval"` samengesteld met alle scripts waarvan de naam begint met `jquery.validate`.

Dit alleen is niet genoeg. Deze scriptbundles moeten nu nog opgenomen worden in onze view(s). Een ideale plaats hiervoor is *_Layout.cshtml*. In dit bestand vind je reeds één instructie die de bundle `"~/bundles/jquery")` opneemt.

- Je voegt nu een gelijkaardig statement toe dat ook de tweede bundle opneemt :

```
...
<body>
    ...
    @Scripts.Render("~/bundles/jquery")
    @Scripts.Render("~/bundles/jqueryval") ← Red arrow
    @Scripts.Render("~/bundles/bootstrap")
    @RenderSection("scripts", required: false)
</body>
...
```

Dit is nog niet alles : er moeten nog twee instellingen gebeuren. Dit kan ofwel voor iedere view apart ingesteld worden ofwel voor de ganse webapplicatie ineens via het bestand *Web.config*.

We beginnen met de tweede optie, het bestand *Web.config* in de root van het project. Als je dit bestand opent dan zal je volgende instellingen terugvinden :

```
...
<configuration>
    <appSettings>
        ...
        <add key="ClientValidationEnabled" value="true" />
        <add key="UnobtrusiveJavaScriptEnabled" value="true" />
    </appSettings>
    ...

```

De client validation is dus standaard enabled voor alle views. We hoeven dus zelf niks in te stellen. Wil je dit niet en wil je enkel voor bepaalde views de client validation activeren dan voeg je in de view (.cshtml-bestand) volgende code toe :

```
@{
    HtmlHelper.ClientValidationEnabled = true;
    HtmlHelper.UnobtrusiveJavaScriptEnabled = true;
}
```

- Nogmaals : je hoeft zelf niks meer toe te voegen want client validation is standaard enabled.
- We proberen nu de validatie uit in de view .../Persoon/VanTotWedde : tik een letter in het veld VanWedde en ga naar het volgende veld. Je krijgt al meteen een foutbericht terwijl je nog niet op Zoeken hebt geklikt.

Gebruik je voor de boven- of ondergrens van de wedde een getal met een decimaal teken dan heb je opnieuw een probleem. Ook hier moeten we opnieuw een bugfix doen.

Het probleem schuilt deze keer in het jqueryval.js script dat je geactiveerd hebt voor de client side validatie. Deze herkent de komma niet. Door een kleine truc kan je de komma omzetten naar een punt.

- Voeg in de folder Scripts een javascriptfile toe en noem deze bijvoorbeeld jqueryFixes.js.
- Tik onderstaande code in deze file :

```
$.validator.methods.range = function (value, element, param) {
    var globalizedValue = value.replace(",","");
    return this.optional(element) || (globalizedValue >= param[0] &&
        globalizedValue <= param[1]);
}
```

```
$.validator.methods.number = function (value, element) {
    return this.optional(element) || /^
    (?:(\d+|\d{1,3})(?:[\s\.,]\d{3})+)(?:[\.,]\d+)?$/ .test(value);
}
```

- Deze bovenstaande code moet nu de originele code uit jqueryval.js overschrijven. Dit kan door in de folder App_Start het bestand BundleConfig.cs te wijzigen. In de method RegisterBundles() wijzig je het tweede statement lichtjes :

```
bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(
    "~/Scripts/jquery.validate*").Include("~/Scripts/jqueryFixes.js"));
```

- Probeer opnieuw uit.

13.7 Samenvatting

- Je kan met Html-helperfuncties forms samenstellen. Je moet dan niet zelf de nodige html-tags noteren.
- Een formulier waarin je wijzigingen aanbrengt op de server maak je als volgt :
 - Je maakt een Model class die de form voorstelt
 - Een GET request toont de form
 - Een POST request verwerkt de gesubmitte form na validatie
- Voor een formulier dat enkel gegevens opvraagt doe je het volgende :
 - Je maakt een Model class die de form voorstelt en ook het resultaat van de Get
 - Een GET request toont de form
 - Een GET request verwerkt de gesubmitte form na validatie
- Voeg validatie toe aan jouw formulier met annotations. Controleer in de controller of het formulier geldig is door *ModelState.IsValid* te evalueren.
- Gebruik de helperfunctie *Html.EditorFor()* om automatisch de gepaste tag te laten genereren afhankelijk van het type van het formulieveld. Je geeft het type aan met een *DataType* annotation.
- Standaard is naast server validation ook client validation ingeschakeld. Zorg er wel voor dat de nodige scripts zijn opgenomen.

13.8 Oefening

In de webapplicatie MVCBierenApplication voeg je onder de lijst met bieren een hyperlink toe naar een formulier waarin je een nieuw bier kan toevoegen.

Verzorg de opmaak van de bierenlijst a.d.h.v. een tabel.

Voeg ook validatie toe : de naam van het bier is maximaal 20 tekens lang en is verplicht in te vullen, het alcoholpercentage moet tussen 0 en 15 liggen.

14 Data uit een database lezen met het entity framework

In de voorbije hoofdstukken hebben we telkens met statische data gewerkt. In dit hoofdstuk zullen we data uit SQL Server gebruiken.

14.1 De database voorbereiden

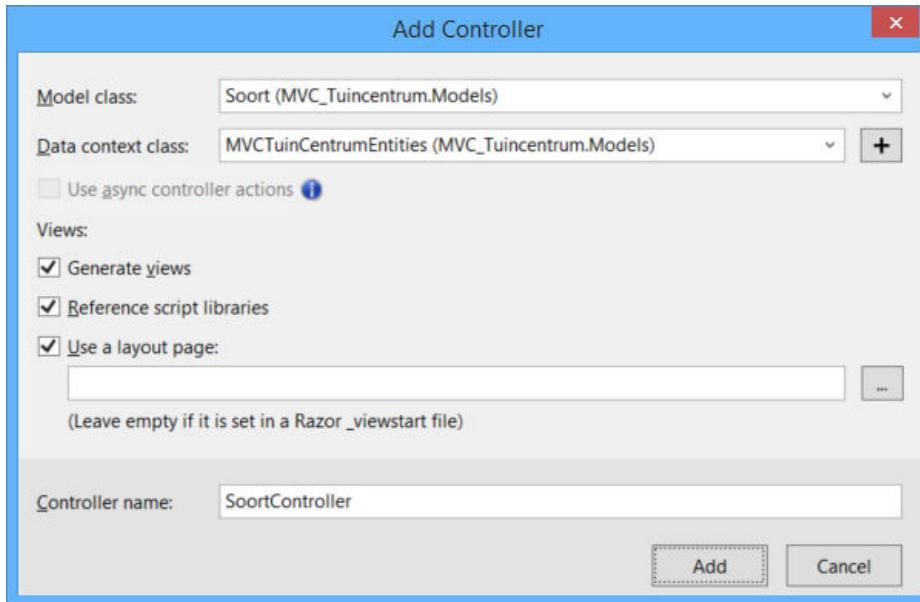
In de folder met oefenbestanden vind je een sql-script met de naam *createMVCTuincentrum.sql*.

- Open dit document in de SQL Server Management Studio en voer het script uit. Je zou dan over een database MVCTuinCentrum moeten beschikken met de tabellen Planten, Soorten en Leveranciers.
- Maak een nieuwe ASP.NET Webapplicatie en noem de applicatie *MVC_Tuincentrum*. Kies de *MVC* template en wijzig de authentication in *No Authentication*.
- Voeg in de Server Explorer een Data Connection toe naar de database MVCTuinCentrum.
- In de folder *Models* voeg je via *Add – New Item...* een *ADO.NET Entity Data Model* toe. Noem het *Tuincentrum*.
- Kies in het dialoogvenster voor *EF Designer from database*. Kies de data connection *MVCTuincentrum*, selecteer alle tables en klik op *Finish*. Een script maakt nu de nodige classes aan in de folder *Models*.
- Wijzig de namen van de entity types naar enkelvoud : dus *Soort*, *Plant* en *Leverancier* i.p.v. *Soorten*, *Planten* en *Leveranciers*. Vooraleer je de entity *Soorten* kan hernoemen naar *Soort* zal je het veld *Soort* moeten hernoemen naar *Naam*. De navigation properties *Leveranciers* en *Soorten* van de entity *Plant* hernoem je naar hun enkelvoudsvorm.
- Kies in het menu voor *Save All*. De nodige classes worden opnieuw aangemaakt, deze keer in enkelvoudsvorm.
- Build de applicatie.

14.2 Scaffolding

Tot nu toe hebben we bij het aanmaken van een controller in het dialoogvenster *Add Controller* steeds gekozen voor een empty controller. Visual Studio beschikt echter over een ingenieus systeem dat niet alleen een controller aanmaakt, maar voor jou ook alle nodige lees-, wijzig-, verwijder- en toevoeg-actions mét bijhorende views aanmaakt. Dit wordt scaffolding genoemd, in het Nederlands te vertalen als 'in de steigers zetten'.

- Voeg een nieuwe controller toe en kies deze keer de template *MVC 5 Controller with views, using Entity Framework*. Klik op *Add*.
- Als Model class kies je de class *Soort* en als Data context class *MVCTuinCentrumEntities*. Wijzig de naam van de controller in *SoortController*. Klik op *Add*.



- Je kan het resultaat al eens bekijken door het project te starten en te browsen naar .../Soort.

Je krijgt een Index-pagina te zien met een lijst van alle soorten. Per soort is er een hyperlink naar een Edit, Details en Delete pagina. Bovenaan kan je via een hyperlink naar een pagina Create een nieuwe soort toevoegen.

Index		
Create New		
Naam	MagazijnNr	
1-jarig	2	Edit Details Delete
2-jarig	3	Edit Details Delete
Bol	1	Edit Details Delete
Boom	3	Edit Details Delete
Heester	4	Edit Details Delete
Heide	2	Edit Details Delete
Klim	2	Edit Details Delete
Kruid	3	Edit Details Delete
Vast	1	Edit Details Delete
Water	4	Edit Details Delete

- Probeer nu een soort te wijzigen of aan te maken, of de details op te vragen van een soort.

Opmerking : als je een nieuwe soort toevoegt mag de naam van de soort maar maximaal 10 tekens lang zijn. Tik je er meer in dan krijg je een fout. Voorlopig is er immers nog geen validatie toegepast.

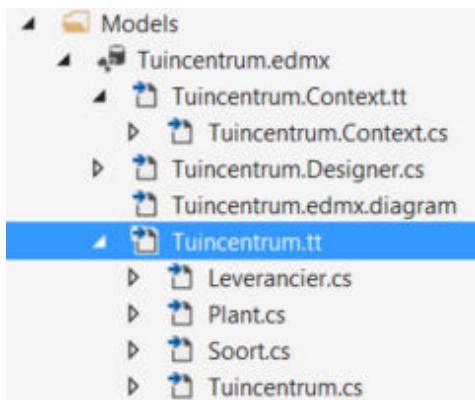
In de source van de SoortController zie je de verschillende actions met erboven in commentaar de url waarmee je ze kan aanspreken. De database spreekt je aan via een private variabele *db* van het type MVCTuinCentrumEntities :

```
private MVCTuinCentrumEntities db = new MVCTuinCentrumEntities();
```

- Je kan nu ook voor de planten en voor de leveranciers een controller maken met scaffolding.

14.3 Buddy classes

Als je in de Solution Explorer *Tuincentrum.edmx* openklapt, dan zie je onder *Tuincentrum.tt* de classes *Soort*, *Leverancier*, *Plant* en *Tuincentrum*.



Deze bestanden werden door Visual Studio aangemaakt en worden best door ons niet aangepast (zie commentaar in de bovenste lijnen van deze codebestanden). Elke wijziging die we aanbrengen wordt immers door VS opnieuw overschreven telkens we in de edmx een aanpassing doen en de edmx opslaan.

Wanneer we nu velden willen valideren stelt er zich een probleem. We kunnen de nodige annotations niet toevoegen want de classes werden door Visual Studio gegenereerd en mogen niet aangepast worden. We lossen dit op door 'buddy classes' te gebruiken.

- Voeg in de folder *Models* een class *PlantProperties* toe. In die class voegen we een property *VerkoopPrijs* toe met een rangevalidator via de annotation `[Range]`.

```
using System.ComponentModel.DataAnnotations;  
...  
public class PlantProperties  
{  
    [Range(0, 1000)]  
    public decimal VerkoopPrijs { get; set; }  
}
```

We kunnen deze 'buddy class' nu aan de class *Plant* uit het designerbestand koppelen via een nieuwe partial class *Plant*.

- Voeg in de folder *Models* een nieuwe class toe. We kunnen deze class niet ook nog eens *Plant.cs* noemen want er staat al een dergelijke class in deze folder. Noem de extra class daarom *PlantUitbreiding.cs*.
- In de code van deze class voeg je het keyword *partial* toe en wijzig je de naam van de class naar *Plant*. Voeg ook een annotation toe :

```
using System.ComponentModel.DataAnnotations;  
...  
[MetadataType(typeof(PlantProperties))]  
public partial class Plant  
{  
}
```

Via het keyword partial geef je aan dat dit een partial class is. Met de annotation `[MetadataType(typeof(PlantProperties))]` duid je aan in welke class de bijhorende validatieregels zitten.

- Browse nu naar .../Plant en klik bij één van de planten op *Edit*. Probeer nu maar eens de verkoopprijs van een plant op bijvoorbeeld 2000 te zetten. De validatie doet zijn werk.

Ook hier is er opnieuw een probleem bij het gebruik van getallen met decimalen en moeten we een bugfix doen.

- Je voegt opnieuw een class DecimalModelBinder.cs toe in de root van het project.
- Neem uit paragraaf 13.4.4. de code over voor DecimalModelBinder.cs.
- In Global.asax.cs voeg je de modelbinder toe in de method Application_Start() :


```
ModelBinders.Binders.Add(typeof(decimal), new DecimalModelBinder());
ModelBinders.Binders.Add(typeof(decimal?), new DecimalModelBinder());
```
- Je brengt nu ook de clientvalidatie in orde door in _Layout.cshtml de bundel jqueryval toe te voegen :


```
@Scripts.Render("~/bundles/jqueryval")
```
- Je voegt in de folder Scripts opnieuw een bestand jqueryFixes.js toe met dezelfde inhoud als daarnet.
- Tenslotte wijzig je de code in RegisterBundles() in BundleConfig.cs :


```
bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(
    "~/Scripts/jquery.validate*").Include("~/Scripts/jqueryFixes.js"));
```
- Je kan nu ook prijzen met decimalen intikken.

14.4 Een zoekform toevoegen

We voegen nu aan de applicatie MVC_Tuincentrum een formulier toe waarmee we een soort opzoeken. We beginnen met het aanmaken van een class voor de form.

- Voeg in de folder *Models* een class *ZoekSoortForm.cs* aan.

Het is de bedoeling dat we in een textbox het begin van de naam van een soort in kunnen tikken.

- Vul de class ZoekSoortForm als volgt aan :

```
using System.ComponentModel.DataAnnotations;
...
public class ZoekSoortForm
{
    [Display(Name = "Begin soortnaam")]
    [Required(ErrorMessage = "Verplicht")]
    public string beginNaam { get; set; }
}
```

- Build de solution.

- In de SoortController voeg je nu een action toe :

```
// GET: /Soort/Zoekform
public ViewResult ZoekForm()
{
    return View(new ZoekSoortForm());
}
```

- Maak een bijhorende empty view (without model).
- Voeg volgende code toe :

```
@model MVC_Tuincentrum.Models.ZoekSoortForm
@{
    ViewBag.Title = "ZoekForm";
}
<h2>Soorten zoeken op begin naam</h2>
@using (Html.BeginForm("BeginNaam", "Soort", FormMethod.Get))
{
    @Html.LabelFor(m=>m.beginNaam)
    @Html.ValidationMessageFor(m=>m.beginNaam)
    @Html.EditorFor(m=>m.beginNaam)
    <input type="submit" value="zoeken" />
}
```

We laten de functie `Html.BeginForm()` de nodige html form-tags aanmaken en geven aan dat een action `BeginNaam` uit de SoortController de verwerking van de form via een Get-request voor zijn rekening zal nemen.

De form zelf bestaat uit een label, een input element dat hoort bij een string (een textbox dus) en een validationmessage.

- Voeg in de SoortController de action `BeginNaam` toe, die de form verwerkt :

```
public ViewResult BeginNaam(ZoekSoortForm form)
{
    if(this.ModelState.IsValid)
    {
        var query=from soort in db.Soorten
                  where soort.Naam.StartsWith(form.beginNaam)
                  orderby soort.Naam
                  select soort;
        var soorten=query.ToList();
    }
}
```

Vooraleer we deze code verder aanvullen verduidelijken we al even wat hier staat. De action `BeginNaam` krijgt de ingevulde waarde uit het formulier binnen in de vorm van een `ZoekSoortForm`-object met de naam `form`. Wanneer er geen validatieproblemen zijn – de `ModelState.IsValid` is true – dan zoeken we met een LINQ-query alle soorten op waarvan de naam begint met de in het formulier ingetikte waarde (`form.beginNaam`). We zetten het resultaat om naar een List en stoppen dit in een variabele `soorten`.

Het is de bedoeling dat we onder het formulier het resultaat van onze zoekopdracht weergeven. In de class *ZoekSoortForm.cs* voegen we een property *Soorten* toe van het type *List<Soort>* toe :

```
public class ZoekSoortForm
{
    [Display(Name = "Begin soortnaam:")]
    [Required(ErrorMessage = "Verplicht")]
    public string beginNaam { get; set; }
    public List<Soort> Soorten { get; set; }
}
```

In de *BeginNaam* action in de controller bewaren we het resultaat van de zoekopdracht nu in deze extra property. Daarna activeren we de action *ZoekForm* met als extra parameter *form* waarin de zoekterm en het resultaat van de zoekopdracht zit.

- Vul in de SoortController de code van de action *BeginNaam* als volgt verder aan :

```
...
var soorten=query.ToList();
form.Soorten = soorten;
}
return View("ZoekForm", form);
```

We passen nu nog de zoekform aan zodat gevonden soorten in een tabel worden weergegeven.

- Voeg onderstaande code toe aan *ZoekForm.cshtml* :

```
...
@if (Model.Soorten != null)
{
    <table>
        <thead>
            <tr>
                <th>Soort</th>
                <th>Magazijnnr</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var soort in Model.Soorten)
            {
                <tr>
                    <td>@soort.Naam</td>
                    <td>@soort.MagazijnNr</td>
                </tr>
            }
        </tbody>
    </table>
}
```

- Browse nu naar .../Soort/ZoekForm en test uit.

We kunnen nog een kleine verbetering aanbrengen aan onze code. We hebben in de controller in de action *BeginNaam* de opzoeking gedaan aan de hand van een LINQ-query. Beter zou zijn een aparte method, bijvoorbeeld *FindByBeginNaam* te schrijven in een service *SoortService* en in de action *BeginNaam* deze service-method op te roepen.

- Voeg aan het project een folder *Services* toe en voeg er een class *SoortService* aan toe :

```
using MVC_Tuincentrum.Models;
...
public class SoortService
{
    public List<Soort> FindByBeginNaam(string beginNaam)
    {
        using (var db = new MVCTuinCentrumEntities())
        {
            var query = from soort in db.Soorten
                        where soort.Naam.StartsWith(beginNaam)
                        orderby soort.Naam
                        select soort;
            var soorten = query.ToList();
            return soorten;
        }
    }
}
```

De method *FindByBeginNaam* zoekt via een LINQ-query de soorten op waarvan de naam begint met de inhoud van de parameter *beginNaam*.

- Opdat de SoortController deze code nu zou kunnen gebruiken voeg je bovenaan SoortController.cs deze regel toe :

```
private SoortService soortService = new SoortService();
```

We voegen dus een private variabele *soortService* toe. We zeggen dat de controller een dependency heeft naar de service. In de action *BeginNaam* kunnen we de service nu oproepen.

- Wijzig de code :

```
public ViewResult BeginNaam(ZoekSoortForm form)
{
    if(this.ModelState.IsValid)
        form.Soorten = soortService.FindByBeginNaam(form.beginNaam);
    return View("ZoekForm", form);
}
```

14.5 De applicatie publiceren

In hoofdstuk 8 hebben we al even een webapplicatie gepubliceerd naar de lokale IIS op jouw toestel. We gaan dit nu opnieuw doen voor de tuincentrum webapplicatie.

14.5.1 Een nieuwe SQL Server user

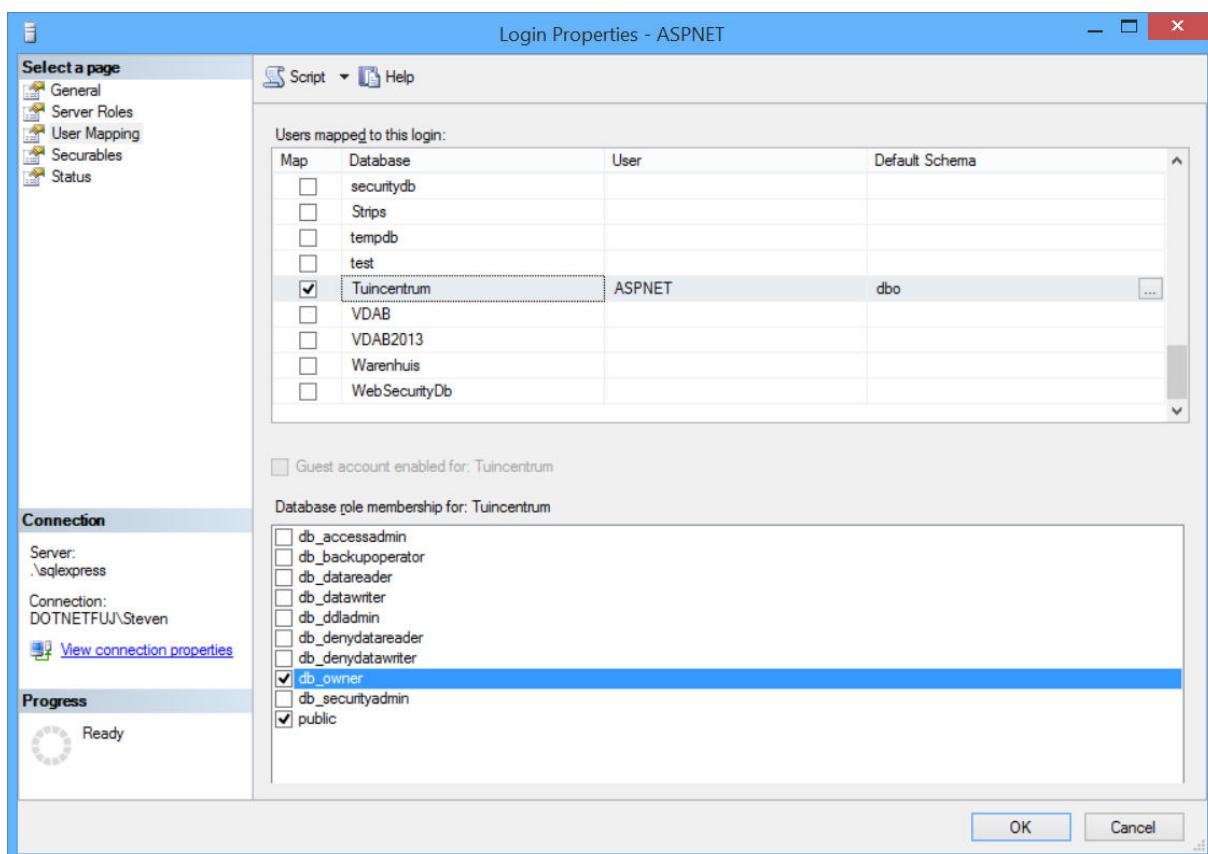
Bij het uittesten van de webapplicatie vanuit Visual Studio hebben we Windows Authentication gebruikt om de database te benaderen. Dit kunnen we niet meer doen wanneer we onze webapplicatie publiceren op een ‘echte’ webserver. Een gebruiker die via het internet naar onze database wil gaan zal niet gekend zijn door onze SQL Express. Daarom zullen we SQL Server Authentication gebruiken.

- Start SQL Server Management Studio en connecteer met jouw instance van SQL Express (dit mag via jouw windows account, zolang je maar administrator rechten hebt).
- Klap het onderdeel Security open en daarbinnen het onderdeel Logins.

- Klik met de rechtermuistoets op Logins en kies New Login...
- Geef als Login name bijvoorbeeld ASPNET, kies voor SQL Server authentication, tik een voldoende veilig wachtwoord en zorg er zeker voor dat je de optie *User must change password at next login* uitvinkt.
- Klik op OK.

Je geeft deze user nu het recht om de database Tuincentrum te beheren.

- Klik in het onderdeel Security / Logins met de rechtermuistoets op de nieuwe usernaam ASPNET en kies Properties.
- Aan de linkerkant kies je voor User Mapping. Rechts krijg je dan de lijst met databases te zien. Je selecteert de database Tuincentrum. Onderaan vink je dan de optie *db_owner* aan. Daarna klik je op OK.

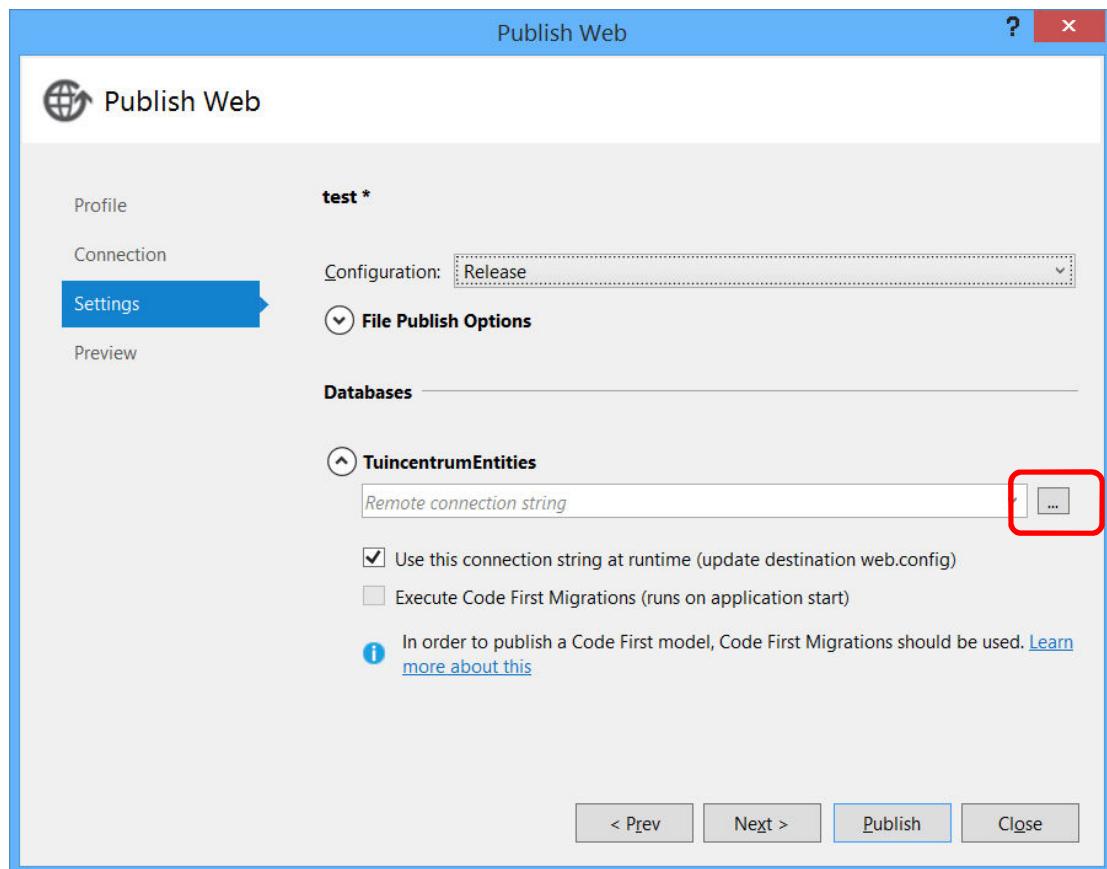


14.5.2 De webapplicatie publiceren

We kunnen de webapplicatie nu publiceren.

- Klik in de Solution Explorer met de rechtermuistoets op het project en kies Publish. Zoals je weet moet je daarvoor VS hebben opgestart met Administrator rechten.
- Als publish target kies je voor Custom.
- Geef het profiel een naam, bijvoorbeeld *LocalIIS*.
- Op het tabblad Connection tik je bij Server *localhost* en als Site name tik je bijvoorbeeld *Default Web Site/Tuin*. Kies Next.

- Op het tabblad Settings klik je naast de plaats waar je de *Remote connection string* kan ingeven op de 3 puntjes.



- In het dialoogvenster *Destination Connection String* vul je de naam van de SQL Server in : .\sqlexpress. Vervolgens kies je voor *Use SQL Server Authentication* en vul je de user ASPNET en bijhorend wachtwoord in.
- Selecteer tenslotte onderaan de database Tuincentrum en klik op OK.
- Klik op Publish en probeer de webapplicatie uit door de browsen naar <http://localhost/tuin>.

14.6 Samenvatting

- Via scaffolding kan je heel makkelijk automatisch een controller laten aanmaken voor de classes die horen bij de tabellen uit jouw database.
- Aangezien we geen wijzigingen mogen aanbrengen in deze automatisch aangemaakte classes moeten we validatieregels toepassen in buddy classes. Deze koppelen we dan aan de gegenereerde classes via partial classes.

14.7 Oefening

Zorg ervoor dat de gegevens in de webapplicatie MVCBierenApplication uit de databank Bieren komt. Gebruik het sql-script *createMVCBieren.sql* om de bierendatabase aan te maken.

15 Upload

In dit hoofdstuk voorzien we de Tuincentrumapplicatie van de mogelijkheid een foto up te laden die hoort bij een plant.

We houden het vrij eenvoudig : via een formulier zal de gebruiker via een dialoogvenster een foto kunnen selecteren op de schijf. Deze foto wordt dan geüpload en bewaard in de folder *Images/Fotos* met als bestandsnaam het nummer van de plant en extensie ".jpg". De applicatie zal dus enkel werken voor .jpg-bestanden.

In de overzichtsview van de planten zal de foto van de plant worden weergegeven indien beschikbaar.

15.1 Verwijzing naar een uploadformulier

We passen eerst de view *Index.cshtml* in de folder *Views/Plant* aan zodat er per plant een extra hyperlink naar een upload-action te zien is. Deze link krijgt als parameter het nummer van de plant.

- Wijzig de code in *Index.cshtml* in de folder *Views/Plant* als volgt :

```
...
<td>
    @Html.ActionLink("Edit", "Edit", new { id=item.PlantNr }) |
    @Html.ActionLink("Details", "Details", new { id=item.PlantNr }) |
    @Html.ActionLink("Delete", "Delete", new { id=item.PlantNr }) |
    @Html.ActionLink("Foto uploaden", "UpLoaden", new { id = item.PlantNr })
</td>
</tr>
...
...
```

Index						
Create New						
PlantNr	Naam	Naam	Naam	Kleur	VerkoopPrijs	
1	Rododendron	Heester	Mooiweer	Rood	7,25	Edit Details Delete Foto uploaden
2	Sering	Heester	Mooiweer	Paars	7,25	Edit Details Delete Foto uploaden
3	Kruisdistel	Vast	Mooiweer	Blauw	1,12	Edit Details Delete Foto uploaden
4	Vuurwerkboom	Heester	Paradijsboom	Wit	1,80	Edit Details Delete Foto uploaden

15.2 Het uploadformulier

In de *PlantController* maken we nu een extra method *Uploaden()* :

```
[HttpGet]
public ViewResult Uploaden(int id)
{
    return View(id);
}
```

Deze method heeft een int-parameter : het plantnummer en geeft dit door aan de bijhorende view.

- Voeg een view toe aan de actionmethod *Uploaden* en voeg onderstaande code toe :

```
@model int
{@
    ViewBag.Title = "Uploaden";
}
<h2>Foto uploaden</h2>
@using (Html.BeginForm("FotoUpload", "Plant", new { id=Model },
    FormMethod.Post, new { enctype="multipart/form-data"}))
{
    <input type="file" name="Foto" />
    <input type="submit" value="Uploaden" />
}
```

We bekijken even de 5 parameters van de functie *Html.BeginForm()* : "FotoUpload" is de naam van de method uit de "Plant"-controller die de verwerking van het formulier voor zijn rekening zal nemen. We geven het plantnummer door via een nieuw onbenoemd object waarvan we de property id de int-waarde uit het Model geven. We gebruiken een Post-request en de laatste parameter geeft aan dat er een file of binaire data wordt doorgestuurd.

Het formulier zelf bestaat verder uit een knop om een bestand te kiezen en een knop om te submitten.

15.3 Het formulier verwerken

In de PlantController maken we nu een method *FotoUpload* aan.

- Voeg onderstaande actionmethod toe in *PlantController.cs* :

```
using System.IO;
...
[HttpPost]
public ActionResult FotoUpload(int id)
{
    if (Request.Files.Count > 0) { (1)
        var foto = Request.Files[0];
        var absoluutPadNaarDir = (2)
            this.HttpContext.Server.MapPath("~/Images/Fotos");
        var absoluutPadNaarFoto =
            Path.Combine(absoluutPadNaarDir, id + ".jpg"); (3)
        foto.SaveAs(absoluutPadNaarFoto); (4)
    }
    return RedirectToAction("Index"); (5)
}
```

- Maak in de root van de webapplicatie een nieuwe folder *Images* aan met daarin een folder *Fotos*.

We controleren hier eerst of er in het dialoogvenster een file is aangeduid (1). Als dat zo is dan halen we de foto-gegevens uit de request (2), stellen naam en pad van de te bewaren foto samen (3) en bewaren tenslotte de foto in de map *Images/Fotos* (4). Daarna volgt er een redirection naar de index-pagina van de PlantController (5).

15.4 Foto weergeven in detailspagina

De foto is nu gekozen en geüpload maar nog nergens zichtbaar. We zullen de foto laten zien in de view *Details* in de folder *Views/Plant*.

- Onderaan *Details.cshtml* voeg je onderstaande vetgedrukte code toe :

```
...
</dd>
<dt>Afbeelding</dt>
<dd>
    
</dd>
</dl>
</div>
...

```

We proberen het even uit.

- Start het project en browse naar .../Plant/Index
- Klik op de hyperlink *Foto uploaden* naast de eerste plant, Rododendron.
- Kies *Bestand kiezen* en selecteer uit de map met oefenbestanden de afbeelding *rhododendron.jpg* en klik op *Uploaden*. De folder *Images/Fotos* bevat nu een bestand 1.jpg.
- Op de indexpagina klik je op de hyperlink *Details* van de plant Rododendron

Resultaat :

The screenshot shows a web page titled 'Details' for a plant named 'Rododendron'. The page has a header 'Plant' and contains the following data:

	Naam	Rododendron
Kleur	Rood	
VerkoopPrijs	7,25	
Naam	Mooiweer	
Naam	Heester	
Afbeelding		

At the bottom of the page are links 'Edit' and 'Back to List'.

Opmerkingen :

- De maximale grootte van een te uploaden bestand is 4MB. Je kan dit wijzigen naar bijvoorbeeld 5MB door in *Web.config* het onderstaande toe te voegen :

```
...
<system.web>
    <httpRuntime maxRequestLength="5000"/>
...

```
- Indien je het uploaden uitprobeert op een 'echte' IIS (en niet de ingebouwde IIS in VS) dan mag je niet vergeten de folder /Images/Fotos aan te maken en schrijfrechten toe te kennen op die folder aan de groep IIS_IUSRS. Je doet dit door in de verkenner met de rechtermuistoets te klikken op de folder en Eigenschappen te kiezen. Op het tabblad Beveiliging kies je de groep IIS_IUSRS en je klikt op Bewerken. Je vinkt in de kolom Toestaan de machtiging Schrijven aan en je klikt op OK.

16 Globalization

Je hebt vast al gezien dat in veel websites de gebruiker een taal kan kiezen waarin de webpagina's worden weergegeven. In onze Tuincentrumwebapplicatie gaan we dit ook doen.

16.1 Taal en Locale

De taal die de webpaganabezoeker gebruikt wordt uitgedrukt met een ISO-code. Hieronder vind je een lijstje van courante talen en hun ISO-code :

nl	Nederlands
fr	Frans
en	Engels

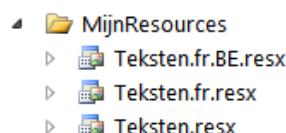
Afhankelijk van het land waarin men woont worden er ook verschillende getal- en datumnotaties gebruikt. De combinatie taal/land wordt een locale genoemd. Het is op basis van deze locale dat we bepaalde teksten zullen gebruiken in de webpagina of getallen en datums zullen opmaken.

Voorbeelden van enkele locales :

nl-BE	Nederlands in België
nl-NL	Nederlands in Nederland
fr-BE	Frans in België
fr-FR	Frans in Frankrijk

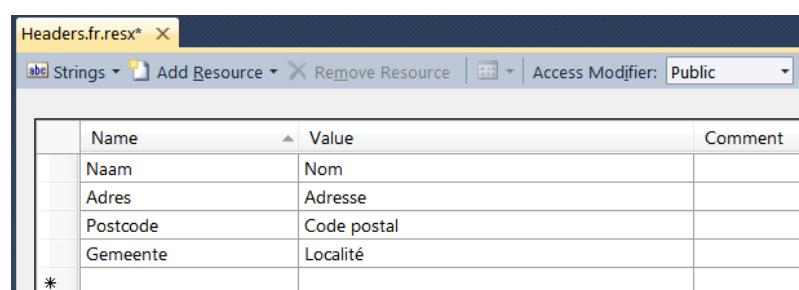
16.2 Resourcefiles

Teksten voor diverse talen of locales sla je op in resourcefiles (.resx), bijvoorbeeld in een map *MijnResources*, zoals in onderstaande afbeelding.



In dit geval bevat de folder *MijnResources* een resourcefile met teksten voor de locale fr-BE (*Teksten.fr.BE.resx*). Dus Franse teksten voor Belgische websitebezoekers. Er is ook een resourcefile voor alle Franstalige bezoekers (*Teksten.fr.resx*) buiten België. Voor alle andere gebruikers worden de teksten uit *Teksten.resx* gebruikt. De meest specifieke locale heeft dus voorrang.

De resourcefiles bevatten entries die bestaan uit een Name en een Value :



Name	Value	Comment
Naam	Nom	
Adres	Adresse	
Postcode	Code postal	
Gemeente	Localité	

Het is de bedoeling dat je nu in de diverse resourcefiles een entry opneemt met de Name 'Naam', een entry voor 'Adres', enz. Bovenaan rechts zie je een Access Modifier. Deze zet je op Public.

16.3 Verschillende teksten en datum- en getalnotaties gebruiken

Wanneer je een website maakt die diverse talen ondersteunt moet je er dus voor zorgen dat de teksten uit de juiste resourcefile en de correcte datum- en getalnotaties worden gebruikt.

De teksten instellen gebeurt door `Thread.CurrentCulture` in te stellen. Voor de datum- en getalnotaties stel je `Thread.CurrentThread.CurrentCulture` in. De code hiervoor ziet er bijvoorbeeld als volgt uit :

```
var culture = "fr-BE";
Thread.CurrentThread.CurrentCulture = new CultureInfo(culture);
```

Je kan de `Current(UI)Culture` automatisch laten instellen. De browser geeft in elke request de taal of locale mee in een header `Accept-Language`. Je kan in `Web.config` instellen dat `CurrentUICulture` en `CurrentCulture` automatisch moeten worden ingesteld met de waarde uit die `Accept-Language` header :

```
...
<system.web>
  <globalization culture="auto" uiCulture="auto" enableClientBasedCulture="true"/>
...
...
```

16.4 Voorbeeld

16.4.1 Het navigatiemenu

Om te kunnen navigeren naar de diverse onderdelen van onze Tuincentrumapplicatie passen we de inhoud van de homepage en ook het menu in de hoofding aan.

- Pas in `_Layout.cshtml` de hyperlinks aan :

```
...
</button>
@Html.ActionLink("Tuincentrum", "Index", "Home", null,
    new { @class = "navbar-brand" })
</div>
<div class="navbar-collapse collapse">
  <ul class="nav navbar-nav">
    <li>@Html.ActionLink("Home", "Index", "Home")</li>
    <li>@Html.ActionLink("Planten", "Index", "Plant")</li>
    <li>@Html.ActionLink("Soorten", "Index", "Soort")</li>
    <li>@Html.ActionLink("Leveranciers", "Index", "Leverancier")</li>
  </ul>
</div>
</div>
<div class="container body-content">
  <img class="extramargin" src "~/Images/tuincentrum.png" alt="tuincentrumlogo"/>
  @RenderBody()
  <hr />
  <footer>
    <p>&copy; @DateTime.Now.Year - .NET Ontwikkelaar C#</p>
  </footer>
</div>
...
```

- Kopieer de afbeelding voor het tuincentrum-logo naar de map `Images`. Je vindt het in de map met oefenbestanden.

- Vervang alle code in de view *Index.cshtml* uit de folder *Views\Home* door het onderstaande :

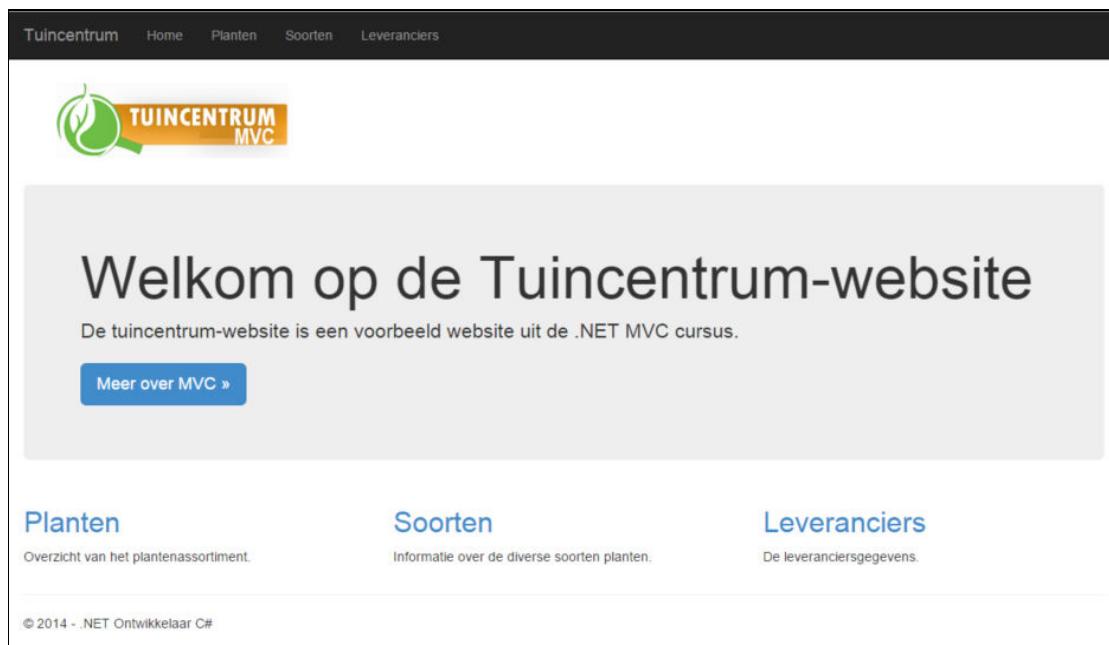
```
@{
    ViewBag.Title = "Index";
}
<div class="jumbotron">
    <h1>Welkom op de Tuincentrum-website</h1>
    <p class="lead">De tuincentrum-website is een voorbeeld website uit de .NET
MVC cursus.</p>
    <p><a href="http://asp.net/mvc" class="btn btn-primary btn-lg">Meer over MVC
&raquo;</a></p>
</div>

<div class="row">
    <div class="col-md-4">
        <h2>@Html.ActionLink("Planten", "Index", "Plant")</h2>
        <p>Overzicht van het plantenassortiment.</p>
    </div>
    <div class="col-md-4">
        <h2>@Html.ActionLink("Soorten", "Index", "Soort")</h2>
        <p>Informatie over de diverse soorten planten.</p>
    </div>
    <div class="col-md-4">
        <h2>@Html.ActionLink("Leveranciers", "Index", "Leverancier")</h2>
        <p>De leveranciersgegevens.</p>
    </div>
</div>
```

- Voeg in *Site.css* volgende css-code toe om de figuur van extra marge te voorzien.

```
.extramargin
{
    margin : 25px;
}
```

- Bekijk al eens het resultaat :



16.4.2 Meertalige homepagina

We gaan nu de welkomsttekst op de homepagina, afhankelijk van de taal die ingesteld is in de browser, laten verschijnen in het Frans of het Nederlands.

Resources bewaren we best ofwel in een folder *App_LocalResources* ofwel *App_GlobalResources*. Het verschil tussen beide soorten resources is dat local resources gekoppeld zijn aan één webpagina terwijl global resources in de ganse webapplicatie kunnen gebruikt worden. In deze cursus maken we enkel gebruik van global resources.

- Klik in de Solution Explorer met de rechtermuistoets op het project en kies *Add – Add ASP.NET Folder – App_GlobalResources*.
- Aan deze folder voeg je een nieuwe resourcefile toe via *Add – New Item... - Resources File*. En noem deze *Teksten.resx*.
- Voeg een entry toe met als Name : *Welkom* en Value : *Welkom op de Tuincentrum-website*
- Voeg een tweede resourcefile toe *Teksten.fr.resx* :
- Voeg ook hier een entry toe met als Name : *Welkom* en als Value : *Bienvenue sur le site Tuincentrum*

We maken nu gebruik van deze teksten op de indexpagina van de homecontroller.

- In de folder *Views/Home* wijzig je *Index.cshtml* als volgt :

```
...
<div class="jumbotron">
<h1>@Resources.Teksten.Welkom</h1>
...

```

Via `@Resources.Teksten.Welkom` verwijst je naar de entry met key `Welkom` in de resourcefile `Teksten`, of `Teksten.fr` of...

Om nu de juiste taalversie van de tekst te laten verschijnen voeg je in de root van het project in *Web.config* het volgende toe :

```
...
<system.web>
  <globalization culture="auto" uiCulture="auto" enableClientBasedCulture="true"/>
...

```

- Start de website in IE(11). Je krijgt normaal de Nederlandse tekst. Klik in IE via  (*Tools/Extra - Internet options (Internetopties)*) op tabblad *General (Algemeen)* op knop *Languages (Talen)*.
- Voeg na een klik op de knop *Set Language Preferences (Taalvoorkeuren instellen)* de taal *French (Belgium) [fr-BE] (français (Belgique))* toe of als deze er al staat selecteer die en zorg ervoor dat deze bovenaan komt te staan door de knop *Move up (Omhoog)* te gebruiken.
- Sluit het dialoogvenster, klik tot twee keer toe op *OK* en refresh de pagina. Je zou de Franse tekst moeten zien.
- Herstel terug naar Nederlandse instellingen. Refresh en je ziet terug de Nederlandse tekst.

- In Chrome kies je *Instellingen – Geavanceerde instellingen weergeven...* Vervolgens klik je onder de rubriek *Talen* op de knop *Taal- en invoerinstellingen...* Daar kan je extra talen zoals Frans toevoegen en desgewenst naar boven schuiven.



Browsers worden regelmatig geüpdatet. Het is dus mogelijk dat de manier waarop je de taalinstellingen wijzigt een klein beetje anders is.

16.4.3 De meertaligheid instellen via knoppen

We kunnen de gebruiker ook de gewenste taal laten instellen via twee knoppen op onze pagina zonder dat de browserinstellingen moeten gewijzigd worden.

- Voeg in *_Layout.cshtml* onderstaande vetgedrukte code toe net onder de reeds bestaande navigatiebalk :

```
...
<div class="navbar-collapse collapse">
    <ul class="nav navbar-nav">
        <li>@Html.ActionLink("Home", "Index", "Home")</li>
        <li>@Html.ActionLink("Planten", "Index", "Plant")</li>
        <li>@Html.ActionLink("Soorten", "Index", "Soort")</li>
        <li>@Html.ActionLink("Leveranciers", "Index", "Leverancier")</li>
    </ul>
    <ul class="nav navbar-nav navbar-right">
        <li>
            <a href='@Url.Action(ViewBag.Title, new { culture = "nl-BE" })'>
                <img src "~/Images/nederlands.png" alt="nederlands" />
            </a>
        </li>
        <li>
            <a href='@Url.Action(ViewBag.Title, new { culture = "fr-BE" })'>
                <img src "~/Images/francais.png" alt="français" />
            </a>
        </li>
    </ul>
</div>
...

```

- Kopieer de afbeeldingen *nederlands.png* en *francais.png* naar de folder *Images*.

Om te vermijden dat een klik op één van de taal-knoppen ons telkens naar de homepage zou brengen, halen we de action uit *ViewBag.Title*. We geven als extra parameter de gewenste locale mee. Dit zorgt ervoor dat de link er bijvoorbeeld als volgt uitziet : .../Leverancier?culture=nl-BE. De bootstrap css-class *navbar-right* zorgt ervoor dat de taal-navigatiebalk aan de rechterkant wordt getoond.

We moeten nu code toevoegen die de culture uit de request haalt en deze instelt in de huidige Culture en UICulture.

- Open in de folder *Views* het bestand *_ViewStart.cshtml*. Wat het nut is van deze view leerden we reeds in paragraaf 3.7 Templates. Vervolledig deze view met onderstaande code :

```
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
```

```

        var culture = Request["culture"]; (1)
        if (!culture.IsEmpty()) { (2)
            Session["culture"] = culture;
        }
        if (culture.IsEmpty()) { (3)
            culture = (string)Session["culture"];
        }
        if (!culture.IsEmpty()) { (4)
            Culture = UICulture = culture;
        }
    }
}

```

In bovenstaande code kijken we eerst of de request een string met de naam culture bevat (1). Is dat inderdaad zo dan bewaren we de inhoud op sessionscope (2). Het voordeel hiervan is dat we dan later de culture kunnen instellen ook al staat die niet in de request en dat gebeurt in (3). Hebben we een culture opgehaald (uit request of sessionscope) dan stellen we die in.

Je kan dit al eens uitproberen. Daarna kunnen we ook het menu op de homepagina aanpakken.

- Voeg in de 2 resourcefiles extra entries toe :

De Nederlandse versie :

Name : <i>Leveranciers</i>	Value : <i>Leveranciers</i>
Name : <i>Planten</i>	Value : <i>Planten</i>
Name : <i>Soorten</i>	Value : <i>Soorten</i>

De Franse versie :

Name : <i>Leveranciers</i>	Value : <i>Fournisseurs</i>
Name : <i>Planten</i>	Value : <i>Plantes</i>
Name : <i>Soorten</i>	Value : <i>Genres</i>

- In *_Layout.cshtml* wijzigen we nu de links :

```

<ul class="nav navbar-nav">
    <li>@Html.ActionLink("Home", "Index", "Home")</li>
    <li>@Html.ActionLink(Resources.Teksten.Planten, "Index", "Plant")</li>
    <li>@Html.ActionLink(Resources.Teksten.Soorten, "Index", "Soort")</li>
    <li>@Html.ActionLink(Resources.Teksten.Leveranciers, "Index",
                           "Leverancier")</li>
</ul>

```

- In *Index.cshtml* breng je gelijkaardige wijzigingen aan :

```

<div class="row">
    <div class="col-md-4">
        <h2>@Html.ActionLink(Resources.Teksten.Planten, "Index", "Plant")</h2>
        <p>Overzicht van het plantenassortiment.</p>
    </div>
    <div class="col-md-4">
        <h2>@Html.ActionLink(Resources.Teksten.Soorten, "Index", "Soort")</h2>
        <p>Informatie over de diverse soorten planten.</p>
    </div>
    <div class="col-md-4">
        <h2>@Html.ActionLink(Resources.Teksten.Leveranciers, "Index",
                           "Leverancier")</h2>
        <p>De leveranciersgegevens.</p>
    </div>
</div>

```

Als je nu klikt op de knop 'Français' dan zou je homepage er als volgt moeten uitzien :

The screenshot shows the homepage of a website named 'TUINCENTRUM MVC'. At the top, there is a navigation bar with links: 'Tuincentrum', 'Home', 'Plantes' (which is highlighted with a red box), 'Genres', and 'Fournisseurs'. To the right of the navigation bar are two small flags: the Netherlands flag and the France flag. Below the navigation bar is the website's logo, which consists of a magnifying glass icon over a green leaf, followed by the text 'TUINCENTRUM' and 'MVC' in orange. The main content area has a light gray background and features a large heading 'Bienvenue sur le site Tuincentrum'. Below this, a sub-headline says 'De tuincentrum-website is een voorbeeld website uit de .NET MVC cursus.' There is a blue button labeled 'Meer over MVC »'. At the bottom of the main content area, there are three buttons with red boxes around them: 'Plantes' (with the sub-headline 'Overzicht van het plantenassortiment.'), 'Genres' (with the sub-headline 'Informatie over de diverse soorten planten.'), and 'Fournisseurs' (with the sub-headline 'De leveranciersgegevens.'). At the very bottom of the page, there is a small copyright notice: '© 2014 - .NET Ontwikkelaar C#'

16.4.4 Errormessages in verschillende talen

Afhankelijk van de gekozen taal willen we de validatiefoutbericht van de prijs van de planten (tussen 0 en 1000) in het Nederlands of Frans weergeven.

- Voeg in de twee Teksten-resourcefiles een entry toe.

De Nederlandse versie :

Name : *RangePrijs*
Value : *De verkoopprijs moet liggen tussen {1} en {2}.*

Voor de Franse resourcefiles :

Name : *RangePrijs*
Value : *Le prix de vente doit se situer entre {1} et {2}.*

- Wijzig in *PlantProperties.cs* het volgende :

```
public class PlantProperties {
    [Range(0, 1000, ErrorMessageResourceType=typeof(Resources.Teksten),
          ErrorMessageResourceName="RangePrijs")]
    public decimal VerkoopPrijs { get; set; }
}
```

Je kan dit nu uitproberen :

- Klik op de taalknop *Français* en kies het menu-item *Plantes*.
- Klik naast één van de planten op de Edit-link en wijzig de prijs naar bijvoorbeeld 2000.
- Verander van invulveld of klik op Save. De foutbericht verschijnt in het Frans.

16.4.5 Labels in verschillende talen

We kunnen ook desgewenst de labels aanpassen in meerdere talen. Bij wijze van voorbeeld gaan we het label van de verkoopprijs van een plant laten veranderen in enkele views.

Het opschrift van een label voor een veld kan je instellen met de Display-annotation. Maar vooraleer we dit kunnen doen moeten we nog een wijziging aanbrengen aan onze global resources files. Een global resources file heeft als Access Modifier altijd internal en dat is onvoldoende. Dit moet public zijn. Om dat te bekomen doe je het volgende :

- Selecteer in de Solution Explorer de twee resourcefiles : *Teksten.resx* en *Teksten.fr.resx*.
- Kies F4 om de properties te kunnen aanpassen.
- Zet de property *Build Action* op *Embedded Resource*. De property *Custom Tool* zet je op *PublicResXFileCodeGenerator*. Wanneer je nu één van beide resourcefiles zou openen dan zal je zien dat de Access Modifier op Public staat.
- Je voegt in de resourcefiles een entry met Name *LabelPrijs* toe. In de nederlandstalige resourcefile krijgt deze dan de waarde "Verkoopprijs", in de franstalige "Prix de vente".

In de partial class *PlantProperties* breng je het label aan voor het veld *Verkoopprijs* :

```
public class PlantProperties
{
    [Display(ResourceType = typeof(App_GlobalResources.Teksten),           (1)
             Name = "LabelPrijs")]
    [Range(0, 1000, ErrorMessageResourceType=typeof(App_GlobalResources.Teksten),
          ErrorMessageResourceName="RangePrijs")]
    public decimal Verkoopprijs { get; set; }
}
```

(1) Je verwijst deze keer naar een resource-entry met *App_GlobalResources.<naamVanDeEntry>*

In de verschillende views (*Edit.cshtml*, *Create.cshtml*, *Details.cshtml* en *Delete.cshtml*) wordt nu afhankelijk van de gekozen taal het correcte label getoond.

Naam	Rododendron
Kleur	Rood
Prix de vente	7,25
Naam	Mooiweer
Naam	Heester
Afbeelding	

- Probeer maar uit.

16.5 Samenvatting

- Wil je internationaal gaan met jouw website dan kan je de tekst die je ziet op jouw website ophalen uit resourcefiles. Per taal of beter per culture, maak je een aparte file met daarin de teksten in een specifieke taal.
- De gebruiker kiest de gewenste taal door de instellingen van zijn browser te wijzigen. Je kan de website ook voorzien van een knop om te veranderen van taal. In dit laatste geval moet je via code de gewenste culture instellen.
- Naast de vaste tekst in jouw pagina kan je ook foutberichten, labels, ... in andere talen laten weergeven.

17 ActionFilters

Soms is het wenselijk dat een stuk code bij alle of meerdere controllers moet worden uitgevoerd. We kunnen dit verwezenlijken door ActionFilters te gebruiken.

Bij deze methode maken we eerst een class die erft van `ActionFilterAttribute`. In die class schrijven we dan code in één van de onderstaande methods. Naast de naam van de method is het tijdstip vermeld wanneer de method wordt uitgevoerd :

- `OnActionExecuting()` : net voor de controller
- `OnActionExecuted()` : net na de controller
- `OnResultExecuting()` : net voor de view
- `OnResultExecuted()` : net na de view

Nu zijn er twee manieren om aan te geven voor welke controllers de code uit de filter moet worden uitgevoerd. Je kan in een controller met een annotation aangeven dat een bepaalde actionfilter moet worden uitgevoerd. Moet de actionfilter bij alle controllers worden uitgevoerd dan kan je dat in het bestand `FilterConfig.cs` uit de folder `App_Start` in de method `RegisterGlobalFilters()` aangeven.

17.1 ActionFilters voor alle controllers

Als voorbeeld houden we een statistiek bij van het aantal bezoeken voor elke pagina van onze webapplicatie. We starten met het aanmaken van een class `StatistiekActionFilter.cs`, afgeleid van `ActionFilterAttribute`.

- Maak een nieuwe class `StatistiekActionFilter` en bewaar deze in een nieuwe folder `Filters` :

```
using System.Web.Mvc;
...
public class StatistiekActionFilter : ActionFilterAttribute
{
    static Dictionary<string, int> statistiek = new Dictionary<string, int>(); (1)
    public override void OnActionExecuting(ActionExecutingContext filterContext)
    {
        string url = filterContext.HttpContext.Request.Url.ToString();
        lock (statistiek)
        {
            if (statistiek.ContainsKey(url))
                statistiek[url]++; (3)
            else
                statistiek[url] = 1; (4)
        }
    }
    public static Dictionary<string, int> Statistiek (2)
    {
        get
        {
            return statistiek;
        }
    }
}
```

Deze class erft van ActionFilterAttribute en bevat een static dictionary *statistiek* waarin we per url (een string) het aantal bezoeken (een int) bewaren (1). Deze statistiek kan via een read-only property opgevraagd worden (2). In een method *OnActionExecuting()* passen we de statistiek aan : we vragen de url op en verhogen de statistiek als de url reeds in de dictionary is opgenomen (3) of we zetten het aantal op 1 als dat niet het geval is (4).

We voegen deze actionfilter nu toe aan alle controllers door in *FilterConfig.cs* de filter toe te voegen.

- Wijzig de code in *FilterConfig.cs* als volgt :

```
public static void RegisterGlobalFilters(GlobalFilterCollection filters)
{
    filters.Add(new HandleErrorAttribute());
    filters.Add(new Filters.StatistiekActionFilter());
}
```

- Om de statistiek te kunnen bekijken voeg je een empty controller toe met de naam *StatistiekController*.
- Voeg aan de Index-method een bijhorende View toe :

```
@{
    ViewBag.Title = "Index";
}
<h2>Statistiek van deze website</h2>
<table>
    <thead>
        <tr>
            <th>URL</th>
            <th>Bezoeken</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var entry in
            MVC_Tuincentrum.Filters.StatistiekActionFilter.Statistiek)
        {
            <tr>
                <td>@entry.Key</td>
                <td>@entry.Value</td>
            </tr>
        }
    </tbody>
</table>
```

- Je kan dit nu uitproberen door naar enkele verschillende url's te surfen in jouw webapplicatie en vervolgens naar de statistiekkpagina .../*Statistiek*.

Een mogelijk resultaat :

URL	Bezoeken
http://localhost:52502/	2
http://localhost:52502/Leverancier	1
http://localhost:52502/Soort	1
http://localhost:52502/Soort/Details/21	
http://localhost:52502/Soort/Edit/2	1
http://localhost:52502/Plant	4
http://localhost:52502/Plant/Details/31	
http://localhost:52502/Plant/Details/61	
http://localhost:52502/Plant/Details/22	
http://localhost:52502/Statistiek	1

© 2014 - .NET Ontwikkelaar C#

17.2 ActionFilters voor één controller

Wil je een ActionFilter enkel bij één bepaalde controller laten uitvoeren dan gebruik je een annotation.

- Zet in *FilterConfig.cs* de zopas toegevoegde ActionFilter in commentaar.
- Open *PlantController.cs* en voeg bovenaan de annotation **[StatistiekActionFilter]** toe :

```
using MVC_Tuincentrum.Filters;
...
[StatistiekActionFilter]
public class PlantController : Controller
{
    ...
}
```
- Start opnieuw de applicatie en bezoek enkele pagina's. Bekijk daarna de statistiek en stel vast dat enkel pagina's met betrekking tot de planten vermeld zijn.

17.3 Filter overrides

Sinds MVC5 kan je een global filter uitschakelen voor een bepaalde controller door in die controller bovenaan een annotation **[OverrideActionFilters]** toe te voegen. Wil je bijvoorbeeld niet dat de action methods uit de LeverancierController opgenomen worden in de statistiek dan voeg je het onderstaande toe in de LeverancierController :

```
[OverrideActionFilters]
public class LeverancierController : Controller
{
    ...
}
```

We proberen dit even uit.

- Open *FilterConfig.cs* en activeer de global filter opnieuw door de regel code terug uit commentaar te halen.
- Voeg in de *LeverancierController* bovenaan de annotation `[OverrideActionFilters]` toe.
- Start de applicatie en bekijk enkele pagina's waaronder zeker enkele uit de *LeveranciersController*.
- Bekijk de statistiek en stel vast dat de actions uit de leverancierscontroller niet voorkomen in de statistiek.

Op deze manier worden filters voor alle actions uit één controller uitgeschakeld. Maar je kan ook filters uitschakelen voor een specifieke action. Dat doe je door de annotation `[OverrideActionFilters]` net voor de action method te plaatsen.

Een voorbeeld :

- Haal in *LeverancierController.cs* de annotation `[OverrideActionFilters]` bovenaan weg en verplaats hem net voor de action *Details()*.

```
// GET: Leverancier/Details/5
[OverrideActionFilters]
public ActionResult Details(int? id)
...
```

- Start de applicatie opnieuw en bekijk enkele pagina's waaronder zeker de action *Details* uit de *LeveranciersController* en nog enkele andere uit deze controller.
- Bekijk de statistiek en stel vast dat de actions *Details* uit de leverancierscontroller niet voorkomt in de statistiek.

17.4 Samenvatting

- Bevat jouw website code die vlak voor of na een controller moet uitgevoerd worden, dan kan je dit in een filter opnemen en deze laten uitvoeren.
- Code die bij elke controller moet uitgevoerd worden neem je op in *RegisterGlobalFilters* in *FilterConfig.cs*. Moet code uitgevoerd worden bij alle actions van één controller, dan geef je dat aan met een annotation.
- Je kan een global of controller filter overriden (uitschakelen) via de annotation `[OverrideActionFilters]`.

18 Custom Routing

In paragraaf 2.2.3 Routing hadden we het al even over de RouteTable en de default route die je in *RouteConfig.cs* terugvindt. In dit hoofdstuk gaan we extra routes of 'custom routes' toevoegen aan dit bestand.

18.1 Routes gebaseerd op een URI template

De defaultroute die standaard in *RouteConfig.cs* zit is van de vorm
`<controller>/<action>/<parameters>`.

Wanneer we alle planten wensen te zien dan kunnen we surfen naar .../*Plant/Index* of korter .../*Plant* omdat *Index* de default action is. De URL's beschrijven al vrij goed wat we precies zullen te zien krijgen maar we zitten wel vast aan een stuk van de benaming van de controllers en actions. De gebruiker heeft ook niet altijd een boodschap aan de interne structuur van ons programma en wil misschien liever een url in de vorm van .../*Plantenlijst* of .../*AllePlanten*.

We kunnen heel eenvoudig een route toevoegen aan de routetabel met de instructie :

```
routes.MapRoute("NaamVanDeRoute", "URL",
    new { controller = "NaamVanDeController", action = "NaamVanDeAction" });
```

Voorbeeld :

```
routes.MapRoute(
    "Alleplanten",
    "Plantenlijst",
    new { controller = "Plant", action = "Index" }
);
```

- Voeg bovenstaande regel code toe in *RouteConfig.cs*, net boven de default route en surf naar .../*Plantenlijst*. Je krijgt dan hetzelfde te zien als wanneer je surft naar .../*Plant* of .../*Plant/Index*.

We gaan nu ook een route toevoegen waarmee we de gegevens van één bepaalde plant kunnen zien. Voorheen moesten we surfen naar .../*Plant/Details/3* om de plant met id = 3 te kunnen zien.

- Voeg nu onderstaande code toe, opnieuw vóór de defaultroute :

```
routes.MapRoute(
    "PlantByNr",
    "Plant/{id}",
    new { controller = "Plant", action = "Details" }
);
```

We voegen hiermee een nieuwe route toe met de naam *PlantByNr*. In de 3^e parameter geven we aan dat de action *Details* uit de *PlantController* moet worden uitgevoerd. Deze verwacht een parameter met naam *id*. Deze zie je in de URL *plant/{id}*. De naam van de parameter, *id*, moet dezelfde zijn als deze in de actioncode in de controller !!

- Surf nu naar .../*Plant/3* en je verkrijgt hetzelfde resultaat als .../*Plant/Details/3*.

Opgepast ! De volgorde waarin je routes toevoegt aan RouteConfig.cs is belangrijk want jouw applicatie zal steeds de eerste route gebruiken die matcht met de ingegeven URL. Je eindigt dus best met de defaultroute.

Op een gelijkaardige manier zouden we routes kunnen toevoegen als .../leverancier/{id} of .../soort/{id}. Maar wat als we bijvoorbeeld alle waterplanten zouden willen uitlijsten en dit in de URL willen schrijven als :

.../soort/water/planten

Om dit te kunnen doen moeten we eerst aan de slag in de PlantController. Daar maken we een nieuwe action aan met de naam FindPlantenBySoortNaam..

- Voeg onderstaande action toe in de PlantController :

```
public ActionResult FindPlantenBySoortNaam(string soortnaam)
{
    List<Plant> plantenLijst = new List<Plant>();
    plantenLijst = (from plant in db.Planten.Include("Soort")
                    where plant.Soort.Naam.StartsWith(soortnaam)
                    select plant).ToList();
    return View(plantenLijst);
}
```

Deze action levert ons een plantenlijst op waarbij de planten allen behoren tot een soort waarvan de naam begint met de opgegeven stringparameter.

- Voeg nu ook onderstaande bijkomende view toe :

```
@model IEnumerable<MVC_Tuincentrum.Models.Plant>
 @{
    ViewBag.Title = "FindPlantenBySoortNaam";
}
<h2>FindPlantenBySoortNaam</h2>
<table>
    <thead>
        <tr>
            <th>Naam</th>
            <th>Soort</th>
            <th>Leverancier</th>
            <th>Kleur</th>
            <th>Prijs</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model) {
            <tr>
                <td>@Html.DisplayFor(modelItem => item.Naam)</td>
                <td>@Html.DisplayFor(modelItem => item.Soort.Naam)</td>
                <td>@Html.DisplayFor(modelItem => item.Leverancier.Naam)</td>
                <td>@Html.DisplayFor(modelItem => item.Kleur)</td>
                <td>@Html.DisplayFor(modelItem => item.VerkoopPrijs)</td>
            </tr>
        }
    </tbody>
</table>
```

Indien je nu al zou surfen naar .../Plant/FindPlantenBySoortNaam?soortnaam=water dan krijg je een fout. MVC bestudeert deze URL en zal de route met naam *PlantByNr* gebruiken omdat het een patroon Plant/{id} opmerkt en dat is niet de bedoeling. Daarom voegen we een gepaste route toe.

- Om via een URL .../soort/water/planten deze waterplanten uit te lijsten voeg je volgende route toe in *RouteConfig.cs* (opnieuw net voor de defaultroute) :

```
routes.MapRoute(
    "PlantenVanEenSoort",
    "soort/{soortnaam}/planten",
    new { controller = "Plant", action = "FindPlantenBySoortNaam" }
);
```

We geven de route een naam : *PlantenVanEenSoort*. De URL begint met 'soort', dan de naam van de soort en tenslotte 'planten'. Dit zorgt ervoor dat de action *FindPlantenBySoortNaam* uit de *PlantController* geactiveerd wordt. De parameter voor de action wordt uit de url gehaald.

- Browse nu naar .../soort/water/planten en je zou onderstaand resultaat moeten krijgen. Probeer ook eens .../soort/boom/planten.

FindPlantenBySoortNaam

Naam	SoortLeverancier	Kleur	Prijs
Kalmoes	Water Struik	Onbekend	1,67
Wolgras	Water Dezaaier	Wit	1,31
Waterlelie	Water De groene kas	Wit	4,46
Blaasjeskruid	Water Bloem	Geel	0,93
Cypergras	Water Baumgarten	Onbekend	1,86
Dotterbloem	Water Struik	Geel	1,67
Kikkerbeet	Water Spitman	Wit	0,47
Lisdodde	Water Struik	Geel	1,67
Waterhyacint	Water Baumgarten	Blauw	1,86

Als laatste voorbeeld gaan we een beetje in dezelfde stijl alle planten van een bepaalde leverancier uitlijsten. We willen een URL in de volgende stijl : .../leverancier/3/planten

We beginnen opnieuw met een nieuwe action in de *PlantController*.

- Voeg onderstaande action toe in de *PlantController* :

```
public ActionResult FindPlantenByLeverancier(int? levnr)
{
    List<Plant> plantenLijst = new List<Plant>();
    plantenLijst = (from plant in db.Planten.Include("Leverancier")
                    where plant.Leverancier.LevNr == levnr
                    select plant).ToList();
    return View(plantenLijst);
}
```

Deze keer geven we het nummer van de leverancier mee als parameter. Een LINQ-statement zorgt voor het ophalen van de juiste planten. De bijhorende view is nagenoeg dezelfde als daarnet.

- Voeg deze view toe :

```
@model IEnumerable<MVC_Tuincentrum.Models.Plant>
@{
    ViewBag.Title = "Planten van een leverancier";
}
<h2>Planten van een leverancier</h2>
<table>
    <thead>
        <tr>
            <th>Naam</th>
            <th>Soort</th>
            <th>Leverancier</th>
            <th>Kleur</th>
            <th>Prijs</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model) {
            <tr>
                <td>@Html.DisplayFor(modelItem => item.Naam)</td>
                <td>@Html.DisplayFor(modelItem => item.Soort.Naam)</td>
                <td>@Html.DisplayFor(modelItem => item.Leverancier.Naam)</td>
                <td>@Html.DisplayFor(modelItem => item.Kleur)</td>
                <td>@Html.DisplayFor(modelItem => item.VerkoopPrijs)</td>
            </tr>
        }
    </tbody>
</table>
```

- In de routetabel voegen we vóór de defaultroute een nieuwe route toe :

```
routes.MapRoute(
    "PlantenVanEenLeverancier",
    "leverancier/{levnr}/planten",
    new { controller = "Plant", action = "FindPlantenByLeverancier" }
);
```

- Je kan nu volgende URL uitproberen : .../leverancier/3/planten.

We hebben nu een aantal extra routes gedefinieerd in *RouteConfig.cs*. Dit gebeurde telkens met de instructie *routes.MapRoute()*. De eerste parameter van *MapRoute()* is de naam van de route. Je kan deze naam in een view gebruiken in een hyperlink met de functie *@Url.RouteUrl()*.

In het hoofdstuk Hyperlinks hebben we reeds hyperlinks gebruikt die verwezen naar een Url of naar een action. Een derde soort verwijzing is dus een verwijzing naar een route. In onderstaande voorbeelden verwijst je naar de eerste route die we maakten, namelijk “Alleplanten” en naar de route die we daarnet maakten, “PlantenVanEenLeverancier”, waarbij we de plantenlijst voor leverancier met id 3 tonen :

```
<a href="@Url.RouteUrl("AllePlanten")">Alle planten</a>
<a href="@Url.RouteUrl("PlantenVanEenLeverancier", new { levnr=3})">plantenlijst</a>
```

18.2 Routes gebaseerd op een query string

De routes die we tot nu toe hebben toegevoegd waren allen gebaseerd op een URL. Je kan echter ook routes baseren op de URL én de querystring. Dit betekent dat we bijvoorbeeld MVC voor een

bepaalde URL een route A laten volgen wanneer er één querystring is met de naam *levnr* of route B wanneer er twee querystrings zijn met naam *minWedde* en *maxWedde*.

Een voorbeeld : we gaan een route toevoegen die ons leidt tot een action die alle planten uitlijst tussen een minimum- en een maximumprijs én een route die leidt tot alle planten van een bepaalde kleur. Beide routes zullen dezelfde url gebruiken maar een verschillend aantal parameters.

- In de PlantController voeg je twee actions toe :

```
public ActionResult FindPlantenBetweenPrijs(decimal minPrijs,
                                             decimal maxPrijs)
{
    List<Plant> plantenLijst = new List<Plant>();
    plantenLijst = (from plant in db.Planten
                    where plant.VerkoopPrijs >= minPrijs &&
                          plant.VerkoopPrijs <= maxPrijs
                    select plant).ToList();
    ViewBag.minprijs = minPrijs;
    ViewBag.maxprijs = maxPrijs;
    return View(plantenLijst);
}

public ActionResult FindPlantenVanEenKleur(string kleur)
{
    List<Plant> plantenLijst = new List<Plant>();
    plantenLijst = (from plant in db.Planten
                    where plant.Kleur == kleur
                    select plant).ToList();
    ViewBag.kleur = kleur;
    return View(plantenLijst);
}
```

We moeten nu een class aanmaken die de interface *IRouteConstraint* implementeert. Deze class gebruiken we dan zodadelijk in de *routes.MapRoute()*-instructie in *RouteConfig.cs*.

- Voeg in de root van de webapplicatie een class *QueryStringConstraint.cs* toe :

```
using System.Web.Routing;
...
public class QueryStringConstraint : IRouteConstraint
{
    private string[] verwachteParameters;

    public QueryStringConstraint(string[] verwachteParameters)
    {
        this.verwachteParameters = verwachteParameters;
    }

    public bool Match(HttpContextBase httpContext, Route route,
                      string parameterName, RouteValueDictionary values,
                      RouteDirection routeDirection)
    {
        foreach (string verwachteParameter in verwachteParameters)
        {
            if (!httpContext.Request.QueryString.AllKeys.Contains(
                verwachteParameter, StringComparer.OrdinalIgnoreCase))
                return false;
        }
        return true;
    }
}
```

- Voeg in *RouteConfig.cs* vóór de defaultroute volgende twee routes toe :

```

routes.MapRoute(
    "FindPlantenByPrijsBetween",
    "planten",
    new { controller = "Plant", action = "FindPlantenBetweenPrijzen" }, (1)
    new {
        QueryConstraint = new QueryStringConstraint(
            new string[] { "minprijs", "maxprijs" }) (2)
    }
); (3)

routes.MapRoute(
    "FindPlantenByKleur",
    "planten",
    new { controller = "Plant", action = "FindPlantenVanEenKleur" }, (4)
    new {
        QueryConstraint = new QueryStringConstraint(
            new string[] { "kleur" }) (5)
    }
); (6)

```

We geven telkens eerst de nieuwe routes een naam (1) mee en een URL (2). Bij beide is de URL *planten*. Bij de eerste wordt uit de *PlantController* de action *FindPlantenBetweenPrijzen* uitgevoerd (3), bij de tweede de action *FindPlantenVanEenKleur* (4). Als extra parameter geven we hier een *QueryStringConstraint* mee. Dit is een instance van onze nieuwe klasse *QueryStringConstraint()*. Bij de eerste route geven we de string 'minprijs' en 'maxprijs' mee (5), bij de tweede de string 'kleur' (6).

Dit betekent dat de eerste route enkel zal gekozen worden wanneer de URL .../planten is én er in de querystring twee parameters zijn met naam *minprijs* en *maxprijs*. De tweede route wordt gekozen voor een URL .../planten én een querystring met een parameter *kleur*. Het is de method *Match()* uit de class *QueryStringConstraint* die controleert of de juiste parameters aanwezig zijn in de querystring.

- Voeg aan de method *FindPlantenBetweenPrijzen* volgende view toe :

```

@model List<MVC_Tuincentrum.Models.Plant>
 @{
    ViewBag.Title = "FindPlantenBetweenPrijzen";
}
<h2>Planten met prijs tussen € @ViewBag.minprijs en € @ViewBag.maxprijs</h2>
<table>
    <thead>
        <tr>
            <th>Naam</th>
            <th>Soort</th>
            <th>Leverancier</th>
            <th>Kleur</th>
            <th>Prijs</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model) {
            <tr>
                <td>@Html.DisplayFor(modelItem => item.Naam)</td>
                <td>@Html.DisplayFor(modelItem => item.Soort.Naam)</td>
                <td>@Html.DisplayFor(modelItem => item.Leverancier.Naam)</td>
                <td>@Html.DisplayFor(modelItem => item.Kleur)</td>
            </tr>
        }
    </tbody>
</table>

```

```
<td>@Html.DisplayFor(modelItem => item.VerkoopPrijs)</td>
</tr>
}
</tbody>
</table>
```

- En voeg aan de method *FindPlantenVanEenKleur* onderstaande view toe :

```
@model List<MVC_Tuincentrum.Models.Plant>
 @{
    ViewBag.Title = "FindPlantenVanEenKleur";
}
<h2>Planten met als kleur : @ViewBag.kleur</h2>
<table>
    <thead>
        <tr>
            <th>Naam</th>
            <th>Soort</th>
            <th>Leverancier</th>
            <th>Kleur</th>
            <th>Prijs</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model) {
            <tr>
                <td>@Html.DisplayFor(modelItem => item.Naam)</td>
                <td>@Html.DisplayFor(modelItem => item.Soort.Naam)</td>
                <td>@Html.DisplayFor(modelItem => item.Leverancier.Naam)</td>
                <td>@Html.DisplayFor(modelItem => item.Kleur)</td>
                <td>@Html.DisplayFor(modelItem => item.VerkoopPrijs)</td>
            </tr>
        }
    </tbody>
</table>
```

Het resultaat met twee querystringparameters :

The screenshot shows a browser window titled 'FindPlantenBetween!'. The address bar contains 'localhost:52502/planten?minprijs=2&maxprijs=5.5'. The page header includes 'Tuincentrum' and navigation links for 'Home', 'Planten', 'Soorten', and 'Leveranciers'. Below the header is the 'TUINCENTRUM MVC' logo. The main content area displays a heading 'Planten met prijs tussen € 2 en € 5,5' followed by a table of plant details:

Naam	Soort	Leverancier	Kleur	Prijs
Wijnstok	Klim	Struik	Onbekend	3,72
Dwergcypres	Boom	Bloem	Onbekend	5,02
Forsythia	Heester	Struik	Geel	2,05
Kornoelje	Heester	Struik	Geel	2,05
Aziënboom	Boom	Snitman	Rood	3,53

Het resultaat met één querystringparameter *kleur* :

Naam	Soort	Leverancier	Kleur	Prijs
Vuurdoorn	Heester	Baumgarten	Wit	1,86
Paardekastanje	Boom	De groene kas	Wit	6,51
Populier	Boom	Struik	Wit	1,67
Boterbloem	Vast	Mooiweer	Wit	1,12
Bereklauw	Vast	Dezaaier	Wit	1,31
Draon	Kruid	De Plukker	Wit	0,74

Controleer altijd goed de routetabel. MVC kiest immers altijd de eerste route die voldoet !

18.3 Attribute routing

Sinds MVC 5 is er een nieuwe manier om controller actions te koppelen aan URI's, namelijk via Attribute routing. Hierbij geef je aan de hand van een annotation meteen in de controller zelf aan welke route er bij een bepaalde action past.

18.3.1 Attribute routing inschakelen en toepassen

Om attribute routing in te schakelen roep je in de method *RegisterRoutes* in *RouteConfig.cs* de method *routes.MapMvcAttributeRoutes()* op :

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
    routes.MapMvcAttributeRoutes();
    routes.MapRoute(...
```

- Voeg in *RouteConfig.cs* de bovenstaande vetgedrukte regel code toe.

Attribute routing kan perfect gecombineerd worden met de zogenaamde convention based routing uit de vorige paragrafen. Het heeft echter het voordeel dat je in de controller meteen ziet welke route er past bij een action method.

Bij wijze van voorbeeld passen we attribute routing toe in een nieuwe action method in de LeverancierController.

- Voeg onderstaande action method toe in *LeverancierController.cs* :

```
[Route("Leveranciers/{postnr}")]
public ActionResult FindLeveranciersMetPostNr(string postnr)
{
    List<Leverancier> leveranciersLijst = new List<Leverancier>();
    leveranciersLijst = (from leverancier in db.Leveranciers
                          where leverancier.PostNr == postnr
                          select leverancier).ToList();
    ViewBag.postnr = postnr;
    return View(leveranciersLijst);
}
```

Deze method levert ons een lijst met leveranciers met een welbepaald postnummer. De route die bij deze action hoort geven we aan met de annotation `[Route("Leveranciers/{postnr}")]`.

- Voeg nu een View toe aan deze action method :

```
@model List<MVC_Tuincentrum.Models.Leverancier>
 @{
    ViewBag.Title = "Leveranciers met postnr";
}
<h2>Leveranciers met postnr @ViewBag.postnr</h2>
<table>
    <thead>
        <tr>
            <th>Naam</th>
            <th>Adres</th>
            <th>Woonplaats</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model)
        {
            <tr>
                <td>@Html.DisplayFor(modelItem => item.Naam)</td>
                <td>@Html.DisplayFor(modelItem => item.Adres)</td>
                <td>@Html.DisplayFor(modelItem => item.Woonplaats)</td>
            </tr>
        }
    </tbody>
</table>
```

- Browse nu naar .../Leveranciers/8500 en je krijgt de leveranciers uit Kortrijk.

18.3.2 Default en optionele parameters

Je kan in de annotation een parameter een default waarde geven. In ons voorbeeld is het misschien minder nuttig maar met de annotation `[Route("Leveranciers/{postnr=8500}")]` krijg je telkens de leveranciers uit Kortrijk te zien, tenzij je zelf een postnr opgeeft.

- Wijzig de annotation boven de action method *FindLeveranciersMetPostNr* zodat het een defaultwaarde 8500 krijgt.
- Browse nu naar .../Leveranciers. Je krijgt de leveranciers uit Kortrijk te zien.
- Browse naar .../Leveranciers/8560 en je krijgt de leveranciers uit Wevelgem.

Wil je liever geen defaultwaarde maar toch de mogelijk bewaren om een parameter geen waarde

mee te geven dan kan je een parameter ook optioneel maken. Je voegt na de naam van de parameter een ? toe.

- Wijzig de annotation boven de method FindLeveranciersMetPostNr als volgt :
- ```
[Route("Leveranciers/{postnr?}")]
```
- De parameter kan nu leeg zijn en dus passen we de code in de action method aan : is postnr leeg dan laten we alle leveranciers zien, is de parameter niet leeg dan passen we deze toe in de linq-query :

```
public ActionResult FindLeveranciersMetPostNr(string postnr)
{
 if (postnr == null)
 return View("Index", db.Leveranciers.ToList());
 else
 {
 List<Leverancier> leveranciersLijst = new List<Leverancier>();
 leveranciersLijst = (from leverancier in db.Leveranciers
 where leverancier.PostNr == postnr
 select leverancier).ToList();
 ViewBag.postnr = postnr;
 return View(leveranciersLijst);
 }
}
```

In bovenstaand voorbeeld hadden we i.p.v. twee verschillende views ook twee verschillende linq-query's en slechts één view kunnen gebruiken.

### 18.3.3 Route prefixes en defaultroutes

Als je eens gaat kijken in bijvoorbeeld de SoortController dan zie je dat de URL's die bij de action methods horen allemaal beginnen met /Soort. Als je de routing instelt via annotations zal je dit dus telkens opnieuw moeten tikken aan het begin van de annotation.

Met een route prefix kan je helemaal bovenaan de controller class aangeven dat alle action methods met een bepaald stuk URL moeten beginnen. We passen dit toe op de HomeController omdat deze het minst aantal action methods heeft.

- Open *HomeController.cs* en wijzig deze class als volgt (de wijzigingen zijn in het vet aangegeven) :

```
[RoutePrefix("Thuis")]
public class HomeController : Controller
{
 [Route]
 public ActionResult Index()
 {
 return View();
 }

 [Route("Over")]
 public ActionResult About()
 {
 ViewBag.Message = "Your application description page.";
 return View();
 }
}
```

```
[Route("Contacteer")]
public ActionResult Contact()
{
 ViewBag.Message = "Your contact page.";
 return View();
}
```

Met `[RoutePrefix("Thuis")]` geven we aan dat alle action methods een route krijgen die begint met Thuis/. Bij de Index-method komt daar niks extra bij. Dat geven we aan met de annotation `[Route]`. De About-action heeft dan de route Thuis/Over, de Contact-method krijg je met Thuis>Contacteer. Dit is mogelijk via een annotation `[Route]` mét een parameter.

- Probeer maar eens uit.

Het gevolg hiervan is wel dat oproepen naar Home/, Home/About en Home/Contact niet meer werken.

Je kan evenwel voor een bepaalde action method de route prefix overschrijven met een tilde (~).

- Vervang de annotation voor de action method Contact door het onderstaande :

```
[Route("~/Home/Contact")]
```

Deze ene action method is nu terug toegankelijk via /Home/Contact en niet meer via /Thuis/Contact.

Je merkt dat het aanbrengen van de annotations al snel heel wat werk met zich meebrengt en zeg je allicht terecht dat het werken met de defaultrouting in RouteConfig.cs toch minder werk vergt. Dat klopt, maar ook met annotations kan je aan default routing doen.

- Haal in `HomeController.cs` bij de action methods (Index, About en Contact) de annotations weg. De route prefix bovenaan de class mag blijven staan.
- Onder de prefix noteer je een default route :

```
[RoutePrefix("Thuis")]
[Route("{action=index}")]
public class HomeController : Controller
{
 public ActionResult Index() { ... }

 public ActionResult About() { ... }

 public ActionResult Contact() { ... }
}
```

Met `[Route("{action}")]` geven we aan dat na de prefix gewoon de naam van de action komt. We browsen dus naar /Thuis/Index, /Thuis/About of /Thuis/Contact. Met `=index` geef je aan dat als er naar /Thuis wordt gebrowset, de index-action wordt gekozen. Deze is dus de defaultaction.

De action methods zelf hebben geen route-annotations meer.

### 18.3.4 Route Constraints

Met route constraints kan je opleggen dat parameters van een bepaald type moeten zijn. Met onderstaande route zeg je dat de parameter id van het type int moet zijn :

```
[Route("plantinfo/{id:int}")]
```

We passen dit even toe in de PlantController.

- Voeg onderstaande action methods toe in de PlantController :

```
[Route("plantinfo/{id:int}")]
public ActionResult FindPlantById(int id)
{
 var plant = db.Planten.Find(id);
 if (plant != null)
 return View("Details", plant);
 else
 {
 var planten =
 db.Planten.Include(p => p.Leverancier).Include(p => p.Soort);
 return View("Index", planten.ToList());
 }
}

[Route("plantinfo/{naam}")]
public ActionResult FindPlantByName(string naam)
{
 var plant = (from p in db.Planten
 where p.Naam == naam
 select p).FirstOrDefault();
 if (plant != null)
 return View("Details", plant);
 else
 {
 var planten =
 db.Planten.Include(p => p.Leverancier).Include(p => p.Soort);
 return View("Index", planten.ToList());
 }
}
```

In de action method FindPlantById moet de id van het type int zijn. Dat geven we aan met `{id:int}`. Wanneer de parameter geen int is dan wordt de action method FindPlantByName gebruikt.

Om niet voor elk van beide action methods een nieuwe view te moeten maken ‘recupereren’ we de Details view en geven we de gevonden plant mee.

Wordt de gewenste plant niet gevonden dan geven we alle planten weer met de Index-view.

In het bovenstaande voorbeeld hebben we opgelegd dat de parameter een int moest zijn. In onderstaande tabel zie je welke constraints er nog meer mogelijk zijn.

| Constraint | Beschrijving                     | Voorbeeld       |
|------------|----------------------------------|-----------------|
| long       | Een 64-bit integer waarde        | {id:long}       |
| float      | Een 32-bit floating point waarde | {value:float}   |
| double     | Een 64-bit floating point waarde | {value:double}  |
| decimal    | Een decimaal getal               | {value:decimal} |

|           |                                                               |                                                                                                                      |
|-----------|---------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------|
| min       | Een getal groter dan het opgegeven minimum                    | {waarde:min(1)}<br>(getal groter dan 0)                                                                              |
| max       | Een getal kleiner dan het opgegeven maximum                   | {waarde:max(10)}<br>(getal kleiner dan 11)                                                                           |
| range     | Een getal binnen een range                                    | {waarde:range(1,10)}<br>(getal tussen 1 en 10)                                                                       |
| bool      | Een boolean waarde                                            | {value:bool}                                                                                                         |
| datetime  | Een DateTime waarde                                           | {wanneer:datetime}                                                                                                   |
| alpha     | Bevat enkel kleine en/of hoofdletters                         | {tekst:alpha}                                                                                                        |
| length    | Een string met bepaalde lengte of een lengte binnen een range | {tekst:length(6)}<br>(tekst is exact 6 tekens lang)<br>{tekst:length(1,20)}<br>(tekst is tussen 1 en 20 tekens lang) |
| minlength | Een string met een minimumlengte                              | {tekst:minlength(6)}<br>(tekst van min.6 tekens lang)                                                                |
| maxlength | Een string met een maximumlengte                              | {tekst:maxlength(10)}<br>(tekst van max.10 tekens lang)                                                              |
| regex     | Waarde voldoet aan een regular expression                     | {reknr:regex("^\d{3}-\d{7}-\d{2}")}<br>(reknr heeft vorm van rekeningnr.)                                            |

Constraints kan je ook combineren door ze van elkaar met een dubbelpunt te scheiden.

In onderstaand voorbeeld is de id van de plant maximaal 10 tekens lang en bestaat deze enkel uit tekens A-Z en/of a-z :

```
[Route("plantinfo/{id:alpha:maxlength(10)}")]
```

### 18.3.5 Custom route constraints

Tenslotte kan je ook custom route constraints opleggen. Als voorbeeld zullen we in de route annotation aangeven of de prijzen in de plantenlijst inclusief of exclusief BTW weergegeven worden. De mogelijke waarden voor de parameter zijn dus beperkt tot slechts twee strings : “inclusief” en “exclusief”.

- Voeg in *PlantController.cs* een nieuwe actionmethod toe :

```
[Route("plantenprijzen/{btw:values(inclusief|exclusief)}")]
public ActionResult PrijsLijst(string btw)
{
 ViewBag.Btw = btw;
 return View(db.Planten.ToList());
}
```

Bovenstaande action zullen we kunnen oproepen door te browsen naar .../plantenprijzen/inclusief ofwel naar .../plantenprijzen/exclusief. “inclusief” en “exclusief” zijn de mogelijke waarden voor de parameter btw. De waarde wordt via de ViewBag doorgegeven naar de view.

- Om dit te realiseren maak je eerst in de root van het project een nieuwe class

*ValuesConstraint.cs* :

```
using System.Web.Routing;
...
public class ValuesConstraint : IrouteConstraint (4)
{
 private readonly string[] validOptions;
 public ValuesConstraint(string options)
 {
 validOptions = options.Split('|');
 }

 public bool Match(HttpContextBase httpContext, Route route,
 string parameterName, RouteValueDictionary values,
 RouteDirection routeDirection) (5)
 {
 object value;
 if (values.TryGetValue(parameterName, out value) && value != null)
 {
 return validOptions.Contains(value.ToString(),
 StringComparer.OrdinalIgnoreCase);
 }
 return false;
 }
}
```

Bovenstaande class gebruiken we in *RouteConfig.cs*.

- Voeg in *RouteConfig.cs* onderstaande vetgedrukte code toe :

```
using System.Web.Mvc.Routing;
...
public static void RegisterRoutes(RouteCollection routes)
{
 routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

 var constraintsResolver = new DefaultInlineConstraintResolver(); (1)
 constraintsResolver.ConstraintMap.Add("values", typeof(ValuesConstraint)); (2)
 routes.MapMvcAttributeRoutes(constraintsResolver); (3)
 //routes.MapMvcAttributeRoutes();

 routes.MapRoute(
 ...
)
}
```

Je begint dus met een nieuwe *DefaultInlineConstraintResolver* aan te maken (1). Aan deze class voeg je beperkingen of constraints toe in een ConstraintMap. Zo een constraint heeft een naam en een type. Bij ons is de naam “values” en het type *ValuesConstraint* (2). Bij het activeren van de attributeroutes geven we deze keer de constraintsresolver mee (3).

De class *ValuesConstraint* implementeert de interface *IRouteConstraint* (4). Deze hebben we ook eerder al in de class *QueryStringConstraint* geïmplementeerd. De class *ValuesConstraint* bevat dus ook een method *Match* die ons een boolean oplevert (5). Is de boolean false dan moet de route niet gevuld worden, is het true dan wel. In ons geval checkt de method *Match* of de waarde die de gebruiker ingetikt heeft in de URL matcht met één van de mogelijke values.

- Voeg nu onderstaande view toe bij de action method *PrijsLijst* :

```
@model IEnumerable<MVC_Tuincentrum.Models.Plant>
 @{
 ViewBag.Title = "PrijsLijst";
}
<h2>PrijsLijst</h2>
<table class="table">
 <thead>
 <tr>
 <th>@Html.DisplayNameFor(model => model.Naam)</th>
 <th>@Html.DisplayNameFor(model => model.VerkoopPrijs) (BTW @ViewBag.Btw)</th>
 </tr>
 </thead>
 <tbody>
 @foreach (var item in Model) {
 <tr>
 <td>@Html.DisplayFor(modelItem => item.Naam)</td>
 <td>
 @if (ViewBag.Btw == "inclusief") {
 decimal prijs = item.VerkoopPrijs * 1.21m;
 @Html.DisplayFor(modelItem => prijs)
 }
 else {
 @Html.DisplayFor(modelItem => item.VerkoopPrijs)
 }
 </td>
 </tr>
 }
 </tbody>
</table>
```

We geven van alle planten enkel de naam en de verkoopprijs weer. In de viewbag bevindt zich de waarde van de btw-parameter. Als deze waarde gelijk is aan “inclusief” dan berekenen we de kostprijs met btw. Is de waarde “exclusief” dan geven we gewoon de waarde weer uit de database.

Ook in de header van de tabel geven we aan of de prijzen inclusief of exclusief btw zijn.

- Test uit door te browsen naar .../plantenprijzen/inclusief of naar .../plantenprijzen/exclusief.

In het begin van dit hoofdstuk hebben we een pak routes toegevoegd in *RouteConfig.cs*. Elk van deze routes kreeg een naam zodat we via de functie `@Url.RouteUrl()` een hyperlink konden maken die gebruik maakt van de routenaam. Ook bij attribute routing kunnen we routes een naam geven.

- Bij wijze van voorbeeld geven we in de *PlantController* de route van de laatst toegevoegde action method, *PrijsLijst*, een routenaam *btwinex*.

```
[Route("plantenprijzen/{btw:values(inclusief|exclusief)}", Name="btwinex")]
public ActionResult PrijsLijst(string btw)
{
 ViewBag.Btw = btw;
 return View(db.Planten.ToList());
}
```

- Open *Index.cshtml* uit de folder *Views/Plant* en voeg helemaal onderaan onderstaande hyperlink toe :

```
Prijslijst inclusief BTW
```

- Browse naar .../Plantenlijst, scroll helemaal naar beneden tot onder de lijst en probeer de hyperlink uit.

## 18.4 Samenvatting

- Een default route is van de vorm `<controller>/<action>/<parameters>`. We kunnen hiervan afwijken door custom routes toe te voegen in *RouteConfig.cs*.
- Routes kunnen gebaseerd zijn op een URI template, eventueel met één of meerdere parameters
- Routes kunnen ook gebaseerd zijn op de querystring. Je gebruikt dan een `QueryStringConstraint` om uit te maken welke route er gevuld moet worden.
- Nieuw sinds MVC5 is Attribute Routing. Je bepaalt welke route er bij een action method hoort door in de controller, vlak voor de action method code een annotation `[Route]` te plaatsen.
- Je kan in de annotation ook parameters opgeven en ze eventueel een defaultwaarde geven.
- Via de annotation `[RoutePrefix]` kan je een controller een afwijkende prefix geven. Ben je niet tevreden met de naam van een action, dan kan je deze met de `[Route]` annotation via een andere naam bereiken.
- Via route constraints kan je o.a. eisen dat actionparameters van een bepaald type zijn, binnen een bepaalde range liggen of aan een regular expression voldoen. Je kan zelfs helemaal je eigen custom route constraint samenstellen.

## 19 Security

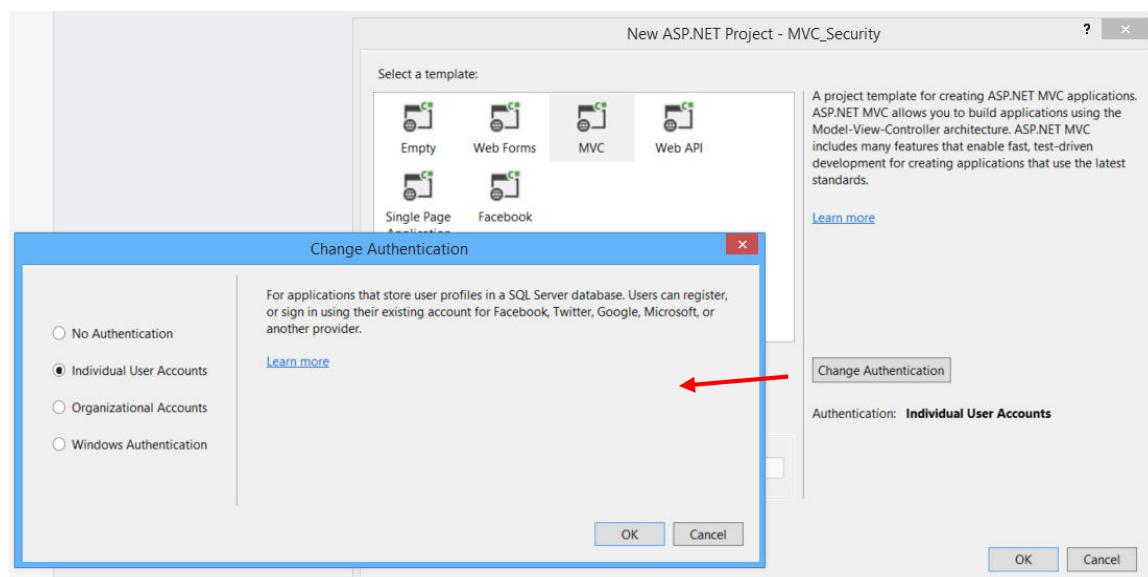
Wanneer we de toegang tot bepaalde webpagina's uit onze webapplication willen beperken tot bepaalde gebruikers of groepen van gebruikers dan hebben we een systeem nodig dat...

- ... voor ons gebruikersgegevens bewaart zoals een login en een wachtwoord
- ... de gebruikers van onze webapplicatie laat authenticeren via hun login en wachtwoord
- ... bijhoudt tot welke pagina's bepaalde gebruikers toegang hebben (authorization)

In dit hoofdstuk leer je hoe je een dergelijk systeem opzet.

### 19.1 Authentication

Wanneer we een nieuwe ASP.NET webapplicatie maken, dan kunnen we in het venster *New ASP.NET Project* via de knop *Change Authentication* een beveiligingssysteem kiezen.



De standaardkeuze is *Individual User Accounts*. Bij dit systeem houden we een userbestand bij in een SQL Server database en beslissen we welke users of groepen van users een pagina wel of niet kunnen bekijken. We spreken ook wel van Forms authentication. De user moet zich dus via een formulier registreren, aan- en afmelden. Deze optie is ideaal als jouw applicatie moet kunnen gebruikt worden door o.a. users die je helemaal niet kent.

Met de *Individual User Accounts* methode kan je gebruikers ook laten inloggen met hun Facebook, Google, Microsoft of Twitter account. Op die manier hoef je zelf geen wachtwoordgegevens op te slaan.

Bij *Organizational Accounts* gebruik je users uit een userbestand dat zich in de Cloud bevindt. Hiervoor wordt WIF (Windows Identity Foundation) gebruikt om users op te halen uit een Azure Active Directory of een Windows Server Active Directory.

*Windows Authentication* gebruikt eveneens users uit een Active Directory. Dit systeem wordt vooral gebruikt voor Intranet toepassingen. Dit is een webapplicatie die binnen een bedrijf wordt gebruikt en dus niet open staat voor zomaar iedereen op het internet.

De eerste optie *No Authentication* spreekt voor zich. Hier heb je geen enkele vorm van security.

In deze cursus gaan we ons houden bij de standaard keuze : *Individual User Accounts*.

## 19.2 Een beveilige ASP.NET MVC webapplicatie

- Maak een nieuwe ASP.NET webapplicatie met de naam *MVC\_Security* en selecteer de MVC template. In het venster *New ASP.NET Project* zie je rechts de default authentication modus voor MVC projecten: *Individual User Accounts*. Bekijk eventueel de andere mogelijkheden via de knop *Change Authentication* maar behoud de standaardwaarde. Klik tenslotte op OK om het project aan te maken.
- Start de webapplicatie en je krijgt een website te zien met reeds alle voorzieningen om je te registreren en je aan- en uit te loggen.
- Klik maar eens op de link *Register*, rechtsbovenaan de homepage en registreer jezelf.

Wanneer dit gelukt is veranderen de opties *Register* en *Log in* in *Hello <Usernaam> !* en *Log off*. De code voor deze menu-items kan je bekijken in de partial view *\_LoginPartial.cshtml* in de folder *Views/Shared*. Je ziet er onder andere dat de username wordt weergegeven met de code *User.Identity.GetUserName()* en dat deze menuitems leiden tot de actionmethods *Manage*, *Login* en *Register* uit de *AccountController*.

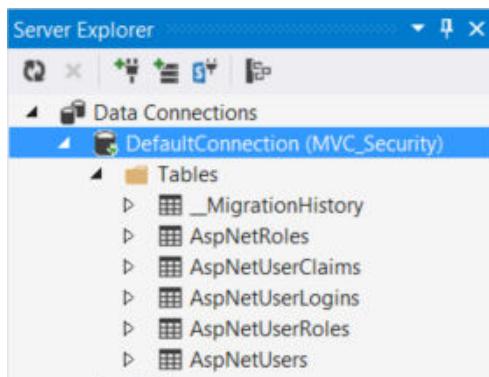
We gaan zo meteen verder met dit voorbeeld maar we bekijken eerst even de SQL Server database waarin de usergegevens worden bijgehouden.

## 19.3 De database

Bij de authorization methode *Individual User Accounts* wordt alle userinformatie opgeslagen in een SQL Server database. Je vindt deze database in de folder *App\_Data*. Wellicht moet je in de Solution Explorer eerst eens op de knop *Show All Files* klikken om deze database te zien.

De naam van de databasefile in de folder *App\_Data* bestaat standaard uit “aspnet-“, dan de naam van het project en tenslotte een datum en tijdstip.

- Dubbelklik de databasefile in de folder *App\_Data* zodat je de inhoud kan bekijken in de Server Explorer.



De database bevat 6 tabellen waarvan voorlopig enkel de tabel *AspNetUsers* voor ons nuttige data bevat, namelijk gegevens over de user die we reeds hebben aangemaakt.

- Klik met de rechtermuistoets op de tabel *AspNetUsers* en kies *Show Table Data*. Je ziet nu de gebruiker die je geregistreerd hebt.

In de SQL Server database hebben we ook nog een tabel *AspNetRoles*. Door users in roles onder te brengen kan je ze makkelijk in één keer bepaalde rechten toekennen. De tabel *AspNetUserRoles* is dan de tussentabel voor de veel-op-veel relatie tussen Users en Roles.

Verder zie je ook nog een tabel *AspNetClaims*. Werken met claims is een alternatieve manier om aan een user bepaalde rechten toe te kennen. Wij zullen niet met claims werken in deze cursus en dus ook niet met de table *AspNetClaims*.

De tabel *AspNetLogins* wordt gebruikt in een scenario waarbij je de website user toelaat zijn Facebook, Microsoft of andere account te gebruiken om in te loggen. In deze cursus zullen we niet werken met deze externe accounts.

### Securitydatabase op een eigen database server en/of in een eigen database

Als je dat wil, dan kan je de securitydatabase een andere naam geven en zelfs op een specifieke database server plaatsen. Meer nog, je kan zelfs een bestaande database uitbreiden met de nodige tabellen zodat die database kan gebruikt worden als securitydatabase. We geven hier aan hoe dat moet, in ons voorbeeld behoud je de standaardnaam en -server.

In het bestand Web.config in de root van jouw project geef je in de tag <connectionStrings> aan waar de securitydatabase zich bevindt (naam en plaats). Standaard ziet dit er als volgt uit :

```
<connectionStrings>
 <add name="DefaultConnection" connectionString="Data Source=(LocalDb)\v11.0;
 AttachDbFilename=|DataDirectory|\aspnet-MVC_Security-20140520122051.mdf;
 Initial Catalog=aspnet-MVC_Security-20140520122051;Integrated Security=True"
 providerName="System.Data.SqlClient" />
</connectionStrings>
```

De database wordt beheerd door de interne data server van Visual Studio : LocalDb. Verder zie je dat de database zich in de DataDirectory (App\_Data) bevindt. Wil je liever een andere naam voor de databasefile dan kan je de naam hier wijzigen. Vergeet het niet twee keer te doen : één keer voor AttachDbFilename en één keer voor Initial Catalog. Wanneer je daarna de applicatie start zal er een nieuwe (lege) databasefile gemaakt worden die de naam krijgt die jij opgegeven hebt.

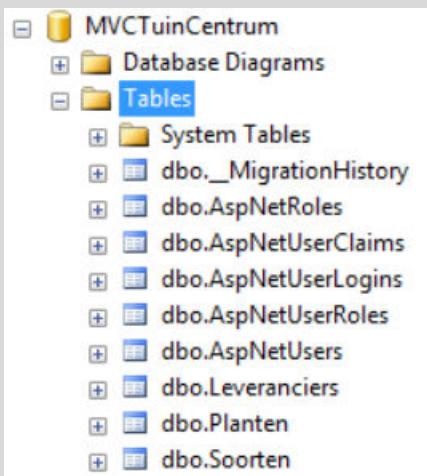
Gebruik je liever een andere databaseserver dan stel je dit in via Data Source. In onderstaand voorbeeld zal er een database DbSecurity worden aangemaakt op de database server SqlExpress :

```
<connectionStrings>
 <add name="DefaultConnection" connectionString="Data Source=.\SqlExpress;
 Initial Catalog=DbSecurity;Integrated Security=True"
 providerName="System.Data.SqlClient" />
</connectionStrings>
```

Je kan zelfs nog een stap verder gaan en een bestaande database gebruiken en deze voorzien van de nodige securitytabellen. In onderstaand voorbeeld gebruiken we de tuincentrum-database, dewelke zal uitgebreid worden met de nodige securitytabellen :

```
<connectionStrings>
 <add name="DefaultConnection" connectionString="Data Source=.\SqlExpress;
 Initial Catalog=MVCTuinCentrum;Integrated Security=True"
 providerName="System.Data.SqlClient" />
</connectionStrings>
```

Het resultaat :



Je kan dit gerust eens uitproberen in een testproject maar voor het vervolg van deze cursus gebruiken we gewoon de standaardwaarden.

## 19.4 Identity model

Via het EntityFramework worden de tabellen in de securitydatabase gekoppeld aan objecten uit de assembly *Microsoft.AspNet.Identity.EntityFramework*.

- Klap in Solution Explorer de map *References* open en dubbelklik op *Microsoft.AspNet.Identity.EntityFramework*.

In de Object Browser kan je nu deze assembly openklappen en dan vind je o.a. de class *IdentityUser<TKey, TLogin, TRole, TClaim>*. Dubbelklik deze class en je krijgt de bijhorende properties en methods te zien. *IdentityUser* bevat properties zoals een *Id*, een *UserName* en een *PasswordHash* (versleuteld wachtwoord). Op de property *Roles* komen we zodadelijk terug, *Claims* en *Logins* zouden we niet gebruiken in deze cursus.

Een *IdentityUser* heeft ook een read-only property *Roles*. Hiermee kan je achterhalen tot welke rollen de user behoort. De property *Roles* is evenwel geen verzameling van *IdentityRole* objecten maar wel van *IdentityUserRole* objecten. Oppassen dus.

Een *IdentityRole* stelt een rol voor waartoe een user kan behoren. Een *IdentityRole* heeft een *Id*, een *Name* en een verzameling *Users*. Dat zijn de users die tot die bepaalde role behoren. Deze verzameling is een read-only property.

Om makkelijk nieuwe users en roles te kunnen aanmaken, verwijderen en opzoeken zijn er de objecten *RoleStore* en *UserStore*. Het *UserStore* object bevat bovendien nog nuttige methods zoals *AddToRoleAsync()*, *IsInRoleAsync()*, enz.

In de hierop volgende paragrafen zullen we deze objecten en methods in de praktijk gebruiken.

## 19.5 Security administratie

De database bevat nu één user die we hebben aangemaakt door ons te registreren via de homepage. In de praktijk zullen we soms users op voorhand willen aanmaken en users tot bepaalde rollen laten behoren. Dit kan op verschillende manieren :

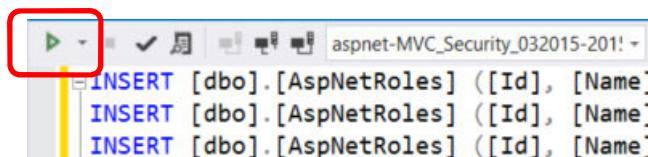
1. Je tikt de gegevens bijvoorbeeld via de SQL Server Management Studio zelf in in de tabellen van de securitydatabase.
2. Je ontwerpt zelf een compleet usermanagementsysteem waarmee je a.d.h.v. een aantal zelf gemaakte formulieren de nodige aanpassingen aanbrengt in de database.
3. Je gebruikt een zogenaamd seeding-mechanisme om de nodige data in de database te stoppen via code.

### 19.5.1 Gegevens zelf intikken

De eerste optie is veruit de simpelste maar praktisch gezien niet echt werkbaar. Hoewel je een *IdentityUser* perfect een hele simpele Id kan geven, is het invullen van een *PasswordHash* minder evident. Gegevens zelf intikken via SQL Server Management Studio is dus geen goede optie.

Wat we wél gaan doen is een aantal users en roles toevoegen via een SQL-script. Dit doen we enkel om in de hierop volgende paragrafen met echte data te kunnen werken.

- Klik in de Server Explorer met de rechtermuistoets op de Data Connection naar de securitydatabase : *DefaultConnection(MVC\_Security)*.
- Kies New Query.
- Plak in deze nieuwe SQL Query de inhoud van het bestand *Security.sql*. Je vindt dit bestand bij de oefenbestanden.
- Net boven het script is er een toolbar met helemaal links een Execute-knop. Klik op deze knop en het script wordt uitgevoerd.



```
aspnet-MVC_Security_032015-201!>
INSERT [dbo].[AspNetRoles] ([Id], [Name])
INSERT [dbo].[AspNetRoles] ([Id], [Name])
INSERT [dbo].[AspNetRoles] ([Id], [Name])
```

Je beschikt nu over enkele users en roles. Bekijk ze eventueel eens door in de Server Explorer met de rechtermuistoets te klikken op de tabel en te kiezen voor Show Table Data. Klik eventueel op de Refresh-knop (Shift+Alt+R) als je niet meteen een verandering ziet.

### 19.5.2 Een eigen user management systeem

Visual Studio bevat geen ingebakken systeem om op een gebruiksvriendelijke manier users en roles toe te voegen. Dit was ooit in vorige versies van VS anders. Bij wijze van voorbeeld gaan we zelf een user management systeem opzetten. Op deze manier leer je de classes en methods die je daarvoor nodig hebt nog beter kennen. We beperken ons tot de basic tools voor het aanmaken van users en roles. Als je dat wil kan je zelf nog extra toevoegingen aanbrengen.

- Maak een nieuwe empty controller aan met de naam *UserController*.
- Voeg een view toe aan de *Index* actionmethod en voeg er onderstaande code aan toe :

```
<p>Aangelogd? : @Request.IsAuthenticated</p>
<p>Ingelogde gebruiker : @User.Identity.Name</p>
<p>Authenticationtype : @User.Identity.AuthenticationType</p>
<p>Is een admin? : @User.IsInRole("admin")</p>
```

Via Request.IsAuthenticated kunnen we onder andere uitvissen of we al dan niet ingelogd zijn. De naam van de ingelogde gebruiker halen we op via de property Name van User.Identity. Aan de property AuthenticationType merk je dat er cookies worden gebruikt voor de authentication. Met de method IsInRole kan je testen of een user zich in één of andere rol bevindt.

- Start de applicatie, log in als één van de reeds aangemaakte users en ga naar .../user/index.

Een mogelijk resultaat :

The screenshot shows a web page with a title 'Index'. Below the title, there is a table-like structure containing the following data:

Aangelogd? :	True
Ingelogde gebruiker :	Fred.Flintstone@Bedrock.com
Authenticationtype :	ApplicationCookie
Is een admin? :	False

At the bottom of the page, there is a copyright notice: © 2015 - My ASP.NET Application.

- Log eventueel uit en ga terug naar de pagina .../user/index en vergelijk het resultaat.
- Voeg nu op de index-pagina van de usercontroller onderstaande hyperlinks toe. Deze leiden naar actions voor het beheren van de users en de roles :

```
<p>@Html.ActionLink("Userbeheer", "Userbeheer", "User")</p>
<p>@Html.ActionLink("Rolebeheer", "Rolebeheer", "User")</p>
```

- Voeg in de usercontroller twee actions toe :

```
using MVC_Security.Models;
using System.Data.Entity;
using Microsoft.AspNet.Identity.EntityFramework;

public ActionResult Userbeheer()
{
 ApplicationDbContext context = new ApplicationDbContext();
 IDbSet< ApplicationUser > alleUsers = context.Users;
 return View(alleUsers);
}
```

```
public ActionResult Rolebeheer()
{
 ApplicationDbContext context = new ApplicationDbContext();
 IDbSet<IdentityRole> alleRoles = context.Roles;
 return View(alleRoles);
}
```

We gebruiken een variabele *context* – je kan uiteraard ook een andere naam kiezen – die ons de lijst met users of roles uit de securitydatabase oplevert. Deze lijst geven we door aan de bijhorende view.

- Voeg nu aan beide methods een (empty) view toe.
- Vervang de code in *Userbeheer.cshtml* door de onderstaande code :

```
@model System.Data.Entity.IDbSet<MVC_Security.Models.ApplicationUser>
 @{
 ViewBag.Title = "Userbeheer";
}
<h2>Userbeheer</h2>

| ID | Username | Delete user |
|----|----------|-------------|
|----|----------|-------------|


```

- Voeg een nieuwe folder *Images* toe aan het project en kopieer er de afbeelding *delete.png* in.
- Voeg in Site.css onderstaande css code toe voor de zebra-class :

```
table.zebra, table.zebra td, table.zebra.th {
 border: 1px solid silver;
 border-collapse: collapse;
}

table.zebra thead {
 color: white;
 background-color: gray;
}

table.zebra tbody tr:nth-child(even) {
 background-color: #DDDDDD;
}
```

- Je kan dit al eens uitgeproberen : browse naar ...\\User\\Userbeheer en je krijgt een overzicht van de users in de securitydatabase.

Het verwijderen van een user werkt voorlopig nog niet op de pagina ...\\User\\Userbeheer en daar brengen we nu verandering in.

- Voeg in de UserController onderstaande actionmethods toe :

```
public ActionResult VerwijderUser(string id)
{
 ApplicationDbContext context = new ApplicationDbContext();
 var user = context.Users.Find(id);
 return View(user);
}

[HttpPost]
public ActionResult VerwijderUserDoorvoeren(string id)
{
 ApplicationDbContext context = new ApplicationDbContext();
 var user = context.Users.FirstOrDefault(u => u.Id == id);
 if (user != null)
 {
 context.Users.Remove(user);
 context.SaveChanges();
 }
 return RedirectToAction("Userbeheer");
}
```

- Voeg ook onderstaande viewcode toe aan de action *VerwijderUser* :

```
@model MVC_Security.Models.ApplicationUser
 @{
 ViewBag.Title = "Verwijderen";
}
<h2>@Model.UserName verwijderen?</h2>
<form method="post" action="~/User/VerwijderUserDoorvoeren/@Model.Id">
 <input type="submit" value="Delete" />
</form>
```

Deze actionmethods behoeven nog weinig uitleg. De eerste action, *VerwijderUser*, haalt de gegevens van een user op a.d.h.v. zijn id en presenteert een bevestigingsview. De tweede action, *VerwijderUserDoorvoeren*, verwijdert de user en gaat dan terug naar het Userbeheerscherm.

- Je kan het verwijderen van een user uitproberen.

Om Userbeheer helemaal af te werken moeten we nog een action *UserDetail* toevoegen.

- Voeg in de user controller een action *UserDetail* toe :

```
public ActionResult UserDetail(string id)
{
 ApplicationDbContext context = new ApplicationDbContext();
 var user = context.Users.Find(id);
 ViewBag.usernaam = user.UserName;
 var rolesVoorUser = user.Roles;
 return View(rolesVoorUser);
}
```

In deze action zoeken we eerst de user op a.d.h.v. de id en geven de usernaam door via de viewbag en de roles via het model.

- Voeg nu onderstaande view toe aan de UserDetail action.

```
@model List<Microsoft.AspNet.Identity.EntityFramework.IdentityUserRole>
@if
 ViewBag.Title = "UserDetail";
}

<h2>Detail van de user <i>@ViewBag.usernaam</i></h2>
<h3>Roles :</h3>

@if (Model.Count != 0)
{
 <table class="zebra">
 <thead>
 <tr>
 <th>Id</th>
 <th>Delete role for user</th>
 </tr>
 </thead>
 <tbody>
 @foreach (var userrol in Model) {
 <tr>
 <td>@userrol.RoleId</td>
 <td align="center">

 </td>
 </tr>
 }
 </tbody>
 </table>
}
else
{
 <p>User @ViewBag.usernaam behoort tot geen enkele rol.</p>
}
Terug naar userbeheer
```

We overlopen even deze code. Bovenaan tonen we de usernaam die we uit de viewbag hebben gehaald. Indien de lijst die we via het model hebben doorgekregen niet leeg is, dan tonen we een lijst met rollen waartoe de user behoort. Is de lijst wel leeg dan melden we dit. Helemaal onderaan voegen we een link toe naar userbeheer.

In de lijst tonen we de id van de rol en een verwijderknop.

- Je kan de userdetail al eens uitproberen. Het verwijderen van een rol implementeren we zodadelijk.
- Voor het verwijderen van een rol voeg je onderstaande actionmethod toe in UserController :

```
public ActionResult VerwijderRoleForMember(string userid, string roleid)
{
 ApplicationDbContext context = new ApplicationDbContext();
```

```

 var user = context.Users.FirstOrDefault(u => u.Id == userid); (1)
 var role = context.Roles.FirstOrDefault(r => r.Id == roleid); (2)
 if (user != null && role != null) (3)
 {
 IdentityUserRole userrole = user.Roles.SingleOrDefault(
 ur => (ur.UserId == userid && ur.RoleId == roleid)); (4)
 user.Roles.Remove(userrole);
 context.SaveChanges();
 }
 return RedirectToAction("UserDetail", "User", new { id = userid });
 }
}

```

Een klik op de verwijder-image start bovenstaande action op. Zoals je kan zien in de viewcode wordt er een userid en een roleid als parameter meegegeven. A.d.h.v. deze id's worden de user (1) en de role (2) opgezocht. We testen nog even of een dergelijke user en role werd gevonden (3) en als dat zo is dan kunnen we de userrole verwijderen. Dit gebeurt in een aantal stappen. Een user heeft een property Roles. Deze beschikt over een method Remove() die een userrole-item als parameter verwacht (5). Daarom halen we deze userrole op via een userid en een roleid (4). Tenslotte moeten we nog de SaveChanges()-method oproepen om de wijziging door te voeren en kunnen we via een redirect terug naar de userdetail.

- Je kan nu ook het verwijderen van een role voor een bepaalde user uitproberen.

Het userbeheer is nu klaar, we kunnen nu min of meer hetzelfde voorzien voor de rollen.

- Voeg onderstaande view toe aan de RoleBeheer action :

```

@model System.Data.Entity.IDbSet<Microsoft.AspNet.Identity.EntityFramework.IdentityRole>
 @{
 ViewBag.Title = "Rolebeheer";
}
<h2>Rolebeheer</h2>
<table class="zebra">
 <thead>
 <tr>
 <th>ID</th>
 <th>Name</th>
 <th>Delete role</th>
 </tr>
 </thead>
 <tbody>
 @foreach (var rol in Model) {
 <tr>
 <td>@rol.Id</td>
 <td>@rol.Name</td>
 <td align="center">
 @if (rol.Name != "admin")
 {

 }
 </td>
 </tr>
 }
 </tbody>
</table>

```

Deze code is vrijwel identiek aan de Userbeheer-code. We implementeren nu het verwijderen van een role.

- Voeg in de UserController onderstaande actions toe voor het verwijderen van een role:

```
public ActionResult VerwijderRole(string id)
{
 ApplicationDbContext context = new ApplicationDbContext();
 var role = context.Roles.Find(id);
 return View(role);
}

[HttpPost]
public ActionResult VerwijderRoleDoorvoeren(string id)
{
 ApplicationDbContext context = new ApplicationDbContext();
 var role = context.Roles.FirstOrDefault(u => u.Id == id);
 if (role != null)
 {
 context.Roles.Remove(role);
 context.SaveChanges();
 }
 return RedirectToAction("Rolebeheer");
}
```

Ook deze code is volledig analoog met de code voor het verwijderen van een user.

- Voeg onderstaande view toe aan de VerwijderRole action :

```
@model Microsoft.AspNet.Identity.EntityFramework.IdentityRole
 @{
 ViewBag.Title = "Role verwijderen";
}
<h2>@Model.Name verwijderen?</h2>
<form method="post" action("~/User/VerwijderRoleDoorvoeren/@Model.Id">
 <input type="submit" value="Delete" />
</form>
```

- Je kan het verwijderen van een role uitproberen.

In de lijst met rollen kunnen we de details ervan opvragen via een link naar RoleDetail. Deze keer voegen we er een extraatje aan toe t.o.v. UserDetail : onder de lijst met members van een role voegen we de mogelijk toe om een user toe te voegen aan de afgebeelde role. Het scherm ziet er dan ongeveer als volgt uit :

## Detail van de role admin

Members :

Name	Delete member from role
Steven	

Kies een user om aan deze rol toe te voegen :

[Terug naar rolebeheer](#)

Om dit te kunnen voorstellen moeten we net als bij userdetail een aantal zaken doorgeven aan de view. Hier is dit de naam van de role en de lijst met members. Daarbovenop moeten we ook de userlijst doorgeven voor de keuzelijst. In totaal één naam en 2 lijsten. Het resultaat van de keuzelijst komt dan ook nog eens in een stringproperty. Om dit allemaal via het model te kunnen doorgeven maken we best een ViewModel-class.

- Voeg in de folder *Models* onderstaande nieuwe class toe en noem deze *RoleDetailViewModel.cs* :

```
using System.Web.Mvc;
...
public class RoleDetailViewModel
{
 public string RoleName { get; set; }
 public string RoleId { get; set; }
 public ICollection<ApplicationUser> UsersUitRole { get; set; }
 public string SelectedUser { get; set; }
 public List<SelectListItem> SelectUser { get; set; }
}
```

- Voeg nu in de UserController onderstaande action toe :

```
public ActionResult RoleDetail(string id)
{
 ApplicationDbContext context = new ApplicationDbContext();
 var role = context.Roles.Find(id); (1)
 RoleDetailViewModel vm = new RoleDetailViewModel(); (2)
 vm.RoleName = role.Name;
 vm.RoleId = role.Id; (3)
 vm.UsersUitRole = (from usr in context.Users
 where usr.Roles.Any(r => r.RoleId == id)
 select usr).ToList(); (4)
 vm.SelectUser = new List<SelectListItem>();
 foreach (var user in context.Users)
 vm.SelectUser.Add(
 new SelectListItem { Text = user.UserName, Value = user.Id }); (5)
 return View(vm);
}
```

We zoeken eerst de role op a.d.h.v. de roleid (1). Dan maken we een.viewmodel en vullen de naam van de rol en de id in (3).

In de userdetail kregen we enkel de id's te zien van de rollen waartoe de user behoort. Dat is niet erg praktisch. Deze keer gaan we een stapje verder en bouwen we een lijst op van users. Dus niet enkel de id's maar dus ook de namen van de users. Een rol heeft een property *Users* maar dit zijn in tegenstelling tot wat de naam van deze property doet vermoeden geen *IdentityUsers* maar wel *IdentityUserRoles*. Dus moeten we de userlijst voor deze role anders samenstellen. We gaan alle users afgaan en kijken of ze een rol bevatten (in de property *Roles*) die gelijk is aan de gekozen rol (4). Deze lijst stoppen we dan in de viewmodelproperty *UsersUitRole*.

Dan bouwen we de keuzelijst op. Dit is een List van *SelectListItems* en we stoppen er alle beschikbare users in (5). Als *Text*-property kiezen we de naam van de user (dit is wat we zien in de lijst) en als *Value*-property de id (dit is wat er gekozen wordt).

- Voeg nu onderstaande view toe aan de action method RoleDetail :

```
@model MVC_Security.Models.RoleDetailViewModel
@{
 ViewBag.Title = "RoleDetail";
}
<h2>Detail van de role <i>@Model.RoleName</i></h2>
<h3>Members :</h3>
@if (Model.UsersUitRole.Count != 0) {
 <table class="zebra">
 <thead>
 <tr>
 <th>Name</th>
 <th>Delete member from role</th>
 </tr>
 </thead>
 <tbody>
 @foreach (var user in Model.UsersUitRole) {
 <tr>
 <td>@user.UserName</td>
 <td align="center">
 <a href="/User/VerwijderMemberFromRole?
 userid=@user.Id&roleid=@Model.RoleId">

 </td>
 </tr>
 }
 </tbody>
 </table>
}
else {
 <p>De role @Model.RoleName heeft geen members.</p>
}

@using (Html.BeginForm("MemberToevoegen", "User", new { roleid = Model.RoleId }, FormMethod.Post)) {
 <fieldset>
 Kies een user om aan deze rol toe te voegen :
 @Html.DropDownList("SelectUser")

 <p><input type="submit" value="Toevoegen"></p>
 </fieldset>
}
Terug naar rolebeheer
```

In het bovenste stuk van de view lijsten we de users uit. Deze keer kunnen we wél de naam afbeelden en dat is een stuk praktischer voor de gebruiker. Naast elke usernaam beelden we een kruisje af voor een verwijderfunctie.

Onder de lijst maken we een formulier. De action die moet uitgevoerd worden na keuze van een user is de actionmethod MemberToevoegen uit de UserController. Deze actionmethod maken we zodadelijk aan. We gebruiken uiteraard een Post-method want we moeten iets toevoegen.

We implementeren eerst de verwijderfunctie.

- Voeg onderstaande *VerwijderMemberFromRole* actionmethod toe in UserController :

```

public ActionResult VerwijderMemberFromRole(string userid, string roleid)
{
 ApplicationDbContext context = new ApplicationDbContext();
 var user = context.Users.FirstOrDefault(u => u.Id == userid);
 var role = context.Roles.FirstOrDefault(r => r.Id == roleid);
 if (user != null && role != null)
 {
 IdentityUserRole userrole = user.Roles.SingleOrDefault(
 ur => (ur.UserId == userid && ur.RoleId == roleid));
 user.Roles.Remove(userrole);
 context.SaveChanges();
 }
 return RedirectToAction("RoleDetail", "User", new { id = roleid });
}

```

Deze actionmethod is vrijwel identiek aan de VerwijderRoleForMember actionmethod. De meeste code hebben ze gemeenschappelijk en dus zouden we die eigenlijk in een aparte method moeten stoppen. Voor één keer zien we het door de vingers.

De code voor MemberToevoegen dan.

- Voeg onderstaande action method toe in de UserController :

```

[HttpPost]
public ActionResult MemberToevoegen(string RoleId, string SelectUser)
{
 ApplicationDbContext context = new ApplicationDbContext();
 var user = context.Users.FirstOrDefault(u => u.Id == SelectUser);
 var role = context.Roles.FirstOrDefault(r => r.Id == RoleId);

 if (user != null && role != null)
 {
 IdentityUserRole userrole = new IdentityUserRole();
 userrole.RoleId = RoleId;
 userrole.UserId = SelectUser;
 user.Roles.Add(userrole);
 context.SaveChanges();
 }

 return RedirectToAction("RoleDetail", "User", new { id = RoleId});
}

```

Deze method heeft twee parameters : een roleid en een userid die via de keuzelijst geselecteerd is. We zoeken de bewuste role en user op en als we die vinden maken we een nieuw IdentityUserRole object aan. De id-property's van dit object worden ingevuld en dan kan de IdentityUserRole toegevoegd worden aan de property Roles van de user.

- Je kan de RoleDetail volledig uitproberen.

Wie zin heeft in een kleine uitdaging kan de userdetail action method aanpassen zodat ook daar de namen van de rollen worden afgebeeld i.p.v. de id's.

Wij houden het hierbij, ons doel is immers bereikt en dat is jou een introductie geven in het gebruiken van IdentityUser, IdentityRole en IdentityUserRole objecten.

### 19.5.3 Seeding

Een seeding mechanisme is een systeem waarbij je via code gegevens invult in de securitydatabase. We proberen het in deze paragraaf even uit.

- Ga via *TOOLS – NuGet Package Manager – Package Manager Console* naar de Package Manager Console en tik *enable-migrations*. Druk op Enter.

Migrations is een entity framework feature dat naar de structuur van jouw database kijkt en vergelijkt met de classes in jouw model. Op basis van die vergelijking creëert het change scripts en voert die uit. Je kan die scripts automatisch laten uitvoeren door in de constructor van de class *Configuration.cs* de eigenschap *AutomaticMigrationsEnabled* op *true* te zetten. In deze cursus gaan we niet verder in op het migrations principe. Wat we wél gaan doen is gebruik maken van de *Seed()*-method in *Configuration.cs*. Via deze method kan je de securitydatabase voorzien van o.a. users en roles.

- In het project is een extra folder *Migrations* toegevoegd. Daarin bevindt zich een bestand *Configuration.cs* dat nu geopend zou moeten zijn in de editor. Mocht dit niet het geval zijn, open dan dit bestand in de folder *Migrations*.
- In *Configuration.cs* zet je in de constructor *AutomaticMigrationsEnabled* op *true*. Dit zorgt ervoor dat jouw database automatisch wordt bijgewerkt.

```
public Configuration()
{
 AutomaticMigrationsEnabled = true;
 ...
}
```

Er is ook reeds een *Seed* method voorzien die jouw database voorziet van data. Deze method gaan we zodadelijk uitvoeren via de package manager console. De *Seed* method bevat reeds enkele lijnen code die in commentaar staan. We voegen nu dergelijke code toe om een nieuwe user aan te maken.

- Wijzig de *Seed* method als volgt :

```
using Microsoft.AspNet.Identity;
using MVC_Security.Models;
...
protected override void Seed(ApplicationDbContext context) {
 var hasher = new PasswordHasher();
 context.Users.AddOrUpdate(u => u.UserName, new ApplicationUser { (1)
 UserName = "steven@vdab.be",
 PasswordHash=hasher.HashPassword("appelmoes") } (2)
);
}
```

Een woordje uitleg bij deze code. De method *Seed* heeft een parameter *context*. Deze is van het type *ApplicationDbContext*, uit sourcefile *IdentityModels.cs* in de folder *Models*. De variabele *context* bevat o.a. de userlijst (*context.Users*) en de rolelijst (*context.Roles*). In bovenstaande code gebruiken we de method *AddOrUpdate* van de userlijst *Users* om een nieuwe user aan te maken (2). In de voorbeeldcode die in de *Seed*-method in commentaar staat wordt deze method ook gebruikt. Wij gebruiken een variant van de *AddOrUpdate*-method, namelijk één die naast een username ook een geïncrypteerd wachtwoord meegeeft. Het wachtwoord encrypteren we met een *PasswordHasher* (1).

- Om deze Seed-method nu uit te voeren ga je terug naar de Package Manager Console en je tikt **update-database**.
- Druk op Enter en de method wordt uitgevoerd.

We kunnen nu controleren of de user effectief is toegevoegd in de securitydatabase.

- Klik met de rechtermuistoets in de Server Explorer op de table *AspNetUsers* en kies *Show Table Data*. Je zou nu de nieuwe user moeten zien staan. Is dit niet het geval dan moet je mogelijks eerst nog eens op de Refresh-knop net boven de lijst klikken.

Er is ook nog een tweede manier om users en roles aan te maken, namelijk via een UserManager die gebruik maakt van een UserStore en een RoleManager die een RoleStore gebruikt.

- Zet de zopas toegevoegde code in commentaar en vervang ze door onderstaande :

```
using Microsoft.AspNet.Identity.EntityFramework;
...
protected override void Seed(ApplicationDbContext context)
{
 if (!context.Users.Any(u => u.UserName == "Directeur@mvc.net")) (1)
 {
 var userStore = new UserStore< ApplicationUser >(context);
 var userManager = new UserManager< ApplicationUser >(userStore); (2)
 var roleStore = new RoleStore< IdentityRole >(context);
 var roleManager = new RoleManager< IdentityRole >(roleStore); (3)
 var user = new ApplicationUser { UserName = "Directeur@mvc.net" };
 userManager.Create(user, "appelmoes"); (4)
 var role = new IdentityRole { Name = "Management" };
 roleManager.Create(role); (5)
 userManager.AddToRole(user.Id, "Management");
 }
}
```

In bovenstaand voorbeeld maken we een nieuwe user aan met de naam *Directeur@mvc.net* maar we controleren eerst of er reeds een dergelijke user bestaat (1). Als dat nog niet het geval is maken we een UserStore en vervolgens een UserManager die gebruik maakt van deze UserStore (2). Beide objecten zijn classes uit de namespace *Microsoft.AspNet.Identity.EntityFramework*.

Op een gelijkaardige manier maken we voor de roles een RoleStore en een RoleManager aan (3).

We maken dan gebruik van de *Create*-method van de userManager om een gebruiker aan te maken (4). Deze method heeft twee parameters : een  *ApplicationUser*  object en een string voor het wachtwoord. Ook de  *RoleManager*  heeft een *Create*-method. Daar gebruiken we enkel een  *IdentityRole*  object als parameter (5). Tenslotte voegen we de user toe aan de rol met de method *AddToRole* van de  *UserManager*  (6).

- Tik in de Package Manager console opnieuw **update-database** en druk op Enter. De Seed-method wordt opnieuw uitgevoerd.
- Controleer opnieuw in de Server Explorer of er een user en een role is aangemaakt. Ook in de table *UserRoles* zou er een nieuwe entry moeten toegevoegd zijn.

## 19.6 Actions beveiligen met de [Authorize] annotation

Het is uiteindelijk de bedoeling dat we de users, nadat ze zich geauthenticeerd (= kenbaar gemaakt) hebben, de toelating geven (= authorize) om bepaalde webpagina's te bekijken. We kunnen dit laten afhangen van de naam van de user maar ook van de rol waartoe ze behoren.

Om te bepalen welke pagina's door welke users/roles mogen bekijken worden gebruik je de [Authorize]-annotation. We passen dit toe in een voorbeeld.

- Voeg aan jouw project twee extra controllers toe : LeraarController en StudentController.
- Voeg aan de index-methods van deze twee controllers telkens een view toe. Op die views geef je met een tekst aan welke soort gebruikers die pagina mogen bekijken.

De view die hoort bij de index-method van de StudentController :

```
@{
 ViewBag.Title = "Index";
}
<h2>Index</h2>
<p>Deze pagina is enkel toegankelijk voor studenten en leraren.</p>
```

En de view die hoort bij de index-method van de LeraarController :

```
@{
 ViewBag.Title = "Index";
}
<h2>Index</h2>
<p>Deze pagina is enkel toegankelijk voor leraren.</p>
```

- Op de index-pagina van de HomeController pas je de div met class "row" als volgt aan :

```
...
<div class="row">
 <div class="col-md-4">
 <h2>@Html.ActionLink("Studenten", "Index", "Student")</h2>
 <p><a class="btn btn-default"
 href("~/Student">Naar de pagina voor studenten »</p>
 </div>
 <div class="col-md-4">
 <h2>@Html.ActionLink("Leraren", "Index", "Leraar")</h2>
 <p><a class="btn btn-default"
 href("~/Leraar">Naar de pagina voor leraren »</p>
 </div>
 <div class="col-md-4">
 <h2>@Html.ActionLink("User administrators", "Index", "User")</h2>
 <p><a class="btn btn-default"
 href("~/User">Naar de pagina voor user admins »</p>
 </div>
 </div>
</div>
```

We geven nu a.d.h.v. annotations aan dat bepaalde pagina's niet door iedereen mogen bekijken worden.

- Voeg in de StudentController een annotation toe :

```
[Authorize]
public class StudentController : Controller
{
 ...
}
```

Met `[Authorize]` bovenaan de controllerclass geven we aan dat alle methods in de controller beveiligd zijn en dus enkel toegankelijk voor ingelogde gebruikers. Methods die wél toegankelijk moeten zijn voor niet-ingelogde gebruikers krijgen een annotation `[AllowAnonymous]` bovenaan.

Wil je enkel bepaalde methods beveiligen dan plaats je de annotation `[Authorize]` vlak boven de te beveiligen actionmethod.

- Start de applicatie en probeer, zonder eerst in te loggen, van de homepagina naar de Studentenpagina te gaan. Je zal geen toegang krijgen en uitgenodigd worden om je in te loggen.
- Bekijk even de url in de browser. Daarin lees je dat je na het succesvol inloggen naar de defaultpagina van de StudentController gestuurd zal worden.
- Log in met gebruikersnaam `Student1@mvc.net` en wachtwoord `Student1` en je komt inderdaad op de indexpagina van de StudentController terecht die enkel voor geregistreerde gebruikers toegankelijk is.

Wil je specifieker bepalen welke ingelogde gebruikers een controller(method) kunnen uitvoeren dan geef je de annotation `Authorize` een parameter `Users` mee.

Hieronder geven we aan dat enkel de gebruikers `Leraar1@mvc.net`, `Leraar2@mvc.net` en `Leraar3@mvc.net` toegang hebben tot de actions uit de LeraarController :

```
[Authorize(Users="Leraar1@mvc.net,Leraar2@mvc.net,Leraar3@mvc.net")]
public class LeraarController : Controller
{
 ...
}
```

- Voeg bovenstaande annotation toe in de LeraarController en probeer er onder verschillende gebruikersaccounts naartoe te browsen. De wachtwoorden voor de leraren zijn respectievelijk `Leraar1`, `Leraar2` en `Leraar3`.

In plaats van users te vermelden in de `Authorize` annotation kan je ook meteen een role noteren. Op die manier kan je een ganse groep users in één keer toegang geven tot een controller of een action in een controller.

De pagina's binnen de UserController kunnen enkel door users uit de role `admin` bekijken worden :

```
[Authorize(Roles = "Admin")]
public class UserController : Controller
{
 ...
}
```

Pagina's voor studenten kunnen door users uit de roles leraren én studenten bekijken worden.

```
[Authorize(Roles = "Student, Leraar")]
public class StudentController : Controller
{
 ...
}
```

- Voeg bovenstaande annotations toe in de UserController en de StudentController en probeer er opnieuw onder verschillende gebruikersaccounts naartoe te browsen. De wachtwoorden voor de studenten zijn Student1, Student2,... Het wachtwoord voor de Admin is Admin0.

## 19.7 Een webapplicatie met authentication publiceren

Ook deze webapplicatie gaan we nu publiceren naar IIS. De securitydatabase die nu in de folder App\_Data staat gaan we overzetten naar SQL Server.

We beginnen met een (lege) database aan te maken in SQL Express.

- Start SQL Server Management Studio en connecteer met je plaatselijke SQL Express.
- Klik met de rechtermuistoets op het onderdeel Databases en kies New Database. Vul de naam in van de nieuwe database, bijvoorbeeld Security en klik op OK.

We geven nu de user ASPNET waarmee we vanuit IIS de database gaan benaderen rechten op deze nieuwe database.

- Open het onderdeel Security en daaronder Logins. Klik met de rechtermuistoets op de user ASPNET en kies Properties.
- Kies de page User Mapping en selecteer de database Security. Vink eronder de optie db\_owner aan en klik op OK.

De securitydatabase beschikt voorlopig nog niet over de nodige tabellen en dus zeker niet over de nodige users. Daar gaan we voor zorgen via migrations. Bij het publiceren zullen de nodige tabellen worden aangemaakt dankzij code die nu reeds in de class <timestamp>\_InitialCreate.cs in de folder Migrations staat. Om de nodige data in die tabellen te stoppen moeten we code schrijven in de Seed-method in Configuration.cs die eveneens in de folder Migrations te vinden is.

- Breng nu in de Seed-method de nodige code aan zodat de database voorzien wordt van de nodige users en roles :

```
protected override void Seed(MVC_Security.Models.ApplicationDbContext context)
{
 // This method will be called after migrating to the latest version.
 var roleStore = new RoleStore<IdentityRole>(context);
 var roleManager = new RoleManager<IdentityRole>(roleStore);
 var userStore = new UserStore<ApplicationUser>(context);
 var userManager = new UserManager<ApplicationUser>(userStore);
 ApplicationUser user;

 if (!context.Roles.Any(u => u.Name == "Student"))
 roleManager.Create(new IdentityRole { Name = "Student" });

 if (!context.Roles.Any(u => u.Name == "Leraar"))
 roleManager.Create(new IdentityRole { Name = "Leraar" });

 if (!context.Roles.Any(u => u.Name == "Admin"))
 roleManager.Create(new IdentityRole { Name = "Admin" });
```

```

 if (!context.Users.Any(u => u.UserName == "Admin@mvc.net")) {
 user = new ApplicationUser { UserName = "Admin@mvc.net" };
 userManager.Create(user, "Admin0");
 userManager.AddToRole(user.Id, "Admin");
 }

 if (!context.Users.Any(u => u.UserName == "Leraar1@mvc.net")) {
 user = new ApplicationUser { UserName = "Leraar1@mvc.net" };
 userManager.Create(user, "Leraar1");
 userManager.AddToRole(user.Id, "Leraar");
 }

 if (!context.Users.Any(u => u.UserName == "Leraar2@mvc.net")) {
 user = new ApplicationUser { UserName = "Leraar2@mvc.net" };
 userManager.Create(user, "Leraar2");
 userManager.AddToRole(user.Id, "Leraar");
 }

 if (!context.Users.Any(u => u.UserName == "Leraar3@mvc.net")) {
 user = new ApplicationUser { UserName = "Leraar3@mvc.net" };
 userManager.Create(user, "Leraar3");
 userManager.AddToRole(user.Id, "Leraar");
 }

 if (!context.Users.Any(u => u.UserName == "Student1@mvc.net")) {
 user = new ApplicationUser { UserName = "Student1@mvc.net" };
 userManager.Create(user, "Student1");
 userManager.AddToRole(user.Id, "Student");
 }

 if (!context.Users.Any(u => u.UserName == "Student2@mvc.net")) {
 user = new ApplicationUser { UserName = "Student2@mvc.net" };
 userManager.Create(user, "Student2");
 userManager.AddToRole(user.Id, "Student");
 }

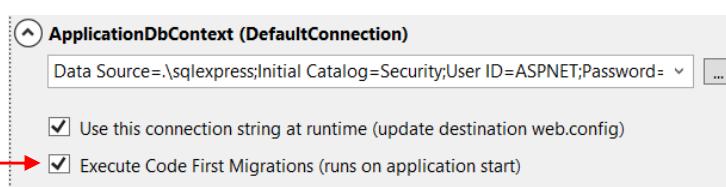
 if (!context.Users.Any(u => u.UserName == "Student3@mvc.net")) {
 user = new ApplicationUser { UserName = "Student3@mvc.net" };
 userManager.Create(user, "Student3");
 userManager.AddToRole(user.Id, "Student");
 }

 if (!context.Users.Any(u => u.UserName == "Student4@mvc.net")) {
 user = new ApplicationUser { UserName = "Student4@mvc.net" };
 userManager.Create(user, "Student4");
 userManager.AddToRole(user.Id, "Student");
 }
 }
}

```

We zijn nu klaar om de webapplicatie te publiceren op IIS.

- Klik in de Solution Explorer met de rechtermuistoets op het project en kies Publish.
- Maak een nieuw Custom profile en geef het een naam.
- Op het tabblad Connection kies je als Server voor localhost. Als site name kies je bijvoorbeeld Default Web Site/Security.
- Als remote connectionstring verwijst je naar de Security database met de user ASPNET. Vink ook de optie *Execute Code First Migrations* aan. Hierdoor zal in de Security database de nodige tabellen worden aangemaakt en de Seed method worden uitgevoerd.



- Klik op Publish.
- Browse nu naar <http://localhost/Security>. Probeer de verschillende logins uit en controleer of je naar de juiste pagina's kan browsen.

## 19.8 Samenvatting

- Je kan jouw webapplicatie beveiligen door deze enkel open te stellen voor gekende gebruikers. Gebruikergegevens kunnen opgeslagen zijn in een eigen database (internet toepassingen), een Active Directory (intranet toepassingen). Daarnaast kan je ook Facebook, Google, Microsoft of Twitter accounts gebruiken.
- Gebruik je een eigen database dan kan je die opslaan in de folder App\_Data binnen jouw webapplicatie of op een SQL Server.
- De user database is opgesteld volgens het Identity Model en bevat tables zoals AspNetUsers voor de users, AspNetRoles voor rollen waartoe users kunnen behoren. AspNetUserRoles is de tussentabel tussen beide voornoemde tabellen.
- Er zijn verschillende manieren om de securitydatabase van data te voorzien. Eén ervan is zelf de data intikken via een tool als SQL Server Management Studio of de VS Server Explorer. Handiger is de data 'seeden' via code (voor de ontwikkelaar) of een userbeheersysteem (laten) ontwerpen waarbij de data door een (liefst geprivilegerde) websitegebruiker wordt ingevuld.

## 19.9 Oefening

Maak een nieuwe, beveiligde MVC-webapplicatie om de bierenlijst te raadplegen en te beheren. Je kan je beperken tot de mogelijkheden raadplegen, toevoegen en verwijderen of je kan via scaffolding een totale applicatie maken. In ieder geval mag iedereen de biereninformatie raadplegen, enkel geregistreerde gebruikers die behoren tot een rol *Administrators* mogen bieren, soorten en leveranciers toevoegen en verwijderen.



## 20 Bootstrap

---

In de voorbije hoofdstukken heb je af en toe enkel css-classes gezien zoals jumbotron, row, col-md-4, etc. Dit zijn allemaal Bootstrap css-classes. In deze paragraaf geven we heel kort een woordje uitleg over Bootstrap. Verwacht hier geen cursus Bootstrap maar eerder een verduidelijking van enkele classes die je meer dan waarschijnlijk tegenkomt als je een MVC Webapplicatie opbouwt.

### 20.1 Bootstrap-classes in MVC web applicaties

Bootstrap is een open source front-end framework dat ervoor zorgt dat jouw user interface er goed uitziet, zowel op een gewoon pc-scherm als op een tablet of een phone. Je kan dit bijvoorbeeld testen door de applicatie via de Opera Mobile Emulator te bekijken. Je zal vaststellen dat deze er ook nog goed uitziet op een phone.

Een standaard MVC 5 web applicatie bevat reeds de nodige bootstrap javascript libraries (folder *Scripts*) en bootstrap css (folder *Content*) zodat je bootstrap meteen kan gebruiken.

- Open terug de webapplication *MVC\_Security*.
- Open in de folder *Views/Shared* het bestand *\_Layout.cshtml*. Je ziet dat daar de bootstrap css-classes ‘navbar’, ‘navbar-header’, e.d. worden gebruikt.
- Open het bestand *Views/Home/Index.cshtml*. Daar worden standaard de css-classes ‘jumbotron’, ‘row’ en ‘col-md-4’ toegepast.

Bootstrap voorziet ook een aantal tags van een default opmaak. Op die manier voorkomt het dat content in diverse browsers op een andere manier wordt weergegeven.

- Zet in het bestand *Views/Shared/\_Layout.cshtml* in de head-tag de regel `@Styles.Render("~/Content/css")` maar eens in commentaar en start de webapplicatie. Jouw website ziet er meteen een stuk minder hip uit.

In *\_Layout.cshtml* wordt er een class ‘container’ toegepast. Deze zorgt ervoor dat de content een max-width heeft en voorzien wordt van marge. Laat je deze class weg dan plakt alles links en rechts tegen de rand van het scherm.

- Haal in *\_Layout.cshtml* de code `@Styles.Render("~/Content/css")` die je net in commentaar hebt gezet terug uit commentaar zodat ze opnieuw gebruikt wordt.
- Vervang de class ‘container’ door een niet bestaande class ‘testcontainer’ zodat de bootstrap class wordt uitgeschakeld :

```
<div class="navbar navbar-inverse navbar-fixed-top">
 <div class="testcontainer">
 ...
 </div>
</div>
<div class="testcontainer body-content">
 ...
</div>
...

```

- Start de applicatie opnieuw. Zowel het menu bovenaan als de body-tekst neemt nu de volledige breedte van het venster in beslag.
- Stop de applicatie en vervang ‘testcontainer’ terug in ‘container’.

## 20.2 Het bootstrap grid systeem

Op de pagina *Views/Home/Index.cshtml* staan 3 artikels naast elkaar. Ze zitten in een div met class ‘row’. In die row zitten divs met class ‘col-md-4’. De 4 geeft aan dat de kolom 4 units breed is. Als je weet dat een row maximaal 12 units breed kan zijn, dan weet je dat hiermee de volledig rij gevuld is. Wijzig je dit in ‘col-md-12’ dan zal de eerste kolom meteen de ganse breedte van het scherm innemen. Is de optelsom van alle units binnen één row meer dan 12 dan schuift een kolom op naar de volgende regel.

- Geef de 1<sup>e</sup> kolom eens een breedte van 6, maak de 2<sup>e</sup> kolom 4 units breed en de 3<sup>e</sup> kolom 5 units.

De 3<sup>e</sup> kolom komt op een volgende regel te staan want de totale breedte is nu 15. In de VS-editor geeft een groene lijn al aan dat de maximale breedte van een rij overschreven is :

```
<div class="row">
 <div class="col-md-6">
 <h2>@Html.ActionLink("Studenten", "Index", "Student")

 <p>Naar de studenten</p>
 </div>
 <div class="col-md-4">
 <h2>@Html.ActionLink("Leraren", "Index", "Leraar")

 <p>Naar de leraren</p>
 </div>
 <div class="col-md-2">
 <h2>@Html.ActionLink("User administratoren", "Index", "UserAdmin")

 <p>Naar de user administratoren</p>
 </div>
</div>
```

Je kan een cel – zo noemen we ook een stuk tekst in een kolom binnen een row – verplaatsen naar rechts door een extra class te gebruiken : col-md-offset-<units>. Dus bijvoorbeeld col-md-offset-3.

- Geef in Index.cshtml de 3 kolommen elk een breedte van 3 units en geef de 2<sup>e</sup> kolom een extra css-class ‘col-md-offset-3’ mee :

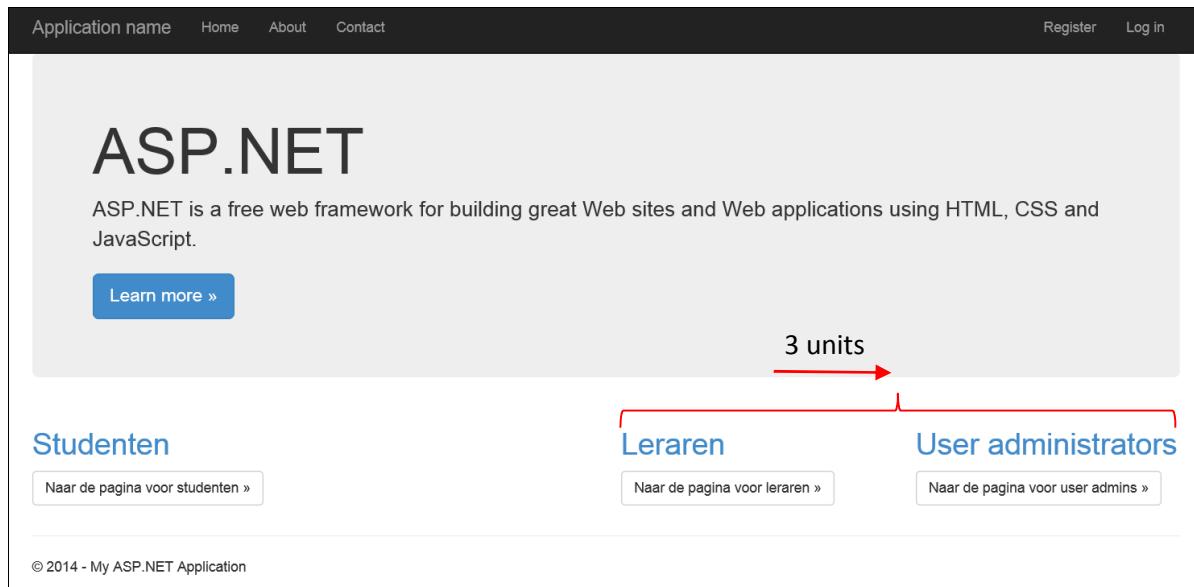
```
<div class="row">
 <div class="col-md-3">
 <h2>@Html.ActionLink("Studenten", "Index", "Student")

 <p>Naar de studenten</p>
 </div>
 <div class="col-md-3 col-md-offset-3">
 <h2>@Html.ActionLink("Leraren", "Index", "Leraar")

 <p>Naar de leraren</p>
 </div>
 <div class="col-md-3">
 <h2>@Html.ActionLink("User administratoren", "Index", "UserAdmin")

 <p>Naar de user administratoren</p>
 </div>
</div>
```

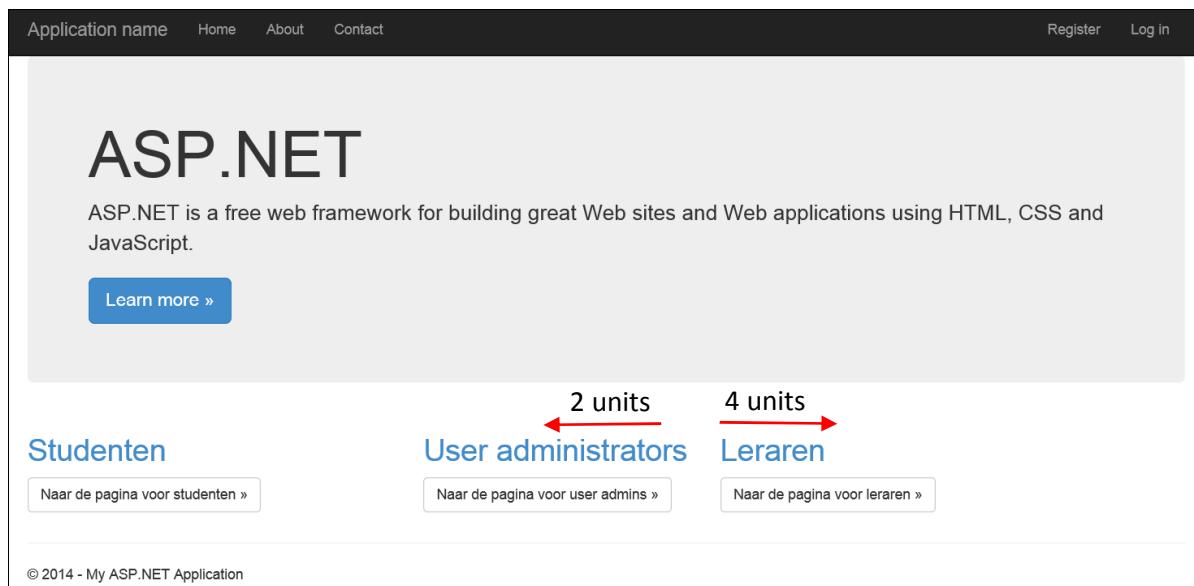
Het resultaat : de tweede cel schuift 3 eenheden op naar rechts, alles wat rechts van deze cel staat schuift eveneens op.



Je kan ook cellen naar links of naar rechts verschuiven met `col-md-push-<units>` en `col-md-pull-<units>`. Het verschil met `col-md-offset` is dat een verschuiven hier geen gevolgen heeft voor de andere cellen. Je loopt wel het risico dat cellen mekaar overlappen.

- Geef de 3 kolommen elk een breedte van 3 units en geef de 2<sup>e</sup> kolom een extra css-class ‘`col-md-push-4`’ mee, de 3<sup>e</sup> kolom een extra class ‘`col-md-pull-2`’.

Het resultaat : de 3<sup>e</sup> kolom komt links van de 2<sup>e</sup> kolom te staan.



Dit alles en nog veel meer over het grid-systeem lees je ook op <http://getbootstrap.com/css/#grid>.

In de voorbije voorbeelden hebben we de cellen steeds een class gegeven waarvan de naam begint met ‘col-md’. Met ‘md’ geven we aan dat deze class van toepassing is op devices van ‘medium’-grootte. Vervang je ‘md’ door ‘lg’ dan zal de class van toepassing zijn op large devices ( $\geq 1200px$ ), ‘xs’ geldt voor extra small devices ( $<768px$ ) en ‘sm’ voor small devices ( $\geq 768px$ ).

Je kan een cel, afhankelijk van het bedoelde medium, een verschillende breedte geven door ze css-classes toe te kennen voor diverse media. Bijvoorbeeld :

```
<div class="col-md-3 col-sm-6">
```

In dit voorbeeld zal de cel 3 units breed zijn bij een medium device en 6 units bij een small device.

Een andere toepassing van de medium-aanduiding lg/md/sm/xs is het verbergen van tags voor devices van een bepaalde grootte. Je kan tags verbergen met de classes hidden-lg, hidden-md, hidden-sm en hidden-xs voor respectievelijk large, medium, small en extra small devices. Analoog zijn er classes om tags zichtbaar te maken : visible-lg, visible-md, visible-sm en visible-xs.

We proberen één en ander uit.

- Geef de 3 divs een class “col-md-3”. De middelste geef je ook nog eens een class “hidden-sm”.
- Bekijk het resultaat en verklein geleidelijk aan de breedte van het venster.

Je zal zien dat op een bepaald moment de middelste kolom verdwijnt én dat de beide cellen onder elkaar komen te staan. Dit terwijl er nochtans voldoende plaats is om de twee cellen naast elkaar te plaatsen. We gaan dit verbeteren.

- Geef de 1<sup>e</sup> en de 3<sup>e</sup> cell een bijkomende class “col-sm-6” mee.

Het resultaat : kolom 1 en 3 staan naast elkaar.

### 20.3 Bootstrap componenten

Bootstrap beschikt over een pak componenten. Dat zijn o.a. glyphs, navigation bars, dropdownlists, progress bars, enz. Een volledige lijst vind je op <http://getbootstrap.com/components>.

Zoals reeds gezegd bevat *index.cshtml* standaard de componenten : *jumbotron* en *navbar*. Een div voorzien van de class jumbotron wordt wat groter weergegeven en krijgt ook een gekleurde achtergrond. Een gelijkaardig component is *well*. Deze zorgt eveneens voor een gekleurde achtergrond maar een net iets kleinere tekstgrootte.

Navigationbars zijn ietsje ingewikkelder om te gebruiken. Kijk maar eens in *\_Layout.cshtml* hoe zo een navigationbar is samengesteld. Hier volstaat het niet één enkele div te voorzien van een class maar wel een combinatie van div-, span-, button-, ul- en li-tags.

Er zijn een aantal varianten voor een navbar. Je hebt de classes *nav-tabs* en *nav-pills*. De eerste geeft de menu-items een tab-uitzicht (bovenste hoeken van het listitem zijn afgerond en er is een lijn zichtbaar onder de items), de tweede geeft ze 4 afgeronde hoeken (geen lijn onder de items).

Een voorbeeld :

- Voeg helemaal onderaan de pagina *Home/Index.cshtml* het onderstaande toe :

```
<div>
 <ul class="nav nav-tabs">
 Home
 About
 Contact

</div>
```

Het resultaat als je de cursor op het middelste item laat rusten :



- Vervang de class *nav-tabs* door *nav-pills* :



Nog andere alternatieven zijn de classes *pagination* en *breadcrumb*.

- Voeg helemaal onderaan de pagina *Home/Index.cshtml* het onderstaande toe :

```
<div>
 <ul class="breadcrumb">
 VDAB
 IT opleidingen
 Ontwikkelaar met C#

</div>
<div>
 <ul class="pagination">
 Prev
 1
 2
 3
 4
 5
 Next

</div>
```

Het resultaat : bij breadcrumbs worden de listitems naast elkaar weergegeven met een schuin streepje ertussen. Pagination geeft de items weer in vakjes.

VDAB / IT opleidingen / Ontwikkelaar met C#

Prev 1 2 3 4 5 Next

Als je dat wil kan je listitems disabled of active maken met de gelijknamige classes. Hieronder is het listitem "2" active (inverse weergave) en listitem "5" disabled (tekst is lichter gekleurd) :

```
<ul class ="pagination">
 Prev
 1
 <li class="active">2
 3
 4
 <li class="disabled">5
 Next

```



## 20.4 Bootstrap tekstopmaak

Naast de standaardopmaak die Bootstrap toepast op headers en paragraphs beschikt Bootstrap ook over classes om tekstopmaak toe te kennen. In *Index.cshtml* vind je bijvoorbeeld een class *lead* terug. De meeste tekstopmaak-classes beginnen evenwel met *text-*, *label-* of *alert-*. Hieronder een voorbeeld waarbij we de *lead*-tekst een ander opmaakclass gegeven hebben :

```
<div class="jumbotron">
 <h1>ASP.NET</h1>
 <p class="label label-info">ASP.NET is a free web framework for building ...</p>
 <p class="label label-danger">ASP.NET is a free web framework for building ...</p>
 <p class="label label-warning">ASP.NET is a free web framework for building ...</p>
 <p class="text text-info">ASP.NET is a free web framework for building ...</p>
 <p class="text text-success">ASP.NET is a free web framework for building ...</p>
 <p class="text text-warning">ASP.NET is a free web framework for building ...</p>
 <p class="alert alert-danger">ASP.NET is a free web framework for building ...</p>
 <p class="alert alert-success">ASP.NET is a free web framework for building ...</p>
 <p class="alert alert-warning">ASP.NET is a free web framework for building ...</p>
 <p>
 Learn more »</p>
</div>
```

Het resultaat :

- Labels worden in een iets kleiner lettertype in reverse weergegeven en hebben een volle achtergrondkleur.
- Text wordt afhankelijk van de bijkomende class in een speciale kleur weergegeven : blauw voor info, groen voor success, oranje voor warning, rood voor danger, feller blauw voor primary,...
- Alert-classes zorgen net als labels voor een volle achtergrond maar in een iets lichtere tint dan de tekstkleur.

# ASP.NET

ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.

ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.

ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.

ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.

ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.

ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.

ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.

ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.

ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.

[Learn more »](#)

## 20.5 Bootstrap forms

Je kan ook makkelijk forms layouten met bootstrap. Ook hiervan heb je al een voorbeeld in een standaard webapplication, meerbepaald op de page *Register.cshtml*. Daar krijgt de form een class *form-horizontal* mee waardoor de labels en velden naast elkaar staan i.p.v. onder elkaar – tenminste als er voldoende ruimte is. De input-velden kregen de class *form-control*. Daardoor worden ze iets groter weergegeven en krijgen ze afgeronde hoeken. De class *control-label* heeft als effect dat een label rechts wordt uitgelijnd en zo dichter bij de control komt te staan.

De registratie form met de table-opmaak :

Email	<input type="text"/>
Password	<input type="password"/>
Confirm password	<input type="password"/>
	<input type="button" value="Register"/>

## 20.6 Bootstrap tables

Een table kan je voorzien van de Bootstrap class *table*. De cellen krijgen dan wat padding en de table krijgt de breedte van de container waartoe deze behoort. Deze container kunnen zoals in één van voorbije paragrafen werd aangetoond bijvoorbeeld een breedte geven met een class col-md-<units>.

Naast een breedte krijgt de table via de class *table* ook extra opmaak in de vorm van horizontale lijnen. Andere opmaak classes voor tables zijn o.a. *table-hover*, *table-striped*,...

Een voorbeeld :

- Voeg onderstaande code toe helemaal onderaan *Home/Index.cshtml*.

```
<div class="row">
 <div class="col-md-6">
 <table class="table">
 <thead>
 <tr><th>Links</th><th>Rechts</th></tr>
 </thead>
 <tbody>
 <tr><td>Linksboven</td><td>Rechtsboven</td></tr>
 <tr><td>Linksonder</td><td>Rechtsonder</td></tr>
 </tbody>
 </table>
 </div>
</div>
```

Het resultaat : de table krijgt de helft van de beschikbare breedte.

<a href="#">Studenten</a>	<a href="#">Leraren</a>	<a href="#">User administrators</a>
<a href="#">Naar de pagina voor studenten »</a>	<a href="#">Naar de pagina voor leraren »</a>	<a href="#">Naar de pagina voor user admins »</a>
<b>Links</b>	<b>Rechts</b>	
Linksboven	Rechtsboven	

Linksonder	Rechtsonder
------------	-------------

## 20.7 Samenvatting

- Bootstrap is een open source front end framework dat een schat aan css classes bevat. Op deze manier geef je jouw webapplication met weinig moeite niet alleen een professioneler uitzicht maar krijgt het ook een herkenbare stijl.

## 21 Project Cultuurhuis

In dit hoofdstuk pakken we een concreet praktijk voorbeeld aan. We maken een web applicatie waarmee gebruikers tickets kunnen reserveren voor allerlei voorstellingen in een cultuurcentrum.

### 21.1 Opgave

#### 21.1.1 Een voorstelling kiezen

De voorstellingen die geboekt kunnen worden zijn ingedeeld in genres : dans, humor, klassiek,... Op de beginpagina toon je de genres als hyperlinks naast elkaar in alfabetische volgorde.

Wanneer de gebruiker een genre kiest, ziet hij alle voorstellingen van dat genre uitgelijst in een tabel. Voorstellingen waarvan de datum reeds voorbij is worden niet afgebeeld. Verder zijn de voorstellingen oplopend gesorteerd weergegeven op datum.

Datum	Titel	Uitvoerders	Prijs	Vrije plaatsen	Reserveren
26/11/14 20:00	Wilde dingen	Kopergietery	€ 5,00	175	<a href="#">Reserveren</a>
26/11/14 20:00	Een hond in de nacht	Speeltheater Holland	€ 6,00	0	<a href="#">uitverkocht</a>

Enkel bij voorstellingen die nog vrije plaatsen hebben ziet de gebruiker een hyperlink *Reserveren*. Is de voorstelling uitverkocht, dan wordt dit aangegeven en is er geen hyperlink *Reserveren*.

Zijn er geen voorstellingen voor het gekozen genre, dan wordt dit gemeld :

### **21.1.2 Het aantal tickets ingeven**

Als de gebruiker de hyperlink *Reserveren* bij een voorstelling aanklikt, ziet hij een nieuwe pagina waarop hij kaarten voor de gekozen voorstelling kan reserveren.

Voorstelling	White Light White Heat - The Velvet Undergr.
Uitvoerders	Bea Van der Maat & Dr Kloot Per W
Datum	06/12/14 20:00
Prijs	€ 5,50
Vrije plaatsen	200
Plaatsen	1 <input type="button" value="x"/>

Via de button *Tickets kiezen* komt de gebruiker terug op de eerste pagina. Onderaan kan de gebruiker bij *Plaatsen* een geheel getal intikken tussen 1 en het aantal vrije plaatsen. De ingevulde waarde moet gevalideerd worden en bij foutieve ingave wordt er een gepaste foutmelding gegeven.

Vrije plaatsen	200
Plaatsen	<input type="text" value="202"/> <span style="border: 1px solid blue; padding: 2px;">X</span>
U moet een waarde invoeren tussen 1 en 200	

De geplaatste reservaties worden bijgehouden in een tijdelijk winkelmandje. Er wordt dus nog niks in een database opgeslagen maar wel in de HTTP session.

Als een gebruiker een tweede keer dezelfde voorstelling reserveert, dan toon je in het invoervak bij *Plaatsen* het aantal dat de gebruiker reeds eerder intikte bij de eerste reservering van die voorstelling. Hij kan dit getal wijzigen.

### 21.1.3 Overzicht van de gevraagde tickets

Na het klikken op de knop *Reserveren* komt de gebruiker op een overzichtspagina terecht waar zijn winkelmandje op is afgebeeld.

Datum	Titel	Uitvoerders	Prijs	Plaatsen	
26/11/14 20:00	Wilde dingen	Kopergieterij	€ 5,00	2	<input type="checkbox"/>
07/12/14 20:00	White Light White Heat - The Velvet Undergr.	Bea Van der Maat & Dr Kloot Per W	€ 5,50	4	<input type="checkbox"/>

Te betalen: € 32,00

© 2014 - Het Cultuurhuis

Bovenaan zijn er twee buttons. Met de button *Tickets kiezen* gaat de gebruiker terug naar de eerste pagina waar opnieuw een voorstelling kan geselecteerd worden. Daarnaast brengt een button *Kassa* de gebruiker naar een bevestigingspagina (zie later).

In een tabel worden de gereserveerde tickets weergegeven. Helemaal rechts is er een kolom met checkboxen. Door een lijn aan te vinken en onderaan op *Selectie verwijderen* te klikken haalt de gebruiker tickets weg uit zijn mandje.

Van zodra er tickets in het mandje liggen moet er op de homepagina een button bijkomen die de gebruiker naar de overzichtspagina (winkelmandje) brengt en ook een button naar een bevestigingspagina (kassa).

### 21.1.4 Identificatie van de klant

Via de button *Kassa* komt de gebruiker op een pagina waar hij zich identificeert en de reservatie definitief vastlegt.

**Het Cultuurhuis**

Tickets kiezen »   Winkelmandje »

Gebruikersnaam

Wachtwoord

Zoek me op   Ik ben nieuw

Bevestigen

© 2014 - Het Cultuurhuis

Een bestaande klant tikt zijn gebruikersnaam en wachtwoord in. Daarna klikt hij op de button 'Zoek me op'. Als deze gebruiker bestaat en het wachtwoord klopt dan ziet de gebruiker zijn gegevens onder de knoppen en wordt de knop *Bevestigen* actief. De textboxen *Gebruikersnaam* en *Wachtwoord* en de knoppen 'Zoek me op' en 'Ik ben nieuw' worden dan inactief.

Stap 1: Wie ben je?

Gebruikersnaam

Wachtwoord

Zoek me op   Ik ben nieuw

Bert Bibber Stripstraat 5 9000 Gent

Stap 2: Bevestigen

Bevestigen

Wanneer de gebruiker een niet bestaande gebruikersnaam/wachtwoord combinatie intikt, dan toon je in de plaats van zijn gegevens een foutmelding.

## Stap 1: Wie ben je?

Gebruikersnaam

Wachtwoord

Verkeerde gebruikersnaam of wachtwoord

Een nieuwe gebruiker klikt op de knop 'Ik ben nieuw'. Dan ziet hij volgend invulformulier :

# Het Cultuurhuis



[Tickets kiezen »](#)

[Winkelmandje »](#)

[Kassa »](#)

### Vul jouw gegevens in

Voornaam

Familienaam

Straat

Huisnr

Postcode

Gemeente

Gebruikersnaam

Wachtwoord

Wachtwoord bevestigen

Alle velden zijn verplicht in te vullen. De inhoud van het herhaalpaswoordveld moet hetzelfde zijn als het paswoordveld. De gebruikersnaam mag nog niet voorkomen in de database. Eventuele validatiefouten komen naast het veld waarvoor er een validatiefout optreedt.

Voornaam	Jan	
Familienaam	Janssen	
Straat	Frankrijklei	
Huisnr		<i>Huisnr is verplicht in te vullen!</i>
Postcode	2000	
Gemeente	Antwerpen	
Gebruikersnaam	bert	<i>Een klant met deze gebruikersnaam komt al voor in de database. Kies een andere naam.</i>
Wachtwoord	***	
Wachtwoord bevestigen	*	<i>Wachtwoord bevestigen verschilt van Wachtwoord. Probeer opnieuw.</i>
<input type="button" value="Registreer me"/>		

Is alles goed ingevuld dan wordt de gebruiker toegevoegd in de database en keert hij terug naar de bevestigingspagina en zijn de gebruikersgegevens afgebeeld net als bij opgezochte gebruikers.

### 21.1.5 Overzicht gelukte/mislukte reservaties

Op de bevestigingspagina kan de gebruiker vervolgens op de knop *Bevestigen* klikken. Je legt dan de reservaties in het mandje vast in de database : je vermindert het aantal vrije plaatsen in de table *Voorstellingen* tenzij het aantal vrije plaatsen ondertussen kleiner is geworden dan de gewenste plaatsen omdat een andere gebruiker ondertussen ook voor die voorstelling gereserveerd heeft. De reservatie voor deze voorstelling kan dan niet doorgaan.

Als je het aantal wél kon verminderen dan voeg je een record toe aan de table Reserveringen. Daarna toon je aan de gebruiker een overzicht van alle gelukte en mislukte reservaties.

# Het Cultuurhuis

[Tickets kiezen »](#)

### Gelukte reserveringen

Datum	Titel	Uitvoerders	Prijs	Plaatsen
26/11/14 20:00	Wilde dingen	Kopergietery	€ 5,00	2
07/12/14 20:00	White Light White Heat - The Velvet Undergr.	Bea Van der Maat & Dr Kloot Per W	€ 5,50	4

### Mislukte reserveringen

Datum	Titel	Uitvoerders	Prijs	Plaatsen
12/12/14 20:00	Cry like a man, part 2	J. Blaute, Paul Michiels & Roland	€ 6,00	2

© 2014 - Het Cultuurhuis

Na het overzicht maak je het mandje leeg.

## 21.2 Een oplossing

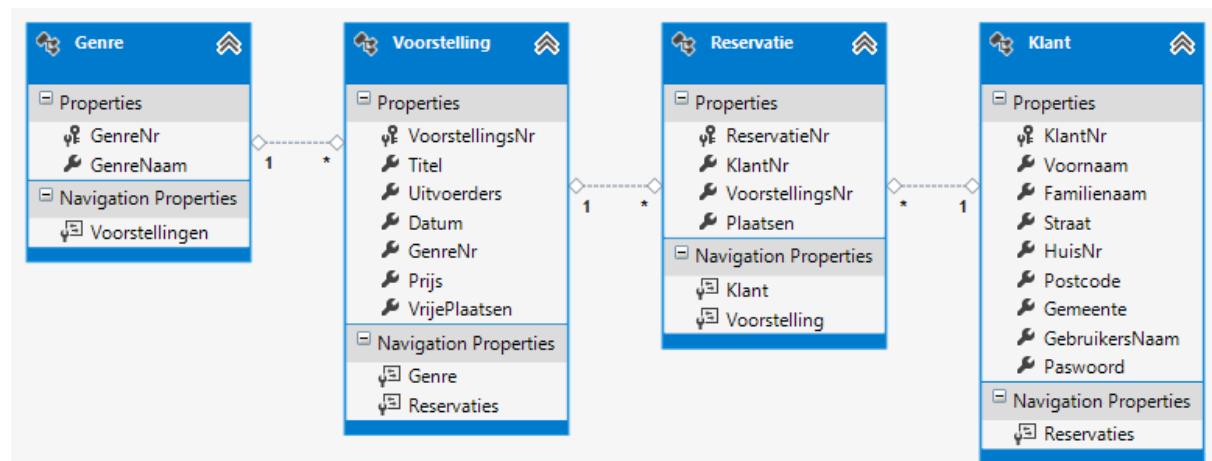
We geven hier een beschrijving hoe je dit project met MVC kan oplossen. Zoals dikwijls zijn er meerdere mogelijke oplossingen, dit is er één van.

Maak een nieuwe ASP.NET Web Application, bijvoorbeeld met de naam *MVC\_Cultuurhuis*. Kies de *MVC template* en wijzig de authentication naar *No Authentication*.

### 21.2.1 Het data model

Open de SQL Server Management Studio en log in op een SQL Server naar wens. Open het sql-script *createCultuurhuis.sql* en voer het uit. Afhankelijk van het tijdstip waarop je deze cursus doorneemt zal je een aantal data van voorstellingen uit de tabel *Voorstelling* moeten wijzigen. Zorg ervoor dat de datum van een klein deel van de voorstellingen in het verleden ligt. Doe dit a.d.h.v. een update sql-statement.

Voeg in Visual Studio in de folder *Models* een *ADO.NET Entity Data Model* toe met de naam *Cultuur*. Kies *EF Designer from database* en klik op *Next*. Kies vervolgens *New connection*. Kies de juiste SQL Server instance en databasename en klik op *OK*. Kies *Next*, vink alle tables aan en kies *Finish*.



Zet de namen van de entities om in hun enkelvoudsvorm. Voor de entity *Genres* hernoem je eerst het veld *Genre* naar *GenreNaam*. De navigation properties *Genres* van de entity *Voorstelling* zet je om naar zijn enkelvoudsvorm. Doe hetzelfde voor de navigation properties *Klanten* en *Voorstellingen* in de entity *Reservatie*. Bewaar de edmx-file.

### 21.2.2 De class CultuurService

We maken een class die de nodige methods bevat waarmee de gewenste gegevens uit de database worden gehaald of erin bewaard, een serviceclass dus. Voeg een class *CultuurService* toe in een nieuwe folder *Services*.

### 21.2.3 De page Voorstellingen

Op de eerste page beelden we een horizontale lijst af met alle genres. Wanneer er een genre gekozen wordt, dan komt er een subtitel onder de lijst met daarin de naam van het genre. Onder de subtitel worden alle voorstellingen uitgelijst die tot het genre behoren.

Om al deze gegevens op te vragen uit de database voorzien we 3 methods in de class *CultuurService* : GetAllGenres(), GetGenre() en GetAlleVoorstellingenVanGenre().

```
using MVC_Cultuurhuis.Models;
...
public List<Genre> GetAllGenres()
{
 using (var db = new CultuurHuisMVCEntities()) {
 return db.Genres.OrderBy(m=>m.GenreNaam).ToList();
 }
}
```

De method GetAllGenres() levert ons een list van Genre-objecten, gesorteerd op naam.

Je voegt ook een method GetGenre() toe die aan de hand van een id het bijhorende genre levert :

```
public Genre GetGenre(int? id)
{
 using (var db = new CultuurHuisMVCEntities()) {
 return db.Genres.Find(id);
 }
}
```

Een vrij eenvoudige Linq-query levert ons het genre op dat bij een id hoort. Waarom de parameter nullable is wordt straks duidelijk.

Tenslotte voegen we nog onderstaande , derde method toe aan de CultuurService class :

```
public List<Voorstelling> GetAlleVoorstellingenVanGenre(int? id)
{
 using (var db = new CultuurHuisMVCEntities())
 {
 var query = from voorstelling in db.Voorstellingen.Include("Genre")
 where voorstelling.Datum >= DateTime.Today &&
 voorstelling.GenreNr == id
 orderby voorstelling.Datum
 select voorstelling;
 return query.ToList();
 }
}
```

Deze method levert ons een List van Voorstelling-objecten op, gesorteerd op datum en waarvan bovendien de datum niet in het verleden ligt.

Voeg aan de HomeController een private variabele van het type CultuurService toe zodat we in de diverse actionmethods van deze service gebruik kunnen maken.

```
using MVC_Cultuurhuis.Services;
...
public class HomeController : Controller
{
 private CultuurService db = new CultuurService();
 ...
}
```

Voor de eerste pagina kiezen we ervoor éénzelfde actionmethod, Index(), te gebruiken, ongeacht of de gebruiker reeds een genre heeft gekozen of niet. We zoeken in de actionmethod eerst alle nodige

gegevens op en geven die door aan de bijhorende view via een class *VoorstellingenInfo*. Deze ziet er als volgt uit :

```
public class VoorstellingenInfo
{
 public Genre Genre { get; set; }
 public List<Genre> Genres { get; set; }
 public List<Voorstelling> Voorstellingen { get; set; }
}
```

Voeg bovenstaande class toe in de folder *Models*.

Wanneer de gebruiker voor het eerst op de indexpagina komt zal *Genre* null zijn en de bijhorende lijst van voorstellingen een lege List. Om die reden hebben twee van de drie methods in *CultuurService* een nullable parameter.

Pas nu de Index-actionmethod in de HomeController als volgt aan :

```
using MVC_Cultuurhuis.Models;
...
public ActionResult Index(int? id)
{
 var voorstellingenInfo = new VoorstellingenInfo();
 voorstellingenInfo.Genres = db.GetAllGenres();
 voorstellingenInfo.Voorstellingen = db.GetAlleVoorstellingenVanGenre(id);
 voorstellingenInfo.Genre = db.GetGenre(id);
 return View(voorstellingenInfo);
}
```

Wijzig de bijhorende view als volgt :

```
@model MVC_Cultuurhuis.Models.VoorstellingenInfo
@{
 ViewBag.Title = "Index";
}

Het Cultuurhuis </h1>

@if (Model.Genre == null) {
 <p class="lead alert alert-info">Kies een voorstellingsgenre</p>
}

 @foreach (var genre in Model.Genres)
 {
 var url = Url.Action("Index", "Home", new { id = genre.GenreNr });
 if (Model.Genre != null && Model.Genre.GenreNaam.Equals(genre.GenreNaam))
 {
 <li class="active">
 @genre.GenreNaam

 }
 else
 {

 @genre.GenreNaam

 }
 }

```

```

@if (Model.Genre != null)
{

 if (Model.Voorstellingen.Count() > 0)
 {
 <table class="table">
 <thead>
 <tr>
 <th>Datum</th>
 <th>Titel</th>
 <th>Uitvoerders</th>
 <th>Prijs</th>
 <th>Vrije plaatsen</th>
 <th>Reserveren</th>
 </tr>
 </thead>
 <tbody>
 @foreach (var voorstelling in Model.Voorstellingen)
 {
 <tr>
 <td class="col-md-2">@Html.DisplayFor(m => voorstelling.Datum)</td>
 <td class="col-md-3">@voorstelling.Titel</td>
 <td class="col-md-2">@voorstelling.Uitvoerders</td>
 <td class="col-md-1">@Html.DisplayFor(m => voorstelling.Prijs)</td>
 <td class="col-md-2">@voorstelling.VrijePlaatsen</td>
 @if (voorstelling.VrijePlaatsen > 0)
 {
 var urlReserveren = Url.Action("Reserveren", "Home",
 new { id = voorstelling.VoorstellingsNr });
 <td class="col-md-2">
 Reserveren
 </td>
 }
 else
 {
 <td class="col-md-2">
 uitverkocht
 </td>
 }
 </tr>
 }
 </tbody>
 </table>
 }
 else
 {
 <p class="alert alert-warning">
 Er zijn geen voorstellingen van dit genre beschikbaar.
 </p>
 }
}

```

Deze view heeft een object van het type `VoorstellingenInfo` als Model.

Bovenaan komt er een titel met een figuurtje. Je kan alle afbeeldingen die bij dit project horen in één keer toevoegen aan een nieuwe folder `Images`.

Bij een eerste passage op deze pagina zal Genre leeg zijn. Als dat het geval is dan komt er een boodschap met de uitnodiging om een genre te kiezen.

Om de lijst met Genres af te beelden doorlopen we de list Model.Genres. Voor elk genre stellen we een url samen die verwijst naar de Index-method met een parameter GenreNr. Om de genres af te beelden gebruiken we de bootstrap classes *nav* en *nav-pills* en om het gekozen genre wat te accentueren geven we dit een class *active*.

Bij een eerste doortocht op de indexpagina is de view nu compleet (Model.Genre is null). Indien deze view echter het resultaat is van een klik op één van de genres dan is Model.Genre niet null en tonen we alle bijhorende voorstellingen.

Indien het gekozen genre voorstellingen heeft (*Model.Voorstellingen.Count() > 0*) dan maken we een html-tabel, voorzien van de bootstrap class *table*.

Per voorstelling tonen we de verschillende velden. Voor de velden *Datum* en *Prijs* voorzien we een aangepaste opmaak. We passen de techniek van de buddy classes toe. Dit betekent dat we in de folder *Models* twee extra classes moeten maken : een class *VoorstellingUitbreiding.cs* en een class *VoorstellingProperties.cs*.

De inhoud van *VoorstellingUitbreid.cs* :

```
[MetadataType(typeof(VoorstellingProperties))]
public partial class Voorstelling
{ }
```

En van *VoorstellingProperties.cs* :

```
public class VoorstellingProperties
{
 [DisplayFormat(DataFormatString = "{0:dd/MM/yy HH:mm}")]
 public System.DateTime Datum { get; set; }
 [DisplayFormat(DataFormatString = "{0:€ #,##0.00}")]
 public decimal Prijs { get; set; }
 public short VrijePlaatsen { get; set; }
}
```

Het tijdstip van de voorstelling geven we weer a.d.h.v. de datum en het uur in 24-uur notatie (vandaar hoofdletter H). De prijs krijgt een euro-teken vooraan en twee cijfers na de komma.

We geven beide velden weer met deze opmaak door *Html.DisplayFor()* te gebruiken.

We plaatsen enkel een hyperlink in de kolom Reserveren indien het aantal vrije plaatsen voor de voorstelling groter is dan nul. De hyperlink verwijst naar een action Reserveren, het voorstellingsnr wordt meegegeven als parameter. Zijn er geen vrije plaatsen dan geven we de tekst 'uitverkocht' in een opvallend lettertype (bootstrap class *label-danger*) weer.

Zijn er geen voorstellingen voor het gekozen genre dan geven we gewoon een tekst weer i.p.v. een tabel.

In deze view hebben we in de tabel voor de opmaak van de velden prijs en aantal ook nog wat css gebruikt. We deden dit omdat een getal best rechts wordt uitgelijnd in een kolom. Dan staan de eenheden netjes onder elkaar, de tientallen, enz. Er is ook een andere mogelijke werkwijze die we in onderstaand kader verder uitleggen maar omdat we deze opmaak in meerdere tabellen nodig hebben kozen we voor de oplossing met css.

In Site.css voeg je onderstaande css-code toe :

```
.table td:nth-child(4) {
 text-align:right;
 padding-right:1em;
}

.table td:nth-child(5) {
 text-align:right;
 padding-right:3em;
}

.table th:nth-child(4) {
 text-align:right;
 padding-right:1em;
}

.table th:nth-child(5) {
 text-align:right;
 padding-right:3em;
}
```

Hiermee voorzie je in alle bootstrap-tables de 4<sup>e</sup> en de 5<sup>e</sup> kolom van extra css. Zowel de headers als de inhoud van de cellen van deze kolommen worden rechts uitgelijnd. En aan de rechterkant brengen we extra padding aan om te voorkomen dat de prijs of het aantal tegen de rechterkant van de tabelkolom wordt geplakt. Voor de prijskolom is de padding 1em breed, voor het aantal 3em. Zoals je in de code van de view kan zien hebben we de kolommen via de bootstrap class col-md-x een bepaalde breedte gegeven. Daardoor kunnen we hier een verschillende padding gebruiken.

We kunnen de uitlijning van de prijs- en aantal-kolommen ook via aangepaste formattering verkrijgen. Dit gebeurt in 2 stappen.

1. In *VoorstellingProperties.cs* breng je via een *DisplayFormat* annotation een formattering aan. Dit zou er als volgt kunnen uitzien :

```
[DisplayFormat(DataFormatString = "{0,3:#0}")].
```

Deze formattering zorgt voor de nodige voorlooptekens. Het getal zal dus altijd over een breedte van 3 tekens worden weergegeven ongeacht het aantal af te beelden cijfers.

2. In de view *Index.cshtml* beeld je de geformatteerde waarde van het veld af met de functie *Html.DisplayFor()*. De browser zal echter de voorlooptekens die je net toegevoegd hebt weer weglaten. Browsers houden immers geen rekening met spaties. In de view moet je dus ook nog eens de *Html*-helperfunctie *Html.Raw()* toepassen op de functie *Html.DisplayFor()*. De code in de view is dan als volgt :

```
<td class="col-md-1">@Html.Raw(Html.DisplayFor(m => voorstelling.Prijs))</td>
```

Je kan het project gerust al eens uitproberen. Controleer best eens of de datums van de voorstellingen min of meer in de buurt van de huidige datum komen. Wijzig eventueel enkele data zodat er voorstellingen worden afgebeeld.

## 21.2.4 De page Reserveren

Voor de page *Reserveren* voorzien we in de HomeController eerst een get actionmethod *Reserveren*. Deze toont ons de gegevens van een geselecteerde voorstelling. Deze gegevens worden eerst opgezocht aan de hand van het voorstellingsnr en doorgegeven via een object van het type Voorstelling.

De actionmethod :

```
public ActionResult Reserveren(int id)
{
 Voorstelling gekozenVoorstelling = db.GetVoorstelling(id);
 return View(gekozenVoorstelling);
}
```

Dit is de code van de method GetVoorstelling() die we in de CultuurService neerschrijven :

```
public Voorstelling GetVoorstelling(int? id)
{
 using (var db = new CultuurHuisMVCEntities())
 {
 return db.Voorstellingen.Find(id);
 }
}
```

De view die hoort bij de *Reserveren* actionmethod :

```
@model MVC_Cultuurhuis.Models.Voorstelling
 @{
 ViewBag.Title = "Reserveren";
}
<div class="jumbotron">
 <h1>Het Cultuurhuis </h1>
 <p>

 Tickets kiezen »

 </p>
</div>
<p class="lead alert alert-info">
 Aantal tickets voor deze voorstelling
</p>
<form class="form-horizontal" method="post"
 action("~/Home/Reserveer/@Model.VoorstellingsNr">
 <div class="form-group">
 <label class="col-md-2 control-label">Voorstelling</label>
 <div class="col-md-10">
 <p class="form-control-static">@Model.Titel</p>
 </div>
 </div>
 <div class="form-group">
 <label class="col-md-2 control-label">Uitvoerders</label>
 <div class="col-md-10">
 <p class="form-control-static">@Model.Uitvoerders</p>
 </div>
 </div>
 <div class="form-group">
 <label class="col-md-2 control-label">Datum</label>
 <div class="col-md-10">
 <p class="form-control-static">@Html.DisplayFor(m => m.Datum)</p>
 </div>
 </div>
```

```

 </div>
 </div>
 <div class="form-group">
 <label class="col-md-2 control-label">Prijs</label>
 <div class="col-md-10">
 <p class="form-control-static">@Html.DisplayFor(m => m.Prijs)</p>
 </div>
 </div>
 <div class="form-group">
 <label class="col-md-2 control-label">Vrije plaatsen</label>
 <div class="col-md-10">
 <p class="form-control-static">@Model.VrijePlaatsen</p>
 </div>
 </div>
 <div class="form-group">
 <label class="col-md-2 control-label">Plaatsen</label>
 <div class="col-md-10">
 <input value="1" required="required" min="1" type="number"
 max="@Model.VrijePlaatsen" name="aantalPlaatsen"
 class="form-control" />
 </div>
 </div>
 <div class="form-group">
 <div class="col-md-offset-2 col-md-10">
 <input value="Reserveren" type="submit" class="btn btn-default" />
 </div>
 </div>
</form>
```

Bovenaan komt er een hyperlink om terug naar de voorstellingenpagina te gaan. Eronder komt een form waarin de voorstellingengegevens afgebeeld worden en er een aantal plaatsen kan gekozen worden. Voorlopig gebruiken we een defaultwaarde van 1. Verderop passen we dit aan en kijken we eerst of de gebruiker reeds voor deze voorstelling een reservatie heeft gedaan maar dat is voor later. Om de input te valideren gebruiken we attributen zoals min en max, required en type. Wanneer de gebruiker op de submit-knop klikt dan wordt een actionmethod *Reserveer* gekozen met als id het voorstellingsnr.

Op de velden Prijs en Datum passen we de opmaak toe uit het model en voor het invulveld gebruiken we een input-tag van een attribuut *type* dat we op *number* zetten.

De verwerking van het formulier gebeurt door een actionmethod *Reserveer()* uit de HomeController. Het voorstellingsnummer wordt als parameter meegegeven :

```

[HttpPost]
public ActionResult Reserveer(int id)
{
 uint aantalPlaatsen = uint.Parse(Request["aantalPlaatsen"]);
 var voorstellingInfo = db.GetVoorstelling(id);

 if (aantalPlaatsen > voorstellingInfo.VrijePlaatsen) {
 return RedirectToAction("Reserveer", "Home", new { id = id });
 }

 Session[id.ToString()] = aantalPlaatsen;

 return RedirectToAction("Mandje", "Home");
}
```

We halen eerst het aantal op dat ingevuld werd in het inputveld. Dit veld kreeg de naam *aantalPlaatsen* en kan via die naam uit de Request gehaald worden. De voorstellingsgegevens halen we op via de method *GetVoorstelling()* uit de CultuurService.

Als er meer tickets gevraagd worden dan beschikbaar dan laten we het invulformulier terug zien. Zijn er wel voldoende tickets dan slaan we het aantal op als een sessionvariabele met als waarde voor de naam het voorstellingsnummer. Vervolgens komen we terecht in een actionmethod *Mandje()* voor een overzicht van de geplaatste reservaties.

### 21.2.5 Het winkelmandje

Voor de pagina waarop de inhoud van het winkelmandje wordt getoond maken we volgende actionmethod :

```
public ActionResult Mandje()
{
 decimal teBetalen = 0;
 List<MandjeItem> mandjeItems = new List<MandjeItem>();

 foreach (string nummer in Session)
 {
 int voorstellingsnummer;
 if (int.TryParse(nummer, out voorstellingsnummer))
 {
 Voorstelling voorstelling = db.GetVoorstelling(voorstellingsnummer);
 if (voorstelling != null)
 {
 MandjeItem mandjeItem =
 new MandjeItem(voorstellingsnummer, voorstelling.Titel,
 voorstelling.Uitvoerders, voorstelling.Datum,
 voorstelling.Prijs, Convert.ToInt16(Session[nummer]));

 teBetalen += (mandjeItem.Plaatsen * mandjeItem.Prijs);

 mandjeItems.Add(mandjeItem);
 }
 }
 }
 ViewBag.teBetalen = teBetalen;
 return View(mandjeItems);
}
```

We initialiseren het totaalbedrag van de tickets op 0 en voorzien een list van MandjeItem-objecten. Een MandjeItem is een class met de voorstellingsgegevens en het aantal gewenste tickets :

```
public class MandjeItem
{
 public int VoorstellingsNr { get; set; }
 public string Titel { get; set; }
 public string Uitvoerders { get; set; }
 [DisplayFormat(DataFormatString = "{0:dd/MM/yy HH:mm}")]
 public System.DateTime Datum { get; set; }
 [DisplayFormat(DataFormatString = "{0:€ #,##0.00}")]
 public decimal Prijs { get; set; }
 public short Plaatsen { get; set; }
```

```

public MandjeItem(int nr, string titel, string uitvoerders,
 DateTime datum, decimal prijs, short plaatsen)
{
 VoorstellingsNr = nr;
 Titel = titel;
 Uitvoerders = uitvoerders;
 Datum = datum;
 Prijs = prijs;
 Plaatsen = plaatsen;
}
}

```

Via een `DisplayFormat`-annotation voorzien we Datum en Prijs van opmaak en we voegen ook een constructor toe.

Om in de actionmethod `Mandje()` de list `mandjeItems` op te vullen overlopen we de nummers die opgeslagen zijn in de Session. Omdat we later ook een klantobject in sessionscope bewaren testen we of de keywaarde wel een int is. We gebruiken de method `GetVoorstelling` om de voorstellingsgegevens op te zoeken. Het aantal gewenste tickets halen we uit de Session via het voorstellingsnummer.

Het totaal te betalen bedrag wordt tenslotte ook bijgewerkt en, als alle bestellingen zijn overlopen, doorgegeven aan de bijhorende view via de `ViewBag`. De List van `MandjeItems` gegeven we door als parameter.

De bijhorende view :

```

@model List<MVC_Cultuurhuis.Models.MandjeItem>
 @{
 ViewBag.Title = "Mandje";
}
<div class="jumbotron">
 <h1>Het Cultuurhuis </h1>
 <p>
 <a href="@Url.Action("Index", "Home")"
 class="btn btn-primary btn-lg">Tickets kiezen »
 <a href="@Url.Action("Bevestiging", "Home")"
 class="btn btn-primary btn-lg">Kassa »
 </p>
</div>
<p class="lead alert alert-info">Inhoud van uw winkelmandje:</p>
<form method="post" action "~/Home/Verwijderen">
 <table class="table">
 <thead>
 <tr>
 <th>Datum</th>
 <th>Titel</th>
 <th>Uitvoerders</th>
 <th>Prijs</th>
 <th>Plaatsen</th>
 <th></th>
 </tr>
 </thead>
 <tbody>
 @foreach (var item in Model)
 {
 <tr>
 <td class="col-md-2">@Html.DisplayFor(m => item.Datum)</td>
 <td class="col-md-3">@item.Titel</td>

```

```
<td class="col-md-3">@item.Uitvoerders</td>
<td class="col-md-1">@Html.DisplayFor(m => item.Prijs)</td>
<td class="col-md-1">@item.Plaatsen</td>
<td class="col-md-2 text-center">
 <input type="checkbox" name="@item.VoorstellingsNr" />
</td>
</tr>
}
</tbody>
<tfoot>
<tr>
 <td></td>
 <td></td>
 <td></td>
 <td></td>
 <td></td>
 <td class="text-center">
 <input type="submit" value="Selectie verwijderen" class="btn"/>
 </td>
</tr>
</tfoot>
</table>
</form>
<p class="alert alert-info">Te betalen: @(String.Format("{0:€ #,##0.00}", ViewBag.teBetalen))</p>
```

Bovenaan komen twee buttons. De eerste brengt je naar de voorstellingenpagina zodat er extra voorstellingen gereserveerd kunnen worden, de tweede naar een bevestigingspagina.

Daaronder komt de inhoud van het mandje. We gieten dit in een table. In de uiterst rechtse kolom zetten we een checkbox zodat de gebruiker een item kan aanvinken om te verwijderen.

Datum en Prijs worden via het model opgemaakt, om de checkboxen en de button uit te lijnen voeg je een Bootstrap *text-center* class toe. De kolommen geef je een breedte via de class *col-mod-x*. Het volstaat dit aan te geven in de body van de tabel. Je hoeft dit niet ook nog eens in de header of footer aan te brengen. De kolommen prijs en plaatsen worden via de css van daarnet automatisch rechts uitgelijnd (met extra padding).

Onderaan wordt het totaalbedrag van de tickets afgebeeld door gebruik te maken van de ViewBag.

Om een item uit het mandje te verwijderen gebruiken we volgende actionmethod *Verwijderen* in de HomeController :

```
[HttpPost]
public ActionResult Verwijderen()
{
 foreach (var item in Request.Form.AllKeys)
 {
 if (Session[item] != null) Session.Remove(item);
 }
 return RedirectToAction("Mandje", "Home");
}
```

Van alle checkboxen die door de gebruiker aangekruist werden komt de waarde van het name-attribuut in Request.Form.Allkeys te staan. Als naam gebruiken we het voorstellingsnummer. Dit nummer gebruiken we als key om het aantal tickets voor een voorstelling in de Session te bewaren.

We gebruiken het nummer hier om de bijhorende entry in de Session te verwijderen. Daarna gebeurt er een redirect naar het aangepaste overzicht van het winkelmandje.

Nu de nodige informatie op Sessionscope is bewaard kunnen we nog twee verbeteringen aanbrengen die we tot nu toe hebben overgeslagen. Het eerste is het al dan niet tonen van een hyperlink naar het mandjeoverzicht, het tweede is de defaultwaarde van het gewenste aantal tickets op de reservatiepagina.

Als de gebruiker nu tickets in het mandje legt en vervolgens teruggaat naar de voorstellingenpagina dan is er geen hyperlink naar het mandje. Deze hyperlink moet toegevoegd worden maar enkel wanneer er tickets op Sessionscope zijn bewaard.

Pas de actionmethod Index() als volgt aan :

```
public ActionResult Index(int? id) {
 ...
 if (Session.Keys.Count != 0)
 ViewBag.mandjeTonen = true;
 else
 ViewBag.mandjeTonen = false;
 return View(voorstellingenInfo);
}
```

Als er waarden bewaard werden in de Session dan zetten we een variabele *mandjeTonen* in de ViewBag op *true*. Zoniet dan is deze waarde *false*. In de bijhorende view *Index.cshtml* testen we de waarde van dit ViewBag-item :

```
...
<div class="jumbotron">
 <h1>Het Cultuurhuis </h1>
 <p>
 @if (ViewBag.mandjeTonen)
 {
 <a href="@Url.Action("Mandje", "Home")"
 class="btn btn-primary btn-lg">Winkelmandje »
 <a href="@Url.Action("Bevestiging", "Home")"
 class="btn btn-primary btn-lg">Kassa »
 }
 </p>
</div>
...
```

Op de reservatiepagina kunnen we invullen hoeveel tickets we voor een voorstelling willen reserveren. Indien de gebruiker nog geen tickets voor deze voorstelling in het mandje heeft gelegd dan is de standaardwaarde 1. Heeft de gebruiker wél al tickets in het mandje liggen voor deze voorstelling dan is de standaardwaarde gelijk aan het reeds eerder ingevulde aantal.

Hiervoor passen we de view *Reserveren.cshtml* aan :

```
...
<div class="form-group">
 <label class="col-md-2 control-label">Plaatsen</label>
 @{
 var aantal = 1;
 if (Session[Model.VoorstellingsNr.ToString()] != null)
```

```
{
 aantal = Convert.ToInt16(Session[Model.VoorstellingsNr.ToString()]);
}
}
<div class="col-md-10">
 <input value="@aantal" required="required" min="1" type="number"
 max="@Model.VrijePlaatsen" name="aantalPlaatsen"
 class="form-control" />
</div>
</div>
...
```

Deze keer is de defaultvalue van de inputbox niet zomaar gelijk aan 1 maar aan een aantal dat we uit de Session halen. Meer bepaald het aantal dat hoort bij de gekozen voorstelling.

## 21.2.6 Bevestiging van de reservaties : klantgegevens opzoeken

Op de bevestigingspagina kan de gebruiker inloggen met zijn gebruikersnaam en wachtwoord. We schrijven in de CultuurService een method die een klant opzoekt a.d.h.v. een naam en een wachtwoord :

```
public Klant GetKlant(string naam, string paswoord)
{
 using (var db = new CultuurHuisMVCEntities())
 {
 return (from klant in db.Klanten
 where klant.GebruikersNaam == naam &&
 klant.Paswoord == paswoord
 select klant).FirstOrDefault();
 }
}
```

We starten met een heel eenvoudige action method in de HomeController :

```
public ActionResult Bevestiging()
{
 return View();
}
```

De bijhorende View :

```
@{
 ViewBag.Title = "Bevestiging";
}
<div class="jumbotron">
 <h1>Het Cultuurhuis <img src "~/Images/bevestig.png"
 alt="bevestiging reservaties" /></h1>
 <p>
 <a href="@Url.Action("Index", "Home")"
 class="btn btn-primary btn-lg">Tickets kiezen »
 <a href="@Url.Action("Mandje", "Home")"
 class="btn btn-primary btn-lg">Winkelmandje »
 </p>
</div>
<form method="post" action("~/Home/Bevestiging" class="form-horizontal">
 <h2>Stap 1: Wie ben je?</h2>

 <div class="form-group">
 <label class="col-md-2 control-label">Gebruikersnaam</label>
 <div class="col-md-10">
```

```

 <input name="naam" id="naam" type="text" class="form-control"/>
 </div>
</div>

<div class="form-group">
 <label class="col-md-2 control-label">Wachtwoord</label>
 <div class="col-md-10">
 <input name="paswoord" id="paswoord" type="password"
 class="form-control" />
 </div>
</div>

<div class="form-group">
 <div class="col-md-offset-2 col-md-10">
 <input type="submit" name="zoek" value="Zoek me op" class="btn"/>
 <input type="submit" name="nieuw" value="Ik ben nieuw" class="btn"/>
 </div>
</div>

<h2>Stap 2: Bevestigen</h2>
<p>
 <input type="submit" name="bevestig" value="Bevestigen" class="btn"
 disabled />
</p>
</form>

```

Onder de hyperlinks naar de voorstellingen- en reservatiemandjepagina komt een form met twee invulvakken en drie knoppen. De bevestigknop is voorlopig nog gdisabled. De verwerking van de form gebeurt opnieuw door de actionmethod *Bevestiging()* na het klikken op de button *zoek*, *nieuw* of *bevestig*.

We vullen de actionmethod *Bevestiging()* nu verder aan. Om te weten op welke knop er geklikt werd testen we de waarde van Request["waarde\_name\_attribuut\_van\_de\_knop"]. Als die niet null is dan werd er op de bewuste knop geklikt.

Voeg alvast het volgende toe aan de actionmethod *Bevestiging()*:

```

public ActionResult Bevestiging()
{
 //gebruiker opzoeken
 if (Request["zoek"] != null)
 {
 }

 //nieuwe gebruiker
 if (Request["nieuw"] != null)
 {
 }

 //bevestig
 if (Request["bevestig"] != null)
 {
 }

 return View();
}

```

Als de gebruiker op de zoek-knop klikt halen we de naam en het wachtwoord uit de Request en zoeken we de klant op. Vinden we die dan geven we de opgezochte gegevens door aan de view via een sessionvariabele. We kiezen hier voor een sessionvariabele omdat we de klantgegevens later nog

zullen nodig hebben om de reservaties door te voeren in de database. Wordt de klant niet gevonden dan geven we een foutbericht door. Vul onderstaande vetgedrukte code toe in Bevestiging() :

```
if (Request["zoek"] != null)
{
 var naam = Request["naam"];
 var paswoord = Request["paswoord"];

 var klant = db.GetKlant(naam, paswoord);

 if (klant != null)
 Session["klant"] = klant;
 else
 ViewBag.errorMessage = "Verkeerde gebruikersnaam of wachtwoord";
 return View();
}
```

Pas nu ook de view *Bevestiging.cshtml* aan. Voeg onderstaande vetgedrukte code toe :

```
...
<div class="jumbotron">
...
</div>
@if (Session["klant"] != null)
{
 var klant = (MVC_Cultuur.Models.Klant)Session["klant"];
 ViewBag.klant = klant.Voornaam + " " + klant.Familienaam +
 " " + klant.Straat + " " + klant.HuisNr +
 " " + klant.Postcode + " " + klant.Gemeente;
}
<form method="post" action("~/Home/Bevestiging" class="form-horizontal">
...
<div class="form-group">
 <div class="col-md-offset-2 col-md-10">
 <input type="submit" name="zoek" value="Zoek me op" class="btn"
 @if (ViewBag.klant != null) { <text>disabled</text> } />
 <input type="submit" name="nieuw" value="Ik ben nieuw" class="btn"
 @if (ViewBag.klant != null) { <text>disabled</text> } />
 </div>
</div>
@if (ViewBag.klant != null) {
 <p class="alert alert-success">@ViewBag.klant</p>
}
@if (ViewBag.errorMessage != null) {
 <p class="alert alert-danger">@ViewBag.errorMessage</p>
}
<h2>Stap 2:Bevestigen</h2>
<p>
 <input type="submit" name="bevestig" value="Bevestigen" class="btn"
 @if (ViewBag.klant == null) { <text>disabled</text> } />
</p>
</form>
```

Als er klantgegevens worden doorgegeven (Session["klant"] != null) dan vullen we een ViewBagwaarde klant op met de klantgegevens. De knoppen zoek en nieuw worden dan gedisabled. Onder deze twee knoppen worden ofwel de klantgegevens afgebeeld ofwel een errorMessage. De knop *bevestig* is enkel bruikbaar als er klantgegevens zijn.

### 21.2.7 Bevestiging van de reservaties : een nieuwe klant

Wanneer de gebruiker op de bevestigingspagina op de knop 'Ik ben nieuw' klikt, dan redirecten we naar een actionmethod Nieuw(). Voeg in de method *Beverstiging()* de redirect toe :

```
//nieuwe gebruiker
if (Request["nieuw"] != null)
{
 //redirect naar nieuwe klant pagina
 return RedirectToAction("Nieuw", "Home");
}
```

In de homecontroller schrijven we volgende code :

```
[HttpGet]
public ActionResult Nieuw()
{
 var nieuwForm = new NieuweKlantForm();
 return View(nieuwForm);
}
```

We maken een object van het type *NieuweKlantForm* en sturen deze door naar de bijhorende view.

De class *NieuweKlantForm.cs* ziet er als volgt uit :

```
using System.ComponentModel.DataAnnotations;
...
public class NieuweKlantForm
{
 [Required(ErrorMessage = "Voornaam is verplicht in te vullen !")]
 public string Voornaam { get; set; }

 [Required(ErrorMessage = "Familienaam is verplicht in te vullen !")]
 public string Familienaam { get; set; }

 [Required(ErrorMessage = "Straat is verplicht in te vullen !")]
 public string Straat { get; set; }

 [Required(ErrorMessage = "Huisnr is verplicht in te vullen !")]
 public string Huisnr { get; set; }

 [Required(ErrorMessage = "Postcode is verplicht in te vullen !")]
 public string Postcode { get; set; }

 [Required(ErrorMessage = "Gemeente is verplicht in te vullen !")]
 public string Gemeente { get; set; }

 [Required(ErrorMessage = "Gebruikersnaam is verplicht in te vullen !")]
 public string Gebruikersnaam { get; set; }

 [Required(ErrorMessage = "Wachtwoord is verplicht in te vullen !")]
 [DataType(DataType.Password)]
 [Display(Name = "Wachtwoord")]
 public string Paswoord { get; set; }

 [Required(ErrorMessage = "Wachtwoord moet bevestigd worden !")]
 [DataType(DataType.Password)]
 [Compare("Paswoord", ErrorMessage = "{0} verschilt van {1}. Probeer opnieuw.")]
 [Display(Name = "Wachtwoord bevestigen")]
 public string HerhaalPaswoord { get; set; }
}
```

Via annotations maken we de invoervelden verplicht. De inhoud van het paswoordveld moet gelijk zijn aan de inhoud van het herhaalpaswoordveld. Voor beide wachtwoordvelden gebruiken we het Password-type. ‘paswoord’ is in principe geen correct Nederlands woord. Daarom krijgen de velden Paswoord en HerhaalPaswoord een afwijkende displayname.

De controle of een ingevulde gebruikersnaam al dan niet reeds bestaat voeren we uit via een custom validationattribute. Dit betekent dat we een class moeten maken die erft van ValidationAttribute en dat we daarin moeten controleren of de gebruikersnaam in de database voorkomt :

```
public class BestaatnognietAttribute : ValidationAttribute
{
 public override bool IsValid(object value)
 {
 if (value == null)
 return true;

 if (!(value is string))
 return false;
 else {
 Services.CultuurService db = new Services.CultuurService();
 return ! db.BestaatKlant((string)value);
 }
 }
}
```

Deze class sla je bijvoorbeeld op in de root van het project. Je voegt onderstaande method *BestaatKlant()* toe in de CultuurService :

```
//gebruikersnaam moet uniek zijn
public bool BestaatKlant(string gebruikersnaam)
{
 using (var db = new CultuurHuisMVCEntities())
 {
 var bestaandeKlant = (from klant in db.Klanten
 where klant.GebruikersNaam == gebruikersnaam
 select klant).FirstOrDefault();
 return bestaandeKlant != null;
 }
}
```

Aan het veld *Gebruikersnaam* in *NieuweKlantForm.cs* voeg je het attribuut toe :

```
...
[Required(ErrorMessage = "Gebruikersnaam verplicht!")]
[Bestaatnogniet(ErrorMessage =
"Een klant met deze gebruikersnaam komt al voor in de database. Kies een andere naam.")]
public string Gebruikersnaam {
 get { return this.gebruikersnaam; }
 set { this.gebruikersnaam = value; }
}
...
```

Het nieuweklant-formulier kan nu getoond worden. Dit is de view bij de actionmethod *Nieuw()* :

```
@model MVC_Cultuurhuis.Models.NieuweKlantForm
 @{
 ViewBag.Title = "Nieuw";
}
```

```

<div class="jumbotron">
 <h1>Het Cultuurhuis </h1>
 <p>
 <a href="@Url.Action("Index", "Home")"
 class="btn btn-primary btn-lg">Tickets kiezen »
 <a href="@Url.Action("Mandje", "Home")"
 class="btn btn-primary btn-lg">Winkelmandje »
 <a href="@Url.Action("Bevestiging", "Home")"
 class="btn btn-primary btn-lg">Kassa »
 </p>
</div>
<p class="lead alert alert-info">Vul jouw gegevens in</p>
@using (Html.BeginForm("Nieuw", "Home", FormMethod.Post,
 new { @class = "form-horizontal", role = "form" }))
{
 <div class="form-group">
 <div class="col-md-2 control-label">@Html.LabelFor(m => m.Voornaam)</div>
 <div class="col-md-10">
 @Html.EditorFor(m => m.Voornaam,
 new { htmlAttributes = new { @class = "form-control",
 @style = "display:inline-block" } })
 @Html.ValidationMessageFor(m => m.Voornaam, string.Empty,
 new { @style = "font-style: italic; padding-left:1em" })
 </div>
 </div>
 <div class="form-group">
 <div class="col-md-2 control-label">@Html.LabelFor(m => m.Familienaam)</div>
 <div class="col-md-10">
 @Html.EditorFor(m => m.Familienaam,
 new { htmlAttributes = new { @class = "form-control",
 @style = "display:inline-block" } })
 @Html.ValidationMessageFor(m => m.Familienaam, string.Empty,
 new { @style = "font-style: italic; padding-left:1em" })
 </div>
 </div>
 <div class="form-group">
 <div class="col-md-2 control-label">@Html.LabelFor(m => m.Straat)</div>
 <div class="col-md-10">
 @Html.EditorFor(m => m.Straat,
 new { htmlAttributes = new { @class = "form-control",
 @style = "display:inline-block" } })
 @Html.ValidationMessageFor(m => m.Straat, string.Empty,
 new { @style = "font-style: italic; padding-left:1em" })
 </div>
 </div>
 <div class="form-group">
 <div class="col-md-2 control-label">@Html.LabelFor(m => m.Huisnr)</div>
 <div class="col-md-10">
 @Html.EditorFor(m => m.Huisnr,
 new { htmlAttributes = new { @class = "form-control",
 @style = "display:inline-block" } })
 @Html.ValidationMessageFor(m => m.Huisnr, string.Empty,
 new { @style = "font-style: italic; padding-left:1em" })
 </div>
 </div>
 <div class="form-group">
 <div class="col-md-2 control-label">@Html.LabelFor(m => m.Postcode)</div>
 <div class="col-md-10">
 @Html.EditorFor(m => m.Postcode,
 new { htmlAttributes = new { @class = "form-control",
 @style = "display:inline-block" } })
 </div>
 </div>
}

```

```
 @Html.ValidationMessageFor(m => m.Postcode, string.Empty,
 new { @style = "font-style: italic; padding-left:1em" })
 </div>
</div>

<div class="form-group">
 <div class="col-md-2 control-label">@Html.LabelFor(m => m.Gemeente)</div>
 <div class="col-md-10">
 @Html.EditorFor(m => m.Gemeente,
 new { htmlAttributes = new { @class = "form-control",
 @style = "display:inline-block" } })
 @Html.ValidationMessageFor(m => m.Gemeente, string.Empty,
 new { @style = "font-style: italic; padding-left:1em" })
 </div>
</div>

<div class="form-group">
 <div class="col-md-2 control-label">@Html.LabelFor(m => m.Gebruikersnaam)</div>
 <div class="col-md-10">
 @Html.EditorFor(m => m.Gebruikersnaam,
 new { htmlAttributes = new { @class = "form-control",
 @style = "display:inline-block" } })
 @Html.ValidationMessageFor(m => m.Gebruikersnaam, string.Empty,
 new { @style = "font-style: italic; padding-left:1em" })
 </div>
</div>

<div class="form-group">
 <div class="col-md-2 control-label">@Html.LabelFor(m => m.Paswoord)</div>
 <div class="col-md-10">
 @Html.EditorFor(m => m.Paswoord,
 new { htmlAttributes = new { @class = "form-control",
 @style = "display:inline-block" } })
 @Html.ValidationMessageFor(m => m.Paswoord, string.Empty,
 new { @style = "font-style: italic; padding-left:1em" })
 </div>
</div>

<div class="form-group">
 <div class="col-md-2 control-label">@Html.LabelFor(m => m.HerhaalPaswoord)</div>
 <div class="col-md-10">
 @Html.EditorFor(m => m.HerhaalPaswoord,
 new { htmlAttributes = new { @class = "form-control",
 @style = "display:inline-block" } })
 @Html.ValidationMessageFor(m => m.HerhaalPaswoord, string.Empty,
 new { @style = "font-style: italic; padding-left:1em" })
 </div>
</div>

<div class="form-group">
 <div class="col-md-offset-2 col-md-10">
 <input name="toevoegen" type="submit" value="Registreer me"
 class="btn btn-primary" />
 </div>
</div>
}
```

Omdat er voldoende ruimte naast de invulvelden is, plaatsen we de validationmessages naast de invulkassen. Dit gebeurt door het display attribuut op inline-block te zetten. We geven ook wat extra css mee aan de validationmessage zodat deze cursief wordt weergegeven.

Om de foutief ingevulde velden een visuele aanduiding te geven, voeg je in *Site.css* onderstaande css toe :

```
/* Styles for validation helpers

.field-validation-error {
 color: #ff0000;
}
.field-validation-valid {
 display: none;
}
.input-validation-error {
 border: 1px solid #ff0000;
 background-color: #ffeeee;
}
```

Het invulformulier wordt verwerkt door een Post-method *Nieuw()* :

```
[HttpPost]
public ActionResult Nieuw(NieuweKlantForm form)
{
 if (this.ModelState.IsValid)
 {
 Klant nieuweKlant = new Klant();
 nieuweKlant.Voornaam = form.Voornaam;
 nieuweKlant.Familienaam = form.Familienaam;
 nieuweKlant.Straat = form.Straat;
 nieuweKlant.HuisNr = form.Huisnr;
 nieuweKlant.Postcode = form.Postcode;
 nieuweKlant.Gemeente = form.Gemeente;
 nieuweKlant.GebruikersNaam = form.Gebruikersnaam;
 nieuweKlant.Paswoord = form.Paswoord;

 Session["klant"] = nieuweKlant;
 db.VoegKlantToe(nieuweKlant);
 return RedirectToAction("Bevestiging", "Home");
 }
 else
 return View(form);
}
```

Als er geen validatiefouten zijn, dan maken we een nieuw klantobject dat we voorzien van de ingevulde gegevens. We bewaren de klantgegevens op sessionscope voor later en voegen de nieuwe klantgegevens toe in de database. Daartoe gebruiken we een nieuwe method *VoegKlantToe()* in de *CultuurService*. Daarna gaan we terug naar de bevestigingspagina. De ingevulde gegevens geven we door via de Session.

De method *VoegKlantToe()* :

```
public void VoegKlantToe(Klant nieuweKlant)
{
 using (var db = new CultuurHuisMVCEntities())
 {

 db.Klanten.Add(nieuweKlant);
 db.SaveChanges();
 }
}
```

Op de bevestigingspagina moeten we geen nieuwe aanpassingen aanbrengen. Net als bij een bestaande klant kunnen voor een nieuwe klant de nodige gegevens uit de sessionvariabele met naam "klant" gehaald worden en de status van de knoppen aangepast worden.

### 21.2.8 Bevestiging van de reservaties : reservaties doorvoeren

Er is nog één knop in de bevestigingspagina die we nog niet hebben behandeld en dat is de laatste, de knop Bevestig.

In de actionmethod *Bevestig()* vullen we het laatste if-statement verder aan.

```
...
//bevestig
if (Request["bevestig"] != null)
{
 //verwerking klantgegevens via Session["Klant"]
 var klant = (Klant)Session["klant"];
 Session.Remove("klant");

 List<MandjeItem> gelukteReservaties = new List<MandjeItem>();
 List<MandjeItem> mislukteReservaties = new List<MandjeItem>();

 //haal alle reservaties uit de session
 foreach (string nummer in Session)
 {
 Reservatie nieuweReservatie = new Reservatie();
 nieuweReservatie.VoorstellingsNr = int.Parse(nummer);
 nieuweReservatie.Plaatsen = Convert.ToInt16(Session[nummer]);
 nieuweReservatie.KlantNr = klant.KlantNr;

 Voorstelling voorstelling = db.GetVoorstelling(
 nieuweReservatie.VoorstellingsNr);
 if (voorstelling.VrijePlaatsen >= nieuweReservatie.Plaatsen)
 {
 //opslaan in database
 db.BewaarReservatie(nieuweReservatie);

 gelukteReservaties.Add(new MandjeItem(voorstelling.VoorstellingsNr,
 voorstelling.Titel, voorstelling.Uitvoerders,
 voorstelling.Datum, voorstelling.Prijs,
 nieuweReservatie.Plaatsen));
 }
 else
 {
 mislukteReservaties.Add(new MandjeItem(voorstelling.VoorstellingsNr,
 voorstelling.Titel, voorstelling.Uitvoerders,
 voorstelling.Datum, voorstelling.Prijs,
 nieuweReservatie.Plaatsen));
 }
 }
 Session.RemoveAll();
 Session["gelukt"] = gelukteReservaties;
 Session["mislukt"] = mislukteReservaties;
 return RedirectToAction("Overzicht", "Home");
}
...
```

We halen hier eerst de klantgegevens uit de sessionscope. Daarna mag deze sessionvariabele terug weg. We maken twee lijsten : één voor de gelukte reservaties en één voor de mislukte. De items zijn van hetzelfde type als bij het overzicht van het mandje.

Daarna overlopen we alle sessionvariabelen. De namen ervan zijn normaal gezien allemaal voorstellingsnummers, de waarden zijn het aantal te reserveren plaatsen. Voor elke sessionvariabele halen we drie gegevens op : het voorstellingsnummer (key van de sessionvariabele), het aantal plaatsen (value van de sessionvariabele) en het klantnummer. Deze gegevens stoppen we in een nieuw Reservatie-object.

We controleren of het aantal plaatsen ondertussen nog beschikbaar is – andere gebruikers zouden ondertussen met de laatste tickets kunnen zijn gaan lopen – en als dat zo is dan roepen we een method BewaarReservatie uit de CultuurService op om de reservatie effectief te bewaren in de database. De detailgegevens over de reservatie stoppen we in de lijst van gelukte reservaties. Zijn er ondertussen niet genoeg tickets meer beschikbaar dan komen de gegevens in de lijst met mislukte reservaties.

Tenslotte kunnen alle sessionvariabelen gewist worden en de beide lijsten via de session doorgegeven worden aan de overzichtspagina.

De code voor de method BewaarReservatie in *CultuurService.cs* :

```
public void BewaarReservatie(Reservatie gelukteReservatie)
{
 using (var db = new CultuurHuisMVCEntities())
 {
 var voorstelling = db.Voorstellingen.Find(
 gelukteReservatie.VoorstellingsNr);
 voorstelling.VrijePlaatsen -= gelukteReservatie.Plaatsen;
 db.Reservaties.Add(gelukteReservatie);
 db.SaveChanges();
 }
}
```

We verminderen het aantal tickets voor de gelukte reservatie en voegen daarna het reservatie-object toe aan de tabel met reservaties.

### **21.2.9 Overzicht van de reservaties**

Om het overzicht te krijgen van de gelukte en mislukte reservaties voeg je volgende actionmethod toe in de HomeController :

```
public ActionResult Overzicht()
{
 List<MandjeItem> gelukteReservaties = (List<MandjeItem>)Session["gelukt"];
 List<MandjeItem> mislukteReservaties = (List<MandjeItem>)Session["mislukt"];
 ViewBag.gelukt = gelukteReservaties;
 ViewBag.mislukt = mislukteReservaties;
 Session.Clear();
 return View();
}
```

Je haalt de twee lijsten uit Sessionscope en geeft ze door via de ViewBag. We maken de sessionscope terug leeg.

De inhoud van de bijhorende view :

```
@using MVC_Cultuurhuis.Models
@{ ViewBag.Title = "Overzicht"; }

Het Cultuurhuis </h1> <p>Tickets kiezen »</p> </div> @if ((ViewBag.gelukt as ICollection<MandjeItem>).Count() > 0) { <h2 class="alert alert-success">Gelukte reserveringen</h2> <table class="table"> <thead> <tr> <th>Datum</th> <th>Titel</th> <th>Uitvoerders</th> <th>Prijs</th> <th>Plaatsen</th> </tr> </thead> <tbody> @foreach (MandjeItem item in ViewBag.gelukt) { <tr> <td class="col-md-2">@Html.DisplayFor(m => item.Datum)</td> <td class="col-md-3">@item.Titel</td> <td class="col-md-3">@item.Uitvoerders</td> <td class="col-md-2">@Html.DisplayFor(m => item.Prijs)</td> <td class="col-md-2">@item.Plaatsen</td> </tr> } </tbody> </table> } @if ((ViewBag.mislukt as ICollection<MandjeItem>).Count() > 0) { <h2 class="alert alert-danger">Mislukte reserveringen</h2> <table class="table"> <thead> <tr> <th>Datum</th> <th>Titel</th> <th>Uitvoerders</th> <th>Prijs</th> <th>Plaatsen</th> </tr> </thead> <tbody> @foreach (MandjeItem item in ViewBag.mislukt) { <tr> <td class="col-md-2">@Html.DisplayFor(m => item.Datum)</td> <td class="col-md-3">@item.Titel</td> <td class="col-md-3">@item.Uitvoerders</td> <td class="col-md-2">@Html.DisplayFor(m => item.Prijs)</td> <td class="col-md-2">@item.Plaatsen</td> </tr> } </tbody> </table> }


```

Deze view bevat weinig nieuws omdat ze vrij gelijkaardig is aan het mandjeoverzicht. We overlopen de twee lijsten die we uit de viewbagwaarden ViewBag.gelukt en ViewBag.mislukt halen en gieten de inhoud telkens in een tabel zoals bij het mandjeoverzicht.

Als je dat nog niet zou gedaan hebben, dan kan je nu het reserveren naar hartelust uitproberen.

## 22 Appendix A : Browsers en webservers

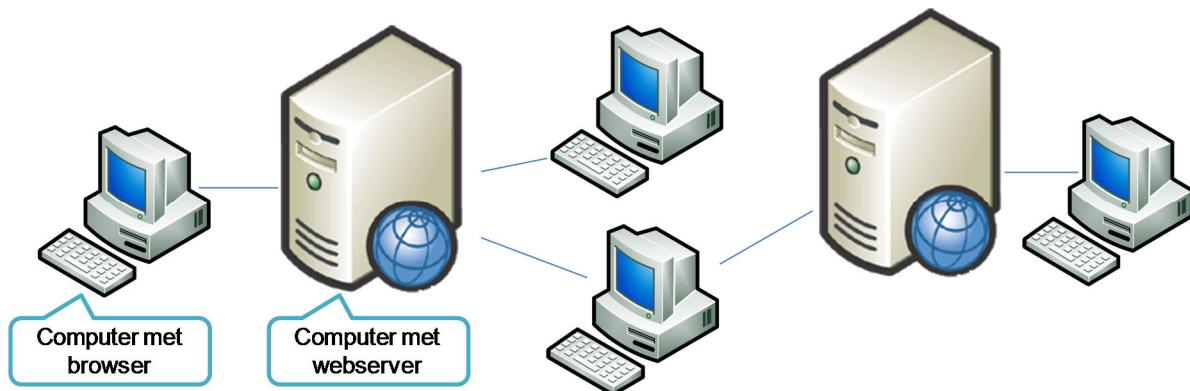
In deze appendix herhalen we nog even wat er precies achter de schermen gebeurt wanneer we naar een website surfen.

### 22.1 Algemeen

Het internet verbindt computers.

- Op sommige van deze computers is een webserver geïnstalleerd.  
Op zo'n webserver zijn één of meerdere websites geïnstalleerd.
- Op sommige van deze computers is een browser geïnstalleerd.

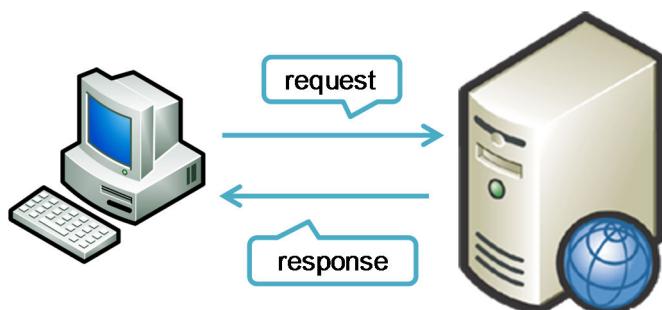
Gebruikers vragen met de browser via het internet informatie aan de websites op de webservers :



### 22.2 Requests, responses, HTTP, TCP/IP

De vraag die de browser aan de webserver stelt (in opdracht van de gebruiker die de browser bestuurt) noemen we een **Request**.

Het antwoord dat de webserver terugstuurt naar de browser noemen we de **Response**. Deze response kan HTML bevatten, een afbeelding, PDF, ...



Computers sturen requests en responses over het internet met HTTP: HyperText Transfer Protocol. Je leert in deze cursus HTTP nader kennen.

Het HTTP protocol gebruikt zelf TCP/IP:

Transmission Control Protocol / Internet Protocol.

- Computers versturen data over een netwerk met het TCP protocol.  
Als de te versturen data groot is, stuurt TCP die data in kleinere stukken.
- TCP gebruikt op zijn beurt IP.  
IP stuurt één (kleiner) stuk data over het netwerk.

Op één computer kunnen meerdere programma's het TCP/IP protocol gebruiken. Elk programma krijgt bij TCP/IP een uniek identificatiegetal: het port number.

- Webservers gebruiken standaard port number 80.
- FTP (File Transfer Protocol) servers gebruiken standaard port number 21.
- Mail servers gebruiken standaard port number 25.

## 22.3 URL

Het internet bevat veel websites. Een website bevat pagina's.

Een browser stuurt een request naar één bepaalde pagina van één website. Iedere pagina heeft een identificatie, om pagina's van elkaar te onderscheiden. Deze identificatie is de URL (Uniform Resource Locator). Wanneer de browser een request stuurt, stuurt hij die request naar de URL die de pagina identificeert.

Een URL heeft onderdelen. De belangrijkste zijn protocol, domeinnaam, padnaam.

De URL `http://pizzaluigi.be/pizzas` heeft volgende onderdelen:

- protocol      http
- domeinnaam   pizzaluigi.be
- padnaam       pizzas



Opmerkingen:

- Als je in een browser een URL zonder protocol tikt (`pizzaluigi.be/pizzas`), voegt de browser zelf het protocol toe bij het versturen van de request.
- Iedere website heeft op het internet een unieke domeinnaam (`pizzaluigi.be`). Je koopt een domeinnaam bij een DNS agent. Dit is een firma die gemachtigd is domeinnamen uit te delen.
- Een padnaam kan bestaan uit meerdere onderdelen, gescheiden door /.  
Voorbeeld: `pizzaluigi.be/dranken/alcoholisch`
- Je kunt een request doen naar een URL en geen padnaam meegeven (`pizzaluigi.be`). Je ziet dan de 'welkompagina' van die website.
- Als een webserver niet het standaard TCP/IP port number (80) gebruikt, vermeld je bij de URL ook het port number, na de domeinnaam.  
Als de webserver waarop de website van pizzaluigi draait, het port number 8080 gebruikt, doe je een request naar `pizzaluigi.be:8080/pizzas`

## 22.4 Statische pagina's – dynamische pagina's

### 22.4.1 Statische pagina

#### 22.4.1.1 *Algemeen*

Een statische pagina is een bestand op de webserver. Als de browser een request stuurt naar de webserver om de statische pagina te verkrijgen, leest de webserver de inhoud van dit bestand en stuurt die inhoud als response naar de browser.

Bestanden met de extensie HTML zijn bijvoorbeeld statische pagina's.

#### 22.4.1.2 *Voorbeeld*

Een website [pizzaluigi.be](http://pizzaluigi.be) bevat bijvoorbeeld een statische pagina `welkom.html`:

```
<!doctype html>
<html lang="nl">
 <head>
 <meta charset="UTF-8">
 <title>Pizza Luigi - welkom</title>
 </head>
 <body>
 <h1>Welkom bij Pizza Luigi</h1>
 </body>
</html>
```

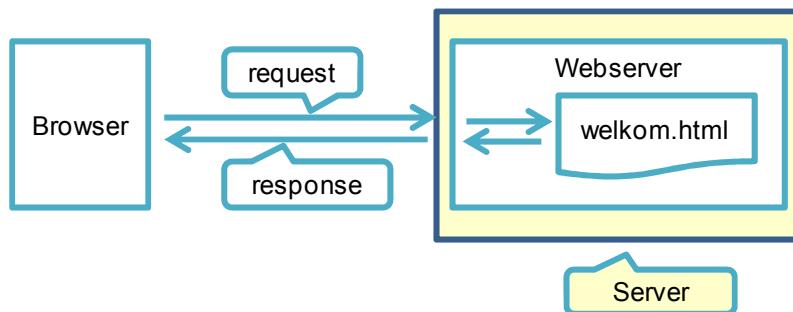
Als je een request doet naar [pizzaluigi.be/welkom.html](http://pizzaluigi.be/welkom.html), leest de webserver de inhoud van dit HTML bestand en stuurt die inhoud als response naar de browser.

De browser toont de pagina aan de gebruiker:

**Welkom bij Pizza Luigi**

#### 22.4.1.3 *Overzicht van een request naar een statische pagina*

In onderstaande afbeelding zie je het proces nog eens afgebeeld. De browser stuurt de request naar de webserver die op de server is geïnstalleerd. Deze webserver levert de pagina `welkom.html` af aan de browser via de response.



Een statische pagina is beperkt in mogelijkheden. Gezien een statische pagina geen code bevat, kan je in een statische pagina bijvoorbeeld geen producten uit een database lezen en informatie over deze producten opnemen in de response.

## 22.4.2 Een dynamische pagina

### 22.4.2.1 *Algemeen*

Een dynamische pagina is een stukje programmacode op de webserver.

Als je een request stuurt naar een dynamische pagina, roept de webserver de code op. De code wordt door een stukje software op de webserver uitgevoerd. Het resultaat hiervan is HTML die vervolgens als response naar de browser wordt gestuurd.

Je kunt in de code bijvoorbeeld producten uit een database lezen en informatie over die producten opnemen in de response.

### 22.4.2.2 *Voorbeeld*

Een website voert code uit bij een request naar [pizzaluigi.be/producten](http://pizzaluigi.be/producten)

- De code leest alle producten uit de database.
- De code stuurt eerst onderstaande html naar de response:

```
<!doctype html>
<html lang="nl">
<head>
<meta charset="UTF-8">
<title>Pizza Luigi - producten</title>
</head>
<body>
<h1>Producten</h1>

```

- De code stuurt per product volgende html naar de response:

```

De productnaam, het teken € en de productprijs

```

- De code stuurt naar de response:

```

</body>
</html>
```

Als de database volgende producten bevat:

Naam	Prijs
Quattro stagioni	3
Margherita	4

stuurt de code een response met HTML naar de browser:

## Producten

- Quattro stagioni € 3
- Margherita € 4

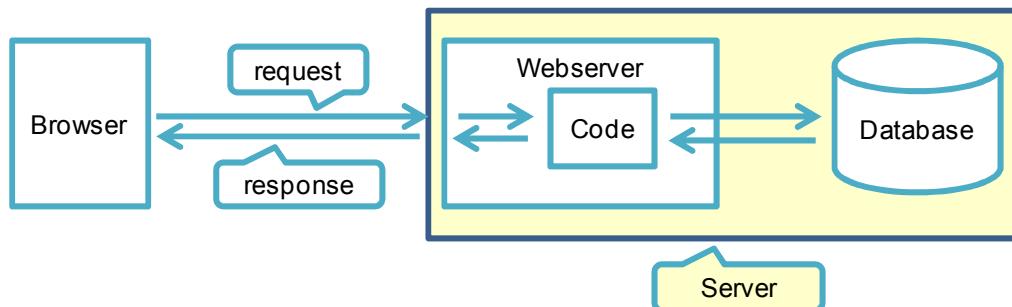
Als Luigi een nieuw product toevoegt aan de database, toont de browser bij een volgende request ook dit nieuwe product, zonder dat de code gewijzigd werd :

## Producten

- Quattro stagioni € 3
- Margherita € 4
- Calabrese € 5

#### 22.4.2.3 Overzicht van een request naar een dynamische pagina

We geven het proces nog even schematisch weer :



Dynamische pagina's zijn dus krachtiger dan statische pagina's.

Je kunt de code die hoort bij een dynamische pagina schrijven in Java, PHP, C#, ... Je leert in deze cursus deze code schrijven in C# volgens het ASP.NET MVC pattern.

#### 22.4.2.4 De IIS-webserver

De webserver van Microsoft (IIS) ondersteunt dynamische pagina's geschreven in C#. De ontwikkelomgeving Visual Studio bevat zelf een ingebouwde webserver. Dit laat toe om gedurende het ontwikkelproces snel een webapplicatie uit te proberen zonder dat de volledige applicatie op een IIS-server moet gepubliceerd worden.

Je gebruikt in deze cursus telkens de ingebouwde webserver in Visual Studio.

### 22.5 Onderdelen van een request

Een request bevat drie onderdelen:

- HTTP method (verplicht onderdeel)
- Headers (optioneel onderdeel)
- Body (optioneel onderdeel)

#### 22.5.1 HTTP method

De HTTP method geeft het soort request aan. Het HTTP protocol definieert dat de HTTP method één van volgende woorden is: GET, POST, PUT, DELETE, HEAD, TRACE, OPTIONS, CONNECT, PATCH.

HTML gebruikt enkel GET en POST.

- GET  
Je gebruikt GET bij een request waarmee je data vraagt.  
Voorbeeld: je doet een request met GET naar `pizzaluigi.be/producten` om producten te vragen. De response bevat data over de producten.
- POST  
Je gebruikt POST voor alle andere soorten requests.  
Voorbeeld: Luigi doet een request met POST naar `pizzaluigi.be/producten/toevoegen` om een product toe te voegen.  
Je gebruikt POST ook bij deze soorten requests:
  - Een request om data te wijzigen.
  - Een request om data te verwijderen.
  - Een request om in te loggen.
  - Een request om een mail te versturen.



**Opmerking:**  
In het vervolg van de cursus zullen we...

- ...een request met de HTTP method GET een 'GET request' noemen
- ...een request met de HTTP method POST een 'POST request' noemen

### 22.5.2 Headers

Een request kan headers bevatten. Headers bevatten informatie over de browser.

Iedere header heeft een naam en een waarde.

Voorbeelden van headers:

- De header met de naam *user-agent* bevat het type browser. Bij de Firefox browser is dit: *Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.2.13) Gecko/20101203 Firefox/3.6.13*
- De header met de naam *accept-language* bevat de taal en het land van de gebruiker. Bij een Nederlandstalige Belgische gebruiker is dit: *nl-be*.

### 22.5.3 Body

Een GET request bevat geen body.

Een POST request kan een body bevatten.

Deze body bevat data die de website nodig heeft om de request te verwerken.

Voorbeeld: Luigi doet een POST request naar [pizzaluigi.be/producten/toevoegen](http://pizzaluigi.be/producten/toevoegen).  
De body bevat data over het toe te voegen product (bijvoorbeeld naam en prijs).

De body heeft volgende structuur:

- Ieder stukje data in de body heeft een naam en een waarde.  
Tussen de naam en de waarde staat het = teken.
- Tussen de stukjes data staat het & teken.

Als Luigi een POST request doet naar [pizzaluigi.be/producten/toevoegen](http://pizzaluigi.be/producten/toevoegen) om een Calabrese van € 5 toe te voegen, is de body `naam=Calabrese&prijs=5`

De browser bouwt de request body op aan de hand van invulvakken van een HTML form. Deze form kan er als volgt uitzien:

## Product toevoegen

Naam:	<input type="text"/>
Prijs:	<input type="text"/>
<input type="button" value="Toevoegen"/>	

Wanneer Luigi de invulvakken invult en dit formulier submit, stuurt de browser een request. De body van deze request bevat de namen en data van de invulvakken.



### Opmerking:

Een browser stuurt een HTML form met een POST request, als het attribuut *method* van de *form* tag de waarde *post* bevat:

```
<form method="post" action="pizzaluigi.be/producten/toevoegen">
```

## 22.6 Query string

Gezien een GET request geen body heeft, kan je in een GET request de body niet gebruiken om data mee te geven. Je kunt wel data meegeven via de query string.

De query string zit op het einde van de URL en bevat één of meerdere parameters.

- Iedere parameter heeft een naam en een waarde.  
Tussen de naam en de waarde staat het = teken.
- Tussen de parameters staat het & teken.

Voorbeeld:

Je doet een GET request naar `pizzaluigi.be/openingsdagen` om de openingsdagen te weten van een bepaalde maand van een bepaald jaar. Je geeft maand en jaar mee als parameters in de query string.

Je stuurt volgende request om de openingsdagen te weten van augustus 2012:

`pizzaluigi.be/openingsdagen?maand=8&jaar=2012`

De gebruiker kan de query string zelf intikken wanneer hij de URL intikt in de adresbalk van de browser. Dit is niet gebruikersvriendelijk.

Om dit op te lossen kan de gebruiker meestal waarden voor deze parameters intikken in een HTML form. Deze form kan er als volgt uitzien:

### Openingsuren

maand:  jaар:

1	2	3	4	5	6	7	8	9	10	11	12
---	---	---	---	---	---	---	---	---	----	----	----

Wanneer de gebruiker deze form submit, stuurt de browser een request.

De query string bevat de namen en data van de invoervakken.



### Opmerking:

Een browser stuurt een HTML form met een GET request, als het attribuut *method* van de *form* tag de waarde *get* bevat: `<form method="get" action="pizzaluigi.be/openingsdagen">`

## 22.7 Onderdelen van een response

Een response bevat drie onderdelen:

- Status code (verplicht onderdeel)
- Headers (optioneel onderdeel)
- Body (optioneel onderdeel)

### 22.7.1 Status code

De status code is een getal met een betekenis gedefinieerd in het HTTP protocol. De status code 200 (OK) betekent dat de request correct verwerkt is. Als de request niet correct verwerkt is, bevat de status code een ander getal.

Enkele veel gebruikte getallen en hun betekenis:

- 404 (Not Found)  
De webserver vond de URL in de request niet.  
Voorbeeld: een request naar `pizzaluigi.be/xxx`
- 401 (Unauthorized)  
Om een request te doen naar de URL moet de gebruiker ingelogd zijn.  
Voorbeeld: een niet-ingelogde gebruiker doet een request naar `pizzaluigi.be/producten/toevoegen`
- 405 (Method not allowed)  
De request bevat een verkeerde HTTP method.  
Voorbeeld: een POST request naar `pizzaluigi.be/welkom.html`  
terwijl de website op deze URL enkel GET requests verwacht.

### 22.7.2 Headers

Een response kan headers bevatten met informatie over de response. Iedere header heeft een naam en een waarde.

Voorbeeld van een response header:

- De header met de naam *content-type* bevat het type data in de body.  
Dit type data heet het MIME-type.  
Populaire MIME-types:
  - `text/html` De body bevat HTML
  - `text/css` De body bevat CSS
  - `text/javascript` De body bevat JavaScript
  - `image/png` De body bevat een afbeelding in PNG formaat
  - `image/gif` De body bevat een afbeelding in GIF formaat
  - `image/jpeg` De body bevat een afbeelding in JPEG formaat
  - `application/pdf` De body bevat PDF

De header *content-type* kan na het MIME-type ook een parameter bevatten. Deze parameter geeft extra detail over het type data in de body.

Voorbeeld: de header *content-type* met de waarde `text/html;charset=UTF-8` geeft aan dat de body HTML bevat. De tekens (letters, cijfers) van die HTML zijn uitgedrukt in een compacte Unicode codering:UTF-8.

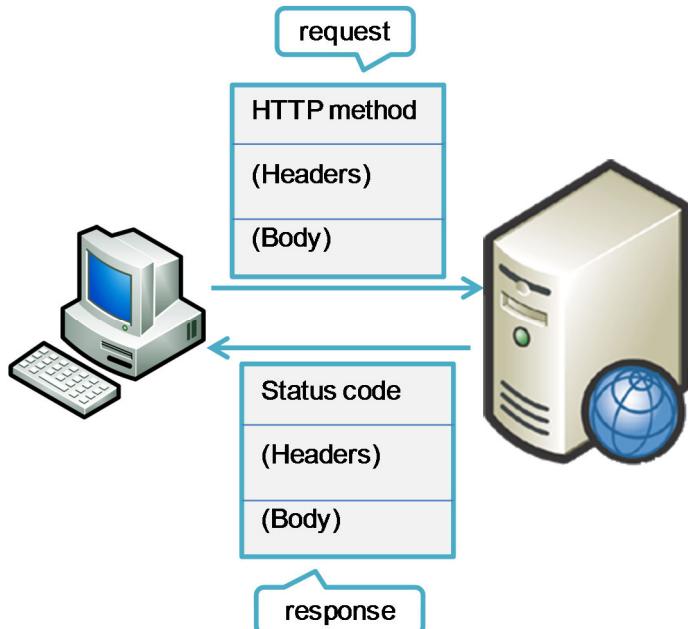
UTF-8 gebruikt zo veel mogelijk één byte om één teken voor te stellen. Voor exotische tekens (ä, ñ, ...) gebruikt UTF-8 meerdere bytes per teken.

### 22.7.3 Body

De body bevat data die door de request wordt gevraagd. Dit kan HTML zijn, een afbeelding, ...

## 22.8 Samenvatting van request en response en hun onderdelen

Hieronder nog eens in schematisch vorm : de request, response en hun onderdelen.





## 23 Appendix B : Tips bij gebruik van Windows 8

### 23.1 Gegevensopmaak

In paragraaf 9.1 “Opmaak in het cshtml-bestand” en ook verder voorzien we een getal van de nodige opmaak. Wanneer in Windows 8 de standaard landinstellingen op Nederlands (België) zijn ingesteld dan zal je geen cijfergroeperingssymbool zien maar wel iets wat op een spatie lijkt.

Wil je toch een zichtbare aanduiding voor het cijfergroeperingssymbool dan kan je dit als volgt instellen in Windows 8 :

1. Open het Control Panel (bijvoorbeeld door een rechtermuisklik linksonderaan de desktop)
2. Kies Language
3. Kies Change date, time, or number formats
4. Klik op het tabblad op de knop Additional settings...
5. Stel het digit grouping symbol in naar wens

### 23.2 Geen afbeeldingen

Wanneer je vanaf hoofdstuk 8 “Publiceren” jouw webapplicatie publiseert op IIS en het uitprobeert in een browser, dan zie je standaard geen afbeeldingen. De oorzaak hiervan is dat IIS in Windows 8 de static content uitschakelt.

Je lost dit als volgt op :

1. Open Programs and Features, ofwel rechtstreeks via een rechtermuisklik linksonderaan de desktop ofwel via het Control Panel.
2. Kies aan de linkerkant Turn Windows features on or off
3. Vink in Internet Information Services – World Wide Web Services – Common HTTP Features de optie Static Content **aan**
4. Klik op OK. De optie is actief na een reboot van Windows 8

### 23.3 Taalininstellingen

In IE10 stel je de taalininstellingen als volgt in :

1. Klik op het wietje rechts bovenaan het browservenster.
2. Kies Internet options
3. Klik op het tabblad General op de knop Languages.
4. Klik op de knop Set Language Preferences
5. Als je de gewenste taal niet ziet staan dan klik je op Add a language
6. Kies bijvoorbeeld French (français) en klik op Open.
7. Kies French (Belgium) (français (Belgique)) en klik op Add
8. Stel de gewenste volgorde in met de knoppen Move up en Move down

### 23.4 SQLServer instance instellen

Om de instance van SQL Server in te stellen doe je het volgende :

1. Start de Developer Command Prompt for VS2013.
2. Tik het commando **aspnet\_regsql -S naamvanjouwsqlserver -E -A all**

## 24 Appendix C : Oplossing oefeningen

---

### 24.1 Fibonacci-reeks (p. 31)

```
@{
 var i = 0;
 var j = 1;
 var tussen = 0;

 <h2>Rij van Fibonacci</h2>
 <p>@i, @j
 @while (tussen < 100)
 {
 tussen = i + j;
 if (tussen < 100) {
 @:, @tussen
 }
 i=j;
 j=tussen;
 }
 </p>
}
```

### 24.2 Bieren (p. 37)

De actionmethod:

```
public ActionResult Index()
{
 var bieren = new List<Bier>();
 bieren.Add(new Bier
 {
 ID = 15,
 Naam = "Felix",
 Alcohol = 7
 });
 bieren.Add(new Bier
 {
 ID = 17,
 Naam = "Roman",
 Alcohol = 7.5F
 });
 return View(bieren);
}
```

De view:

```
@model List<MVCBierenApplication.Models.Bier>
<h2>Alle bieren</h2>
@foreach (var bier in Model)
{
 @bier.ID@:. @bier.Naam
 @: (@bier.Alcohol
 @:%)

}
```

### 24.3 Logo Biertempel en adresgegevens (p.48)

\_Layout.cshtml :

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8" />
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />
 <title>@ViewBag.Title</title>
 @Styles.Render("~/Content/css")
 @Scripts.Render("~/bundles/modernizr")
</head>
<body>

 @RenderBody()
 <hr />
 <p>- De Biertempel - Hoppestraat 103 - 3360 Bierbeek - tel. 016123456 - email: info@biertempel.be -</p>
 @Scripts.Render("~/bundles/jquery")
 @Scripts.Render("~/bundles/bootstrap")
 @RenderSection("scripts", required: false)
</body>
</html>
```

### 24.4 Extra opmaak (p.57)

In een nieuwe map *Views/Shared/DisplayTemplates* voeg je een view *kleuren.cshtml* toe :

```
@model float
 @{
 if (Model < 2) {
 @Model %
 }
 else if (Model < 5) {
 @Model %
 }
 else if (Model < 8)
 {
 @Model %
 }
 else
 {
 @Model %
 }
}
```

Voeg in *Site.css* (folder *Content*) de volgende css-code toe :

```
.groen { color: green; }
.geel { color: yellow; }
.oranje { color: orange; }
.rood { color: red; }
```

Wijzig de class *Bier.cs* als volgt :

```
using System.ComponentModel.DataAnnotations;
...
public class Bier
{
 [DisplayFormat(DataFormatString = "{0:000}")]
 public int ID { get; set; }
 public String Naam { get; set; }
 [UIHint("kleuren")]
 public float Alcohol { get; set; }
}
```

In *Bier/Index.cshtml* kan je de opmaak nu gebruiken :

```
@model List<MVCBierenApplication.Models.Bier>
<h2>Alle bieren</h2>

@foreach (var bier in Model)
{
 <p>@Html.DisplayFor(m=>bier.ID) @bier.Naam @Html.DisplayFor(m=>bier.Alcohol)</p>
}
```

## 24.5 Een bier verwijderen (p.71)

We wijzigen eerst de webapplicatie zodat de bieren via een bierenservice worden opgehaald.

We voegen een class *BierenService.cs* toe :

```
using MVCBierenApplication.Models;
...
public class BierenService
{
 private static Dictionary<int, Bier> bieren =
 new Dictionary<int, Bier>();

 static BierenService()
 {
 bieren[1] = new Bier { ID = 1, Naam = "Romy pils", Alcohol = 4.5F };
 bieren[2] = new Bier { ID = 2, Naam = "Leffe blond", Alcohol = 2.5F };
 bieren[3] = new Bier { ID = 3, Naam = "Rodenbach", Alcohol = 3.5F };
 bieren[4] = new Bier { ID = 4, Naam = "Liefmans goudenband", Alcohol = 6F };
 bieren[5] = new Bier { ID = 5, Naam = "Duvel", Alcohol = 7F };
 }

 public List<Bier> FindAll() {
 return bieren.Values.ToList();
 }

 public Bier Read(int id)
 {
 return bieren[id];
 }

 public void Delete(int id)
 {
 bieren.Remove(id);
 }
}
```

We voegen in de BierController een private variabele toe voor de bierenService en wijzigen de Index-action :

```
using MVCBierenApplication.Services;
...
private BierenService bierenService = new BierenService();

public ActionResult Index()
{
 var bieren = bierenService.FindAll();
 return View(bieren);
}
```

In de Index-view die de bieren uitlijst voegen we een hyperlink naar een verwijderactie toe.

```
@model List<MVCBierenApplication.Models.Bier>
<h2>Alle bieren</h2>

@foreach (var bier in Model)
{
 var url = "Verwijderen/" + bier.ID;
 <p>@Html.DisplayFor(m=>bier.ID) @bier.Naam @Html.DisplayFor(m=>bier.Alcohol) </p>
}
```

In de BierController voegen we een aantal actions toe :

```
public ActionResult Verwijderen(int ID)
{
 var bier = bierenService.Read(ID);
 return View(bier);
}

[HttpPost]
public ActionResult Delete(int ID)
{
 var bier = bierenService.Read(ID);
 this.TempData["bier"] = bier;
 bierenService.Delete(ID);
 return Redirect("~/Bier/Verwijderd");
}

public ActionResult Verwijderd()
{
 var bier = (Bier)this.TempData["bier"];
 return View(bier);
}
```

De views :

*Verwijderen.cshtml* :

```
@model MVCBierenApplication.Models.Bier
@{
 ViewBag.Title = "Verwijderen";
}
<h2>@Model.Naam verwijderen?</h2>
<form method="post" action("~/Bier/Delete/@Model.ID">
 <input type="submit" value="Delete" />
</form>
```

*Verwijderd.cshtml :*

```
@model MVCBierenApplication.Models.Bier
 @{
 ViewBag.Title = "Verwijderd";
}
<h2>Verwijdering van @Model.Naam is doorgevoerd.</h2>
<h3>Terug naar de lijst</h3>
```

## 24.6 Hyperlinks (p. 75)

De index-pagina :

```
@model List<MVCBierenApplication.Models.Bier>
<h2>Alle bieren</h2>

@foreach (var bier in Model)
{
 var url = Url.Action("Verwijderen", "Bieren", new {ID=bier.ID});
 <p>@Html.DisplayFor(m=>bier.ID) @bier.Naam @Html.DisplayFor(m=>bier.Alcohol)
 </p>
}
```

De delete-action in de controller :

```
[HttpPost]
public ActionResult Delete(int id)
{
 var bier = bierenService.Read(id);
 this.TempData["bier"] = bier;
 bierenService.Delete(id);
 return RedirectToAction("Verwijderd");
}
```

## 24.7 Nieuw bier toevoegen en layout verzorgen (p. 102)

In de bierenService voeg je volgende method toe :

```
public void Add(Bier b)
{
 b.ID = bieren.Keys.Max() + 1;
 bieren.Add(b.ID, b);
}
```

In Index.cshtml voeg je onderaan een hyperlink toe naar een nog aan te maken action :

```
<p>@Html.ActionLink("Bier toevoegen", "Toevoegen")</p>
```

Voeg in BierController.cs onderstaande actionmethod toe waarmee het toevoegformulier wordt getoond :

```
[HttpGet]
public ActionResult Toevoegen()
{
 var bier = new Bier();
 return View(bier);
}
```

De bijhorende view :

```
@model MVCBierenApplication.Models.Bier
 @{
 ViewBag.Title = "Toevoegen";
}
<h2>Bier toevoegen</h2>
@using (Html.BeginForm("Toevoegen"))
{
 @Html.LabelFor(m => m.Naam)
 @Html.EditorFor(m => m.Naam)
 @Html.LabelFor(m => m.Alcohol)
 @Html.EditorFor(m => m.Alcohol)
 <input type="submit" value="Toevoegen" />
}
```

In de BierController voegen we dan nog een Post-method toe die het bier effectief toevoegt :

```
[HttpPost]
public ActionResult Toevoegen(Bier b)
{
 bierenService.Add(b);
 return RedirectToAction("Index");
}
```

Om de layout aan te passen voegen we de "zebra"-class toe aan de tabel waarin de bieren worden getoond. De bijhorende css-code voegen we toe aan Site.css (zie cursus) alsook de css-code die de input in een block zet. De index-view wordt nu :

```
@model List<MVCBierenApplication.Models.Bier>
<h2>Alle bieren</h2>
<table class="zebra">
<thead>
<tr>
<th>ID</th>
<th>Naam</th>
<th>Alcohol</th>
<th></th>
</tr>
</thead>
<tbody>
@foreach (var bier in Model)
{
 var url = Url.Action("Verwijderen", "Bier", new { ID = bier.ID });
 <tr>
 <td>@Html.DisplayFor(m=>bier.ID)</td>
 <td>@bier.Naam</td>
 <td>@Html.DisplayFor(m=>bier.Alcohol)</td>
 <td>

 </td>
 </tr>
}
</tbody>
</table>
@Html.ActionLink("Bier toevoegen", "Toevoegen")
```

Voor de validatie voegen we aan de class *bier.cs* een aantal data-annotations toe :

```
public class Bier
{
 [DisplayFormat(DataFormatString = "{0:000}")]
 public int ID { get; set; }
 [Required]
 [StringLength(20, ErrorMessage="Max. {1} tekens voor {0}")]
 public string Naam { get; set; }
 [UIHint("kleuren")]
 [AlcoholGrenzen(ErrorMessage="{0} heeft een ongeldige waarde")]
 public float Alcohol { get; set; }
}
```

In de BierController activeren we de validatie :

```
[HttpPost]
public ActionResult Toevoegen(Bier b)
{
 if (this.ModelState.IsValid)
 {
 bierenService.Add(b);
 return RedirectToAction("Index");
 }
 else
 return View(b);
}
```

En in de view voegen we validationmessages toe en een validationsummary :

```
<h2>Bier toevoegen</h2>
@using (Html.BeginForm("Toevoegen"))
{
 @Html.LabelFor(m => m.Naam)
 @Html.ValidationMessageFor(m => m.Naam, "*")
 @Html.EditorFor(m => m.Naam)
 @Html.LabelFor(m => m.Alcohol)
 @Html.ValidationMessageFor(m => m.Alcohol, "*")
 @Html.EditorFor(m => m.Alcohol)
 <input type="submit" value="Toevoegen" />
 @Html.ValidationSummary()
}
```

Vergeet in Site.css de validationstyles niet toe te voegen.

Om het alcoholpercentage te kunnen valideren voegen we een class *AlcoholGrenzenAttribute* toe :

```
public class AlcoholGrenzenAttribute : ValidationAttribute
{
 public override bool IsValid(object value)
 {
 if (value == null)
 return true;
 if (!(value is float))
 return false;
 var alcoholwaarde = (float)value;
 return ((alcoholwaarde < 15) && (alcoholwaarde > 0));
 }
}
```

## 24.8 Bierendata uit databank (p. 109)

Voeg in de folder *Models* een Entity Data Model toe met de naam *MVCBieren.edmx*. Je gebruikt best niet de naam *Bieren.edmx* omdat dit ook de naam is van één van de tabellen in de database.

Verwijder de oude class *Bier.cs* en breng daarna de nodige correcties aan aan de entitynamen en navigation properties in het Entity Data Model.

In de *BierenService* haal je de static variabele *bieren* weg en ook de static constructor.

Wijzig de methods in *BierenService.cs* als volgt :

```
public List<Bier> FindAll()
{
 using (var db = new MVCBierenEntities())
 {
 return db.Bieren.ToList();
 }
}

public Bier Read(int id)
{
 using (var db = new MVCBierenEntities())
 {
 return db.Bieren.Find(id);
 }
}

public void Delete(int id)
{
 using (var db = new MVCBierenEntities())
 {
 Bier bier = db.Bieren.Find(id);
 db.Bieren.Remove(bier);
 db.SaveChanges();
 }
}

public void Add(Bier b)
{
 using (var db = new MVCBierenEntities())
 {
 db.Bieren.Add(b);
 db.SaveChanges();
 }
}
```

Hier en daar moeten we nog enkele aanpassingen aanbrengen in views. In *Views/Bier/Index.cshtml* staat enkele keren *bier.ID* i.p.v. *bier.BierNr*. Bijvoorbeeld in de url naar de verwijderview een id die op *bier.ID* wordt gezet. Dit wordt *bier.BierNr* :

```
var url = Url.Action("Verwijderen", "Bier", new { ID = bier.BierNr });
```

Ook in *Verwijderen.cshtml* vervangen we ID door Biernr :

```
<form method="post" action="~/Bier/Delete/@Model.BierNr">
```

Het toevoegen van een bier zal nu niet meer lukken omdat er geen brouwernr en ook geen soortnr wordt opgegeven. Het toevoegen volledig uitwerken met keuzelijsten voor brouwer en soort, zou

ons te verleiden. Je kan je beperken tot het toevoegen van twee extra textboxen in de view *Toevoegen.cshtml*.

Door een edmx-bestand te maken en de oorspronkelijke class Bier.cs te verwijderen is de opmaak van het biernr (DisplayFormat) en het alcoholpercentage (UIHint) verdwenen. We voegen dit opnieuw toe via het principe van de buddy classes.

Voeg in de folder *Models* een class *BierProperties* toe :

```
public class BierProperties
{
 [DisplayFormat(DataFormatString = "{0:000}")]
 public int BierNr { get; set; }
 [Required]
 [StringLength(20, ErrorMessage = "Max. {1} tekens voor {0}")]
 public string Naam { get; set; }
 [UIHint("kleuren")]
 [AlcoholGrenzen(ErrorMessage = "{0} heeft een ongeldige waarde")]
 public float Alcohol { get; set; }
}
```

We linken deze class aan de class Bier.cs via een extra partial class BierUitbreiding.cs :

```
[MetadataType(typeof(BierProperties))]
public partial class Bier
{}
```

Als we de webapplicatie nu uitproberen krijgen we een fout omdat het alcoholpercentage van sommige bieren niet is ingevuld. We moeten dit opvangen in de view *Index.cshtml* :

```
...
@if (bier.Alcohol != null)
{
 @Html.DisplayFor(m=>bier.Alcohol)
}
else
{
 Niet gekend
}
...
```

## 24.9 Beveiligde bierenapplicatie (p. 167)

Maak een nieuwe MVC Internet Application en voeg de functionaliteiten uit de vorige oefening toe. Voeg een rol Administrators toe en zorg ervoor dat één van de users tot deze rol behoort zodat je kan uittesten. Aan de actionmethods die betrekking hebben tot het toevoegen en verwijderen van een bier voeg je de annotation `[Authorize(Roles = "Administrators")]` toe.

## COLOFON

Domeinexpertisemanager	Jean Smits
Moduleverantwoordelijke	Steven Lucas
Auteurs	Steven Lucas
Versie	3/06/2015

### Omschrijving module-inhoud

Abstract	Doelgroep	.NET ontwikkelaar met C#
	Aanpak	Begeleide zelfstudie
	Doelstelling	Webapplicaties leren maken met ASP.NET MVC
Trefwoorden	ASP.NET MVC C#	
Bronnen/meer info	<a href="http://www.asp.net/mvc">http://www.asp.net/mvc</a> <a href="http://www.microsoftvirtualacademy.com/">http://www.microsoftvirtualacademy.com/</a>	