



samen sterk voor werk

Knockout.

Deze cursus is eigendom van de VDAB Competentiecentra ©

Inhoudsopgave

1	INLEIDING	1
1.1	Doelstelling	1
1.2	Vereiste voorkennis	1
1.3	Nodige software	1
2	EEN SPA (SINGLE PAGE APPLICATION)	2
2.1	Algemeen	2
2.2	Architectuur van een SPA	2
2.3	Stateless	3
2.4	SPA Javascript libraries	3
2.5	De voorbeeldapplicatie in deze cursus	3
3	MVVM (MODEL – VIEW - VIEWMODEL)	4
3.1	Algemeen	4
4	DECLARATIVE BINDINGS	7
4.1	Binding van de text	7
4.2	Binding van het visible attribuut	8
4.3	Binding van het enable attribuut	8
4.4	Binding van het click event	8
4.5	Binding van een CSS class	9
4.6	Binding van andere attributen	9
5	OBSERVABLE	10
5.1	Algemeen	10
6	BINDING VAN TEXTBOXEN, CHECKBOXEN, ...	12
6.1	Textboxen	12
6.2	Checkboxen	12
6.3	Comboboxen	13
6.3.1	Een combobox afbeelden	13
6.3.2	Een property in het ViewModel invullen met de gekozen waarde uit een combobox	13
6.4	Een combobox gebaseerd op een array van objecten	14

6.5	Radiobuttons	15
7	ARRAYS IN HET VIEWMODEL	16
7.1	Algemeen	16
7.2	Een array van eenvoudige waarden	16
7.3	Een array van objecten	16
7.4	De gebruiker selecteert een array element	17
7.5	Nested foreach bindings	18
7.6	Observables in de array elementen	19
8	OBSERVABLE ARRAY	21
8.1	Algemeen	21
8.2	Functionaliteit van een observable array	21
8.3	Voorbeeld	21
9	EEN REFRESH VAN DE PAGINA	24
9.1	Algemeen	24
9.2	Een bevestiging vragen bij een refresh	24
9.3	De data onthouden in local storage	24
9.4	De data opslaan zodra de gebruiker een wijziging doet	26
10	DATA OPHALEN VAN DE SERVER APPLICATIE	27
10.1	Algemeen	27
11	TEMPLATES	30
11.1	Algemeen	30
12	BACK, FORWARD EN BOOKMARKS	33
12.1	Algemeen	33
12.2	URL fragment	33
12.3	Sammy	33
12.4	Een URL fragment met een parameter	34
13	KNOCKOUT VALIDATION	37
13.1	Algemeen	37

13.2	Plaats van de foutmeldingen	38
14	COLOFON	39

1 INLEIDING

1.1 Doelstelling

Je leert in deze cursus werken met Knockout:
een JavaScript library waarmee je SPA's (single page applications) maakt.

1.2 Vereiste voorkennis

- JavaScript
- Een eenvoudige website kunnen maken met PHP, java of .net

1.3 Nodige software

- Een IDE.
- Een browser.

2 EEN SPA (SINGLE PAGE APPLICATION)

2.1 Algemeen

Een single page application is een website die bestaat uit één pagina.

Terwijl de gebruiker de website bezoekt wordt de pagina nooit helemaal herladen. Onderdelen van die ene pagina krijgen wel een andere inhoud.

Bekende voorbeelden zijn gmail.com en outlook.com.

Een SPA heeft volgende voordelen, vergeleken met een klassieke website

- Een SPA hertekent niet de volledige pagina, maar enkel een onderdeel van de pagina. Dit geeft een stabiel beeld, zonder 'flikker'.
- Bij veel handelingen van de gebruiker wordt enkel JavaScript code in de browser uitgevoerd en geen server sided code.
Dit geeft bij die handelingen een snelle respons aan de gebruiker.

Het nadeel van een SPA is dat hij moeilijk te doorzoeken is door een zoekrobot.

2.2 Architectuur van een SPA

Een SPA bestaat in feite uit twee applicaties.

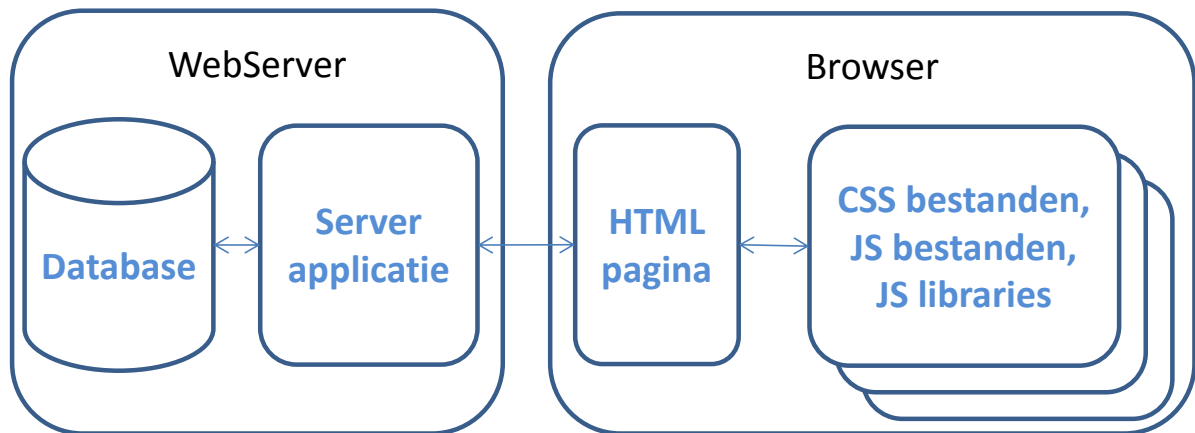
- Een client applicatie in de browser. Deze applicatie bestaat uit
 - een HTML pagina
 - één of meerdere CSS bestanden
 - één of meerdere JavaScript (JS) bestanden
 - JavaScript libraries
- Een server applicatie in de webserver. Deze applicatie doet het volgende:
 - De HTML pagina, CSS bestanden, JS bestanden en JS libraries, waaruit client applicatie bestaat, aanbieden aan de browser.
 - Data in JSON formaat uitwisselen met de client applicatie via het REST protocol. Deze uitwisseling kan zijn
 - De client heeft data nodig van de server.
De client doet een request met de method GET en krijgt een response met de gevraagde data in JSON formaat.
 - De client wil data toevoegen op de server.
De client doet een request met de method POST en geeft in de request body de toe te voegen data mee.
 - De client wil data wijzigen op de server.
De client doet een request met de method PUT en geeft in de request body de te wijzigen data mee.
 - De client wil data verwijderen op de server.
De client doet een request met de method DELETE.

Afhankelijk van de gebruikte server programmeertaal kan de server applicatie volgende vormen aannemen:

- Java
De server applicatie kan een Spring MVC website zijn.
Spring controller beans doen de datauitwisseling via REST.
- .net
De server applicatie kan een ASP.NET MVC website zijn.
WEB API controllers doen de datauitwisseling via REST.

- PHP
De server applicatie is een verzameling PHP pagina's.
Deze doen de datauitwisseling via REST.

Een overzicht:



2.3 Stateless

Een SPA is stateless. Dit betekent dat je geen data over de gebruiker onthoudt in session variabelen in het RAM geheugen van de webserver.

Je onthoudt in een SPA data over de gebruiker (zoals zijn winkelmandje) in het RAM geheugen van de browser, in JavaScript variabelen.

2.4 SPA Javascript libraries

Er bestaan verschillende JavaScript libraries om een SPA te maken

- Knockout
- Backbone
- Angular
- Ember

Je leert in deze cursus werken met Knockout.

De website knockoutjs.com bevat de Knockout library en documentatie.



Opmerking

Je kan in een applicatie waarin je Knockout gebruikt ook JQuery gebruiken.

2.5 De voorbeeldapplicatie in deze cursus

Opdat deze cursus kan gevolgd worden door Java cursisten, .net cursisten en PHP cursisten, wordt in de voorbeeldapplicatie geen Java code, .net code of PHP code opgenomen.

Als er REST communicatie nodig is om data op te halen van de server, stuur je requests naar statische JSON bestanden. In de praktijk stuur je deze requests naar Java code, .net code of PHP code (die zelf deze data lezen uit de database).

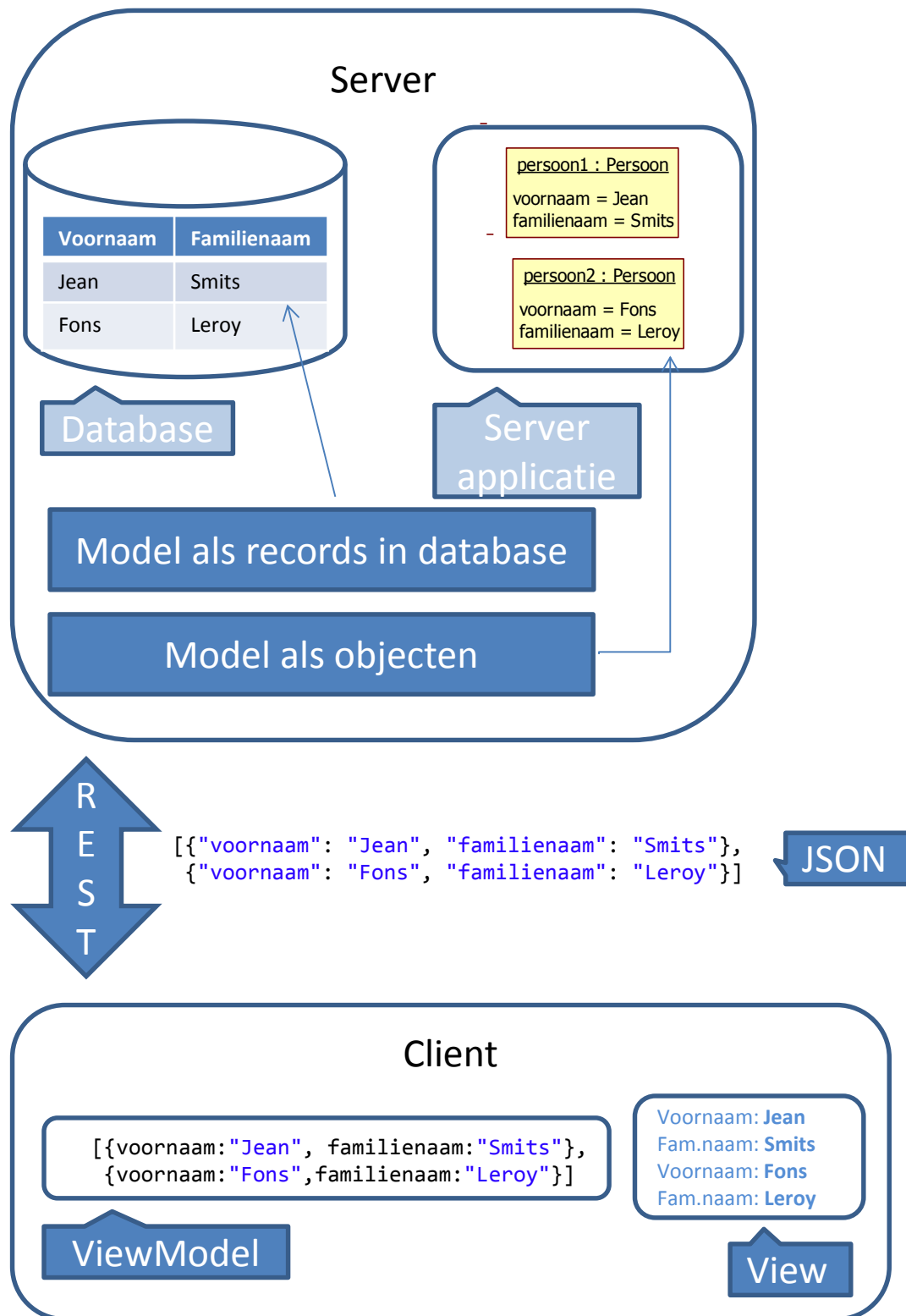
3 MVVM (MODEL – VIEW - VIEWMODEL)

3.1 Algemeen

Knockout gebruikt MVVM (Model – View – ViewModel) als architectuur.

- Model
Dit is de data zoals ze bijgehouden wordt in de webserver.
Deze data kan de vorm aannemen van records in de database.
Deze data kan ook de vorm aannemen van Java, C# of PHP objecten.
- ViewModel
Dit is dezelfde data als het Model,
maar in de vorm van JavaScript objecten in de browser.
- View
Dit is HTML die
 - data uit het ViewModel toont aan de gebruiker.
 - toelaat dat de gebruiker data uit het ViewModel wijzigt via textboxes, checkboxes, ...

Een overzicht:



Om de data te tonen in de View (de HTML),
hoef je bij Knockout geen HTML elementen te manipuleren met JavaScript.

Als je de data in het ViewModel wijzigt,
toont Knockout deze wijzigingen onmiddellijk in de View.

Als je initieel volgende data hebt in het ViewModel

```
{voornaam: "Jean", familienaam: "Smits"}
```

toont Knockout deze data in de View (de HTML)

Voornaam: Jean Familienaam: Smits
--

Als je deze data in het ViewModel wijzigt

```
{voornaam: "Jan", familienaam: "Smit"}
```

wijzigt Knockout onmiddellijk de View

Voornaam: Jan Familienaam: Smit
--

De View is met het ViewModel verbonden via declarative bindings.

Dit is een attribuut (met de naam `data-bind`) dat je toepast op HTML elementen.
Je leert dit kennen in deze cursus.

4 DECLARATIVE BINDINGS

4.1 Binding van de text

Je maakt een Java website, .net website of PHP website.

Je kopieert in deze website de bestanden die bij deze cursus horen.

Als je de website maakt als een .net website, voeg je aan Web.config volgende regels toe, als child elementen van <configuration>.

Met deze regels kan je vanuit je website een .json bestand naar de browser sturen.

```
<system.webServer>
  <staticContent>
    <mimeTypeMap fileExtension=".json" mimeType="application/json"/>
  </staticContent>
</system.webServer>
```

Je maakt het bestand index.html

```
<!doctype html>
<html lang="nl">
  <head>
    <meta charset="UTF-8">
    <title>Werknemers</title>
    <link rel="stylesheet" href="default.css">
  </head>
  <body>
    <h1>Werknemers</h1>
    <dl>
      <dt>Voornaam</dt>
      <dd data-bind="text:voornaam"></dd> (1)
      <dt>Familienaam</dt>
      <dd data-bind="text:familienaam"></dd> (2)
      <dt>Wedde</dt>
      <dd data-bind="text:wedde"></dd> (3)
    </dl>
    <script (4)
      src="http://cdnjs.cloudflare.com/ajax/libs/knockout/3.0.0/knockout-min.js">
    </script>
    <script src="index.js"></script> (5)
  </body>
</html>
```

Je maakt ook het bestand index.js

```
var viewModel = {voornaam:"Jean", familienaam:"Smits", wedde:5000}; (6)
ko.applyBindings(viewModel); (7)
```

- (1) Je doet hier declarative binding met het attribuut data-bind.
Je verbindt de text van dit dd element met de property voornaam uit het ViewModel. Knockout zal de property voornaam afbeelden in het dd element.
- (2) Je verbindt de text van dit dd element
met de property familienaam uit het ViewModel.
- (3) Je verbindt de text van dit dd element
met de property wedde uit het ViewModel.
- (4) Dit is een verwijzing naar de Knockout library.
- (5) Dit is een verwijzing naar je eigen JavaScript code.

- (6) Je maakt een JavaScript object dat zal dienen als ViewModel. Dit object bevat de properties `voornaam`, `familienaam` en `wedde`.
- (7) Je vraagt Knockout (ko) de binding naar de View uit te voeren met de variabele `viewModel`. Knockout stelt nu de tekst van de `dd` elementen in met de inhoud van de properties uit het ViewModel.

Je kan de pagina opslaan en openen in een browser om het resultaat te zien.

4.2 Binding van het visible attribuut

Je voegt een knop met de tekst `Opslag` toe.

Deze tekst is enkel zichtbaar als de `wedde` kleiner is dan 10000.

Je verbindt het `visible` attribuut van de knop met de expressie `wedde < 10000`

- Als deze expressie `true` teruggeeft is de knop zichtbaar.
- Als deze expressie `false` teruggeeft is de knop onzichtbaar.

Je voegt in `index.html` volgende regel toe onder de regel `</dl>`

```
<button data-bind="visible:wedde<10000">Opslag</button>
```

Je kan de applicatie uitproberen.

Als je de `wedde` wijzigt naar 11000 is de knop niet meer zichtbaar.

4.3 Binding van het enable attribuut

De knop wordt nu altijd zichtbaar, maar alleen bruikbaar als de `wedde < 10000`.

Je verbindt het `enable` attribuut van de knop met de expressie `wedde < 10000`

- Als deze expressie `true` teruggeeft is de knop bruikbaar.
- Als deze expressie `false` teruggeeft is de knop onbruikbaar.

Je wijzigt in `index.html` de regel met `<button ...>`

```
<button data-bind="enable:wedde<10000">Opslag</button>
```

Je kan de applicatie uitproberen.

Als je de `wedde` invult met 11000 is de knop niet meer bruikbaar.

4.4 Binding van het click event

Je voegt een knop toe. Als de gebruiker deze knop aanklikt, toon je een popup venster met de netto `wedde`.

Je verbindt daartoe het `click` event van de knop met een functie in het ViewModel. Als de gebruiker de knop aanklikt, voert Knockout deze functie uit.

Je voegt in `index.html` onder de eerste knop een tweede knop toe

```
<button data-bind="click:nettoWedde">Netto wedde</button> (1)
```

- (1) Je mag bij de `click` binding na de op te roepen functie niet `()` tikken.

Je wijzigt in `index.js` de variabele `viewModel`

```
var viewModel = {voornaam : "Jean", familienaam : "Smits", wedde : 11000,  
  nettoWedde : function() {  
    alert(this.wedde / 2);  
  }  
};
```

Je kan de applicatie uitproberen.

4.5 Binding van een CSS class

Als de wedde minstens 7000 is, toon je de wedde in het blauw.
Het bestand `default.css` bevat een CSS class `groteWedde` met deze opmaak.

Als de wedde kleiner is dan 5000, toon je de wedde in het rood.
Het bestand `default.css` bevat een CSS class `kleineWedde` met deze opmaak.

Je verbindt de `css` eigenschap van het `dd` element met een array van properties.

- De naam van iedere property is de naam van een CSS class.
- De waarde van iedere property is een boolean expressie.
 - Als deze expressie `true` is, wordt de CSS class toegepast op het `dd` element.
 - Als deze expressie `false` is, wordt de CSS class niet toegepast op het `dd` element.

Je wijzigt in `index.html` het derde `dd` element.

```
<dd data-bind="text:wedde,  
css:{groteWedde:wedde>=7000, kleineWedde:wedde<5000}"></dd>
```

 (1)

- (1) Knockout past de CSS class `groteWedde` toe als de `wedde` \geq 7000
Knockout past de CSS class `kleineWedde` toe als de `wedde` $<$ 5000

Je kan de applicatie uitproberen.

4.6 Binding van andere attributen

Je wil soms binding doen op andere HTML attributen dan degene die je tot nu gezien hebt. Voorbeeld:

je doet een binding op het `src` attribuut en het `alt` attribuut van een `img` element.

Je toont in dit `img` element het embleem van de firmawagen van de werknemer.
Als zijn wedde minstens 7000 is, is dit een Audi. Anders is dit een Golf.

Je verbindt de `attr` eigenschap van het element met een array van properties.

- De naam van iedere property is de naam van een HTML attribuut
- De waarde van iedere property is de inhoud van dit attribuut

Je voegt in `index.html` onder `</dl>` volgende regel toe:

```
<img id="embleem" data-bind="attr:{src: 'images/'+auto()+'.png', alt:auto()}">
```

Je vult het `src` attribuut met de waarde `images/`, gevolgd door het resultaat van de functie `auto` (in het ViewModel) gevolgd door de waarde `.png`.

Je vult het `alt` attribuut met het resultaat van de functie `auto` (in het ViewModel).

Je wijzigt in `index.js` de variabele `viewModel`

```
var viewModel = {voornaam : "Jean", familienaam : "Smits", wedde : 1100,  
  nettowedde : function() {  
    alert(this.wedde / 2);  
  },  
  auto: function() {  
    return this.wedde < 7000 ? "golf" : "audi";  
  }  
};
```

Je kan de applicatie uitproberen.



Taak: Welkomboodschap: zie takenbundel

5 OBSERVABLE

5.1 Algemeen

Op het moment dat je `ko.applyBindings(viewModel);` uitvoert, doet Knockout de binding: Knockout beeldt de waarden van het `viewModel` af in de View (de HTML).

Als je daarna waarden in het `ViewModel` wijzigt, toont Knockout deze gewijzigde waarden niet zomaar in de View.

Je ziet dit met de knop `Opslag`.

Je wijzigt in `index.html` de eerste knop

```
<button data-bind=" enable:wedde<10000, click:opslag">Opslag</button>
```

Je wijzigt de variabele `viewModel`

```
var viewModel = {voornaam : "Jean", familienaam : "Smits", wedde : 1100,
  nettowedde : function() {
    alert(this.wedde / 2);
  },
  auto: function() {
    return this.wedde < 7000 ? "golf" : "audi";
  },
  opslag:function() {
    this.wedde += 1000;
  }
};
```

Je voert het programma uit. Iedere keer je de knop `opslag` aanklikt, wijzigt de property `wedde` in het `ViewModel`. Je kan dit nazien met de knop `Netto wedde`. De afgebeelde `wedde` in het `dd` element wijzigt echter niet.

Je lost dit op door de property `wedde` in het `ViewModel` voor te stellen met een observable. Als je een observable wijzigt, toont Knockout de gewijzigde waarde onmiddellijk in de View.

Je maakt een observable met de functie `ko.observable()`.

Je kan tussen de haakjes een beginwaarde voor de observable meegeven:

`ko.observable(1100)`.

Een observable is een JavaScript function.

- Als je deze function oproept met lege ronde haakjes, krijg je de waarde van de observable terug: `this.wedde()` geeft bijvoorbeeld 1100
- Als je de function oproept met een waarde tussen (), wijzig je de waarde van de observable(): `this.wedde(2000)` wijzigt de `wedde` naar 2000

Je wijzigt in `index.js` de variabele `viewModel`. Bemerkt dat de functies `nettoWedde`, `auto` en `opslag` nu ronde haakjes gebruiken na `wedde`.

```
var viewModel = {voornaam : "Jean", familienaam : "Smits",
  wedde : ko.observable(1100),
  nettowedde : function() {
    alert(this.wedde() / 2);
  },
  auto: function() {
    return this.wedde() < 7000 ? "golf" : "audi";
  },
};
```

```
opslag:function() {  
    this.wedde(this.wedde()+1000);  
}  
};
```

Je gebruikt in index.html ook ronde haakjes na wedde in de laatste dd:

```
<dd data-bind="text:wedde(),  
    css:{groteWedde:wedde()>=7000, kleineWedde:wedde()<5000}"></dd>
```

Je gebruikt ook ronde haakjes na wedde in de eerste button

```
<button data-bind="enable:wedde()<10000, click:opslag">Opslag</button>
```

Je kan de applicatie uitproberen.

Als de waarde van een observable wijzigt, voert Knockout alle functies uit waarnaar je verwijst in de View, behalve de functies vermeld bij de click binding.

Je ziet dit bij de auto() binding van het img element:

als je de wedde verhoogt tot boven 7000, wijzigt de afbeelding.



Taak: BMI: zie takenbundel

6 BINDING VAN TEXTBOXEN, CHECKBOXEN, ...

6.1 Textboxen

Je kan de inhoud van een textbox binden aan een observable property in het ViewModel.

Deze binding is bidirectioneel

- Knockout toont de waarde van de property in de textbox
- Als de gebruiker de textbox wijzigt, wijzigt Knockout de waarde van de property.

Je maakt in `index.js` observables van de properties `voornaam` en `familienaam`

- ...
- `voornaam : ko.observable("Jean"),`
- `familienaam : ko.observable("Smits"),`
- ...

Je voegt in `index.html` onder `</dl>` volgende regels toe:

```
<label>Voornaam:
<input data-bind="value:voornaam"></label>
<label>Familienaam:
<input data-bind="value:familienaam"></label>
```



Opmerking

Bij `value:` mag je geen ronde haakjes vermelden na de observable.

Je kan de applicatie uitproberen.

Standaard wijzigt Knockout de property pas bij het verlaten van een textbox. Met volgende wijziging wijzigt Knockout de property na iedere toetsaanslag.

```
<label>Voornaam:
<input data-bind="value:voornaam, valueUpdate:'keyup'"></label>
<label>Familienaam:
<input data-bind="value:familienaam, valueUpdate:'keyup'"></label>
```

Je kan de applicatie uitproberen.



Taak: Galgje: zie takenbundel

6.2 Checkboxes

Je kan het attribuut `checked` van een checkbox binden aan een boolean observable in het ViewModel. Deze binding is bidirectioneel.

- Knockout toont de ViewModel property in de checkbox
 - Als de property `true` bevat, toont Knockout een vinkje in de checkbox
 - Als de property `false` bevat, toont Knockout geen vinkje
- Knockout wijzigt de ViewModel property als de gebruiker de checkbox wijzigt
 - Als de gebruiker de checkbox aanvinkt wordt property `true`.
 - Als de gebruiker de checkbox afvinkt wordt de property `false`.

Je voegt in `index.js` aan het ViewModel een boolean observable `gehuwd` toe:

```
...  
wedde : ko.observable(1100),  
gehuwd: ko.observable(true),  
...
```

Je voegt in `index.html` voor `</dl>` het volgende toe, om deze eigenschap te tonen

```
<dt>Gehuwd</dt>  
<dd data-bind="text:gehuwd() ? 'Ja' : 'Nee'"></dd>
```

Je voegt voor `` het volgende toe, om deze eigenschap op te vragen

```
<div><label>  
<input type="checkbox" data-bind="checked:gehuwd">  
Gehuwd  
</label></div>
```



Opmerking

Bij `checked`: mag je geen ronde haakjes vermelden na de observable.

Je kan de applicatie uitproberen.



Taak: Palindroom: zie takenbundel

6.3 Comboboxen

6.3.1 Een combobox afbeelden

Je kan de items van een combobox binden aan een array in het ViewModel.

Je breidt in `index.js` de variabele `viewModel` uit met een array met rijbewijstypes

```
...  
wedde : ko.observable(1100),  
rijbewijzen : [ "A3", "A", "B", "B+E", "C1", "C1+E", "C", "C+E", "D1",  
  "D1+E", "D", "D+E" ],  
...
```

Je voegt in `index.html` voor `` volgende regels toe om deze keuzes in een combobox af te beelden:

```
<label>Rijbewijs  
<select data-bind="options:rijbewijzen"></select>  
</label>
```

Je kan de applicatie uitproberen.

6.3.2 Een property in het ViewModel invullen met de gekozen waarde uit een combobox

Wanneer de gebruiker in de combobox met rijbewijzen een keuze maakt, doe je met deze keuze nog niets.

Dit zal nu veranderen: je zal met de keuze een property `rijbewijs` (van Jean Smits) invullen in het ViewModel. Je gebruikt daarvoor de `value` binding van het `select` element.

Je breidt in `index.js` de variabele `viewModel` uit met een property `rijbewijs`

```
...  
wedde : ko.observable(1100),  
rijbewijs:ko.observable("B"),  
...
```

Je wijzigt in `index.html` de tag `select`

```
<select data-bind="options:rijbewijzen, value:rijbewijs">
```

Je toont het gekozen rijbewijs (van Jean Smits).

Je voegt volgende regels toe voor `</dl>`

```
<dt>Rijbewijs</dt>  
<dd data-bind="text:rijbewijs()"></dd>
```

Je kan de applicatie uitproberen.

Als je het rijbewijs van Jean Smits initieel niet kent,
initialiseer je in `index.js` de observable `rijbewijs` als volgt

```
rijbewijs: ko.observable(),
```

Als je nu de applicatie uitvoert, zie je in de dichtgeklapte combobox `A3`
(het eerste item in de lijst). Dit is verwarrend voor de eindgebruiker.

Je kan deze verwarring vermijden met een uitbreiding op de `select` tag

```
<select data-bind="options:rijbewijzen, value:rijbewijs,  
optionsCaption:'Maak een keuze'">
```

Je kan de applicatie uitproberen. Zolang de observable `rijbewijs` niet ingevuld
wordt met de combobox, zie je in de dichtgeklapte combobox `Maak een keuze`



Taak: Sterrenbeelden: zie takenbundel

6.4 Een combobox gebaseerd op een array van objecten

De array `rijbewijzen` bevat eenvoudige strings.

Je zal nu een combobox baseren op een array waarin elk element een object is.

Je breidt in `index.js` de variabele `viewModel` uit met een array met provincies

```
...  
rijbewijzen : [ "A3", "A", "B", "B+E", "C1", "C1+E", "C", "C+E", "D1",  
                "D1+E", "D", "D+E" ],  
provincies: [{naam:"West-Vlaanderen", hoofdstad:"Brugge"},  
              {naam:"Oost-Vlaanderen", hoofdstad:"Gent"},  
              {naam:"Antwerpen", hoofdstad:"Antwerpen"},  
              {naam:"Limburg", hoofdstad:"Hasselt"},  
              {naam:"Vlaams-Brabant", hoofdstad:"Leuven"}],  
...
```

Ieder array element is een object met de properties `naam` en `hoofdstad`.

Je onthoudt de provincie die de gebruiker in de combobox zal kiezen
in een observable provincie in de variabele `viewModel`.

```
...  
rijbewijs:ko.observable(),  
provincie:ko.observable(),  
...
```

Als je een combobox baseert op de array `provincies`, moet je beslissen of je in de combobox de property naam of de property hoofdstad van iedere provincie toont. Je doet dit met de binding `optionsText`

Je voegt in `index.html` voor `` volgende regels toe:

```
<label>Provincie
<select data-bind="options:provincies, optionsText:'naam', value:provincie,
  optionsCaption:'Maak een keuze'"></select>
</label>
```

Je toont de naam en de hoofdstad van het gekozen land.

Je voegt voor `</dl>` volgende regels toe:

```
<dt>Provincie</dt>
<dd data-bind="with:provincie()"> (1)
  <span data-bind="text:naam"></span>: (2)
  <span data-bind="text:hoofdstad"></span>
</dd>
```

- (1) De `with` binding beïnvloedt het huidige element `dd` niet. Deze binding beïnvloedt wel de binding van de child elementen van het huidige element `dd`. Dit zijn de twee `span` elementen.
- (2) Je doet een binding op de property `naam`. Met de `with` binding zoekt Knockout deze property in de observable property `provincie` (van het `ViewModel`).

Je kan de applicatie uitproberen.

6.5 Radiobuttons

Je zal het geslacht opvragen met twee radiobuttons ☒ Man ☐ Vrouw

Je voegt in `index.js` in de variabele `viewModel` een observable toe waarin je het gekozen geslacht bijhoudt.

```
...
provincie:ko.observable(),
geslacht:ko.observable("M"),
...
```

Je voegt in `index.html` voor `` twee radiobuttons toe

```
<div><label>
<input type="radio" name="geslacht"
  value="M" data-bind="checked:geslacht">Man</label></div> (1)
<div><label>
<input type="radio" name="geslacht"
  value="V" data-bind="checked:geslacht">Vrouw</label></div>
```

- (1) Als de gebruiker deze radiobutton aanklikt, zal de Knockout `checked` de value van deze radiobutton ("`M`") invullen in de observable `geslacht`.

Je toont het gekozen geslacht. Je voegt voor `</dl>` volgende regels toe

```
<dt>Geslacht</dt>
<dd data-bind="text:geslacht() === 'M' ? 'Man' : 'Vrouw'"></dd>
```

Je kan de applicatie uitproberen.



Taak: Conversie: zie takenbundel

7 ARRAYS IN HET VIEWMODEL

7.1 Algemeen

Het ViewModel kan één of meerdere array's bevatten.

Je leert hier hoe je de elementen uit zo'n array afbeeldt in de View.

7.2 Een array van eenvoudige waarden

Je maakt een bestand `werelddelen.js`

```
var viewModel={
  werelddelen : ["Azië", "Afrika", "Noord-Amerika", "Zuid-Amerika", "Antartica",
    "Europa", "Oceanië"]
};
ko.applyBindings(viewModel);
```

Je maakt een bestand `werelddelen.html`, waarin je deze werelddelen toont

```
<!doctype html>
<html>
<head>
<meta charset="UTF-8">
<title>Werelddelen</title>
<link rel="stylesheet" href="default.css">
</head>
<body>
  <h1>Werelddelen</h1>
  <ul data-bind="foreach:werelddelen">                (1)
    <li data-bind="text:$data"></li>                  (2)
  </ul>
  <script
    src="http://cdnjs.cloudflare.com/ajax/libs/knockout/3.0.0/knockout-
min.js"></script>
  <script src="werelddelen.js"></script>
</body>
</html>
```

- (1) De `foreach` binding heeft geen invloed op het huidig element (`ul`). De `foreach` binding zal wel itereren over de items in de vermelde array (`werelddelen`) en per iteratie de child elementen van het huidig element in de browser tonen. In dit voorbeeld zal de binding 7 iteraties doen en per iteratie een `li` element in de browser tonen.
- (2) Je vindt bij iedere iteratie het array element waarnaar de iteratie wijst in een voorgedefinieerde variabele met de naam `$data`. Je toont deze data.

Je kan de pagina uitproberen.

7.3 Een array van objecten

De elementen van een array kunnen JavaScript objecten zijn.

Je wijzigt in `werelddelen.js` de array. De elementen van de array zijn objecten met een property naam en een property oppervlakte.

```
...
werelddelen : [ {naam : "Azië", oppervlakte : 43608000},
                 {naam : "Afrika", oppervlakte : 30335000},
                 {naam : "Noord-Amerika", oppervlakte : 25349000},
```

```
{naam : "Zuid-Amerika", oppervlakte : 21069501},  
{naam : "Antartica", oppervlakte : 13661000},  
{naam : "Europa", oppervlakte : 10498000},  
{naam : "Oceanië", oppervlakte : 8468300} ]
```

...

Je toont in werelddelen.html enkel de naam van ieder werelddeel.

Je wijzigt de tag <li ...>

```
<li data-bind="text:naam"> (1)
```

- (1) Je zit binnen een iteratie. Je doet binding naar een property van het array element waar de iteratie naar verwijst, door de naam van die property te vermelden.

Je kan de pagina uitproberen.

7.4 De gebruiker selecteert een array element

De gebruiker kan een array element selecteren. Je voorziet daartoe bij dit array element een HTML element waarbij je binding doet op het click event.

Je wijzigt in werelddelen.html de tags tot en met

```
<li>  
<span data-bind="text:naam"></span>  
<button data-bind="click:$root.detail">Detail</button> (1)  
</li>
```

- (1) Je bindt het click event aan een functie detail.
Als je voor de functie \$root schrijft, zoekt Knockout deze functie in het ViewModel.
Als je \$root weglaat, zoekt Knockout deze functie in het huidige array element (bij ons een werelddeel object met de properties naam en oppervlakte).
Als de gebruiker de button aanklikt, roept Knockout de functie (detail) op en geeft het huidige array element mee als parameter naar die functie.

Je breidt in werelddelen.js de variabele viewModel uit

```
...,  
detail:function(werelddeel) { (1)  
  alert("Oppervlakte:" + werelddeel.oppervlakte); (2)  
}  
...
```

- (1) Als de gebruiker de button aanklikt, roept Knockout deze functie op en geeft het werelddeel waarbij geklikt werd mee als parameter.
- (2) Je toont de oppervlakte van dit werelddeel.

Je kan deze pagina uitproberen.

In plaats van een button kan je om het even welk HTML element gebruiken.

Je vervangt in werelddelen.html de button bijvoorbeeld door een hyperlink

```
<a href="#" data-bind="click:$root.detail">Detail</a>
```

Je kan de pagina uitproberen.

Je kan de detailinformatie over het geselecteerde werelddeel ook in de pagina tonen, met HTML elementen.

Je breidt in `werelddelen.js` de variabele `viewModel` uit met een observable `werelddeel`. Deze observable zal het gekozen werelddeel object bevatten.

```
...,
gekozenWerelddeel:ko.observable()
...
```

Je wijzigt de binnenkant van de functie `detail`

```
viewModel.gekozenWerelddeel(werelddeel); (1)
```

- (1) Je onthoudt het werelddeel dat de gebruiker selecteerde (`werelddeel`) in de observable `gekozenWerelddeel`.
Opmerking: je kan hier niet schrijven `this.gekozenWerelddeel(werelddeel)`, want `this` verwijst in deze functie niet naar het `ViewModel`, maar naar het werelddeel dat de gebruiker selecteerde.

Je voegt in `werelddelen.html` volgende regels toe na ``

```
<dl data-bind="with:gekozenWerelddeel(),
  visible:gekozenWerelddeel() !== undefined" (1)
  style="display:none"> (2)
  <dt>Naam</dt>
  <dd data-bind="text:naam"></dd>
  <dt>Oppervlakte</dt>
  <dd data-bind="text:oppervlakte"></dd>
</dl>
```

- (1) Je geeft aan dat het `dl` element (met zijn child elementen) moet zichtbaar gemaakt worden zodra de observable `gekozenWerelddeel` ingevuld is.
- (2) Je verbergt het `dl` element als de gebruiker in de pagina binnenkomt. Anders kan de gebruiker dit element een fractie van een seconde zien.

Je kan de pagina uitproberen.



Taak: Fruit: zie takenbundel

7.5 Nested foreach bindings

Als de elementen van de array, waarover je itereert met de `foreach` binding, zelf een array bevatten, kan je binnen die `foreach` binding itereren over de geneste array met een geneste `foreach` binding.

Je wijzigt in `werelddelen.js` de array `werelddelen`.
Ieder element bevat nu een array met enkele landnamen.

```
[ {naam : "Azië", oppervlakte : 43608000,
  landen:["China","India","Maleisië"]},
  {naam : "Afrika", oppervlakte : 30335000,
  landen:["Algerije","Ghana","Oeganda"]},
  {naam : "Noord-Amerika", oppervlakte : 25349000,
  landen:["Bahama's","Canada","Verenigde Staten"]},
  {naam : "Zuid-Amerika", oppervlakte : 21069501,
  landen:["Argentinië","Brazilië","Venezuela"]},
  {naam : "Antartica", oppervlakte : 13661000,
  landen:[]},
  {naam : "Europa", oppervlakte : 10498000,
  landen:["België","Frankrijk","Duitsland"]},
  {naam : "Oceanië", oppervlakte : 8468300,
  landen:["Australië","Indonesië","Nieuw-Zeeland"]}]
```

Je voegt in `werelddelen.html` na `` volgende regels toe:

```
<ul data-bind="foreach:Landen">                                (1)
  <li data-bind="text:$data"></li>                             (2)
</ul>
```

- (1) De buitenste iteratie wijst op dit moment naar één werelddeel object. Hier itereer je over de elementen van de array `landen` in dat werelddeel.
- (2) Een element uit de array `landen` is een eenvoudige waarde, die je aanspreek met `$data`.

Je kan de pagina uitproberen.

Je kan de landen van een werelddeel ook tonen in het detail deel (dat de gebruiker ziet als hij Detail aanklikt).

Je verwijdert de ` ... ` die je hierboven maakte.

Je voegt volgende regels toe voor `</dl>`

```
<dt>Landen</dt>
<dd>
  <ul data-bind="foreach:Landen">
    <li data-bind="text:$data"></li>
  </ul>
</dd>
```


Je kan de pagina uitproberen.

7.6 Observables in de array elementen

De array elementen kunnen observables bevatten.

Je kan dan in de View textboxen, checkboxen, ... binden aan deze observables.

Je maakt een voorbeeld waarin de gebruiker de neerslag per dag kan intikken en ziet wat de gemiddelde neerslag is.

Opmerking:
niet alle browsers tonen  bij een `<input type="number" ...>`

Neerslag	
Dag	Neerslag
maandag	10 
dinsdag	
woensdag	20 
donderdag	
vrijdag	30 
zaterdag	
zondag	40 
Gemiddelde	25

Je maakt een bestand `neerslag.js`

```
var viewModel = {
  dagen : [ {naam : "maandag", neerslag:ko.observable()},
    {naam : "dinsdag",neerslag : ko.observable()},
    {naam : "woensdag", neerslag: ko.observable()},
    {naam : "donderdag", neerslag : ko.observable()},
    {naam : "vrijdag", neerslag:ko.observable()},
    {naam : "zaterdag", neerslag:ko.observable()},
    {naam : "zondag", neerslag:ko.observable()}
  ],
  (1)
```

```
gemiddelde: function() {
    var totaal = 0;
    var index;
    var neerslag;
    var aantal = 0;
    for (index = 0; index !== this.dagen.length; index++) {
        neerslag = this.dagen[index].neerslag();
        if (neerslag !== undefined && (! isNaN(neerslag)) && neerslag !== "") {
            aantal++;
            totaal += parseInt(neerslag);
        }
    }
    return aantal === 0 ? "" : totaal / aantal;
};
ko.applyBindings(viewModel);
document.querySelector(
    "tbody tr:first-child td:nth-child(2) input").focus();           (2)
```

- (1) Ieder array element bevat een dagnaam en een bijbehorende observable voor het getal met de neerslag.
Je zal in de View een textbox koppelen aan deze observable.
- (2) Je zoekt in het tbody element het 1° tr element. Je zoekt daarin het 2de td element. Je zoekt daarin het input element. Je plaatst daarin de cursor.

Je maakt een bestand neerslag.html

```
<!doctype html>
<html>
<head>
<meta charset="UTF-8">
<title>Neerslag</title>
<link rel="stylesheet" href="default.css">
</head>
<body>
<h1>Neerslag</h1>
<table>
<thead><tr><th>Dag</th><th>Neerslag</th></tr></thead>
<tbody data-bind="foreach:dagen">
<tr>
<td data-bind="text:naam"></td>
<td><input data-bind="value:neerslag" type="number"/></td>
</tr>
</tbody>
<tfoot>
<tr>
<td>Gemiddelde</td>
<td data-bind="text:gemiddelde()"></td>
</tr>
</tfoot>
</table>
<script
    src="http://cdnjs.cloudflare.com/ajax/libs/knockout/3.0.0/knockout-
min.js"></script>
<script src="neerslag.js"></script>
</body></html>
```

Je kan de pagina uitproberen.



Taak: Verkopen: zie takenbundel

8 OBSERVABLE ARRAY

8.1 Algemeen

Op het moment dat je `ko.applyBindings(viewModel);` uitvoert, doet Knockout de binding: Knockout toont de waarden van het `viewModel` in de View (de HTML).

Je hebt gezien dat deze waarden ook uit een array kunnen komen.

Als je daarna aan de array elementen toevoegt of verwijdert, toont Knockout deze nieuwe situatie niet zomaar in de View.

Om dit probleem op te lossen voorziet Knockout een observable array.

Als je aan een observable array een element toevoegt of een element verwijdert, toont Knockout deze nieuwe situatie onmiddellijk in de View.

8.2 Functionaliteit van een observable array

- Je maakt een lege observable array met de functie `observableArray`
`var legeArray = ko.observableArray();`
- Je maakt een observable array met initiële elementen, door deze elementen als een array parameter mee te geven aan de functie `observableArray`
`var daltonBrothers = ko.observableArray({"Joe", "Jack", "William"});`
- Het resultaat van de functie `observableArray` is een functie.
Als je informatie leest uit een observable array, vermeld je daarom ronde haakjes na de naam van de observable array
`var aantalElementen = daltonBrothers().length;`
`var eersteBroer = daltonBrothers()[0];`
- Je voegt een element toe aan een observable array met de functie `push`.
Je geeft dit element mee als een parameter. Bemerk dat je hier geen ronde haakjes vermeld na de naam van de observable array
`daltonBrothers.push("Averell");`
- Je verwijdert een element uit een observable array met de functie `remove`.
Je geeft een parameter mee aan `remove`. Knockout verwijdert uit de observable array alle elementen die gelijk zijn aan deze parameter.
`daltonBrothers.remove("William");`
- Je verwijdert alle elementen die aan een voorwaarde voldoen uit een observable array met een tweede versie van de functie `remove`. Je geeft als parameter een functie mee aan `remove`. Knockout roept per element deze functie op en geeft het element mee als een parameter. Als deze functie `true` teruggeeft, verwijdert Knockout het element uit de observable array.
Je verwijdert bijvoorbeeld de namen die met de letter J beginnen
`daltonBrothers.remove(function(element) {return element.charAt(0)=== "J"});`

8.3 Voorbeeld

Je maakt een applicatie waarmee de gebruiker zijn todo's kan intikken.

Je onthoudt deze todo's in een observable array.

Je maakt een bestand `todo.js`

```
var viewModel = {  
  todos : ko.observableArray(),  
  toevoegen : function() {  
    var nieuweLijn = {omschrijving : ko.observable(""),  
                      afgewerkt : ko.observable(false) };  
    this.todos.push(nieuweLijn);  
  }  
};  
ko.applyBindings(viewModel);
```

(1) (2) (3)

- (1) Je zal deze functie uitvoeren als de gebruiker een knop Toevoegen aanklikt.
- (2) Je maakt een object met twee properties
 - a. de omschrijving van de todo
 - b. het feit of de todo afgewerkt is.
- (3) Je voegt dit object toe aan de observable array.

Je maakt een bestand `todo.html`

```
<!doctype html>  
<html>  
<head>  
<meta charset="UTF-8">  
<title>Todo</title>  
<link rel="stylesheet" href="default.css">  
</head>  
<body>  
<h1>Todo</h1>  
  <table>  
    <thead>  
      <tr>  
        <th>Omschrijving</th>  
        <th>Afgewerkt</th>  
      </tr>  
    </thead>  
    <tbody data-bind="foreach:todos()">  
      <tr>  
        <td><input data-bind="value:omschrijving"></td>  
        <td><input data-bind="checked:afgewerkt" type="checkbox"></td>  
      </tr>  
    </tbody>  
  </table>  
  <tfoot>  
    <tr>  
      <td colspan="2" data-bind="text:todos().length+' todo(s)'"></td>  
    </tr>  
  </tfoot>  
</table>  
<button data-bind="click:toevoegen">Toevoegen</button>  
<script  
  src="http://cdnjs.cloudflare.com/ajax/libs/knockout/3.0.0/knockout-  
min.js"></script>  
<script src="todo.js"></script>  
</body>  
</html>
```

Je kan de applicatie uitproberen.

Nadat de gebruiker de knop Toevoegen aanklikt, is het handig dat de cursor klaar staat in het omschrijving tekstvak van de nieuwe todo.

Je voegt daartoe in `todo.js` volgende opdracht toe aan de functie `toevoegen`

```
document.querySelector("tbody tr:last-child td:first-child input").focus();
```

Je zoekt hiermee in het element `tbody` het laatste `tr` element. Je zoekt daarin het eerste `td` element. Je zoekt daarin het `input` element. Je plaatst daarin de cursor.

Je kan de applicatie uitproberen.

Je doorstreept de omschrijving van todo's die de gebruiker aanvinkt als afgewerkt.


Je voegt in `todo.html` volgende regels toe voor `</head>`

```
<style>
.afgewerkt {
text-decoration: line-through;
}
</style>
```

Je breidt het eerste `input` element uit

```
<input data-bind="value:omschrijving, css:{afgewerkt:afgewerkt}">
```

Je kan de applicatie uitproberen.

Je toont naast iedere todo de afbeelding .

Als de gebruiker deze aanklikt, verwijdert hij die todo uit het lijstje.

Je voegt na de laatste `</th>` volgende regel toe

```
<th>Verwijderen</th>
```

Je voegt voor de eerste `</tr>` volgende regel toe

```
<td></td>
```

Je wijzigt `<td colspan="2" ...>` naar `<td colspan="3" ...>`

Je breidt in `todo.js` de variabele `viewModel` uit

```
,
verwijderen:function(teVerwijderenElement) {
viewModel.todos.remove(teVerwijderenElement);
}
};
```

Je kan de applicatie uitproberen.

Je voegt naast de button nog een button `Afgewerkte todo(s) verwijderen` toe.

Je voegt in `todo.html` volgende regel toe

```
<button data-bind="click:afgewerkteVerwijderen">Afgewerkte todo(s)
verwijderen</button>
```

Je breidt in `todo.js` de variabele `viewModel` uit

```
,
afgewerkteVerwijderen:function() {
viewModel.todos.remove(function(element){return element.afgewerkt();});
}
};
```



Taak: Frituur: zie takenbundel

9 EEN REFRESH VAN DE PAGINA

9.1 Algemeen

Als de gebruiker een refresh doet van de pagina, laadt hij de pagina opnieuw op. Hierbij wordt ook alle JavaScript code terug uitgevoerd en krijgen alle variabelen terug hun initiële inhoud.

Dit kan vervelend zijn als de gebruiker reeds data in de pagina intikte en daarna “per ongeluk” de pagina refresht. Je kan dit bijvoorbeeld uittesten in `todo.html`.

Je kan dit probleem op twee manieren oplossen

- Een bevestiging vragen bij een refresh
- De data onthoud in local storage.

9.2 Een bevestiging vragen bij een refresh

Je voegt volgende code toe onder in `todo.js`

```
window.onbeforeunload = function() { (1)
    return "Ben je zeker, er kan data verloren gaan ?"; (2)
};
```

- (1) Als de gebruiker de pagina refresht of de pagina verlaat, treedt in het object `window` het event `onbeforeunload` op. Je koppelt een functie aan dit event. Als het event optreedt, voert de browser deze functie uit.
- (2) Je geeft een string terug. De browser gebruikt deze string in een popup venster dat hij aan de gebruiker toont. Dit venster ziet er bij Chrome zo uit



Enkel als de gebruiker bevestigt, laadt de browser de pagina opnieuw.

Je kan de applicatie uitproberen.

9.3 De data onthouden in local storage

De browser bewaart data die je onthoudt in local storage op de harde schijf. Ook als je de browser verlaat, blijft de data aanwezig op de harde schijf.

Je kan meerdere data naast mekaar onthouden.

Om deze data van mekaar te onderscheiden geef je iedere data een unieke key.

Voorbeeld: je onthoudt de voornaam `Jean` met de key `voornaam`

en de familienaam `Smits` met de key `familienaam`

```
window.localStorage.setItem("voornaam", "Jean");
window.localStorage.setItem("familienaam", "Smits");
```

Als je data terugleest, vermeld je de key van de terug te lezen data

```
var voornaam = window.localStorage.getItem("voornaam");  
var familienaam = window.localStorage.getItem("familienaam");
```

De data zelf moet een string zijn, niet zomaar een JavaScript object.

Als je toch JavaScript objecten wil wegschrijven, zet je het JavaScript object om naar JSON formaat, wat een string is, en je schrijft dit JSON formaat weg.

```
var persoon = {voornaam : "Jean", familienaam : "Smits"};  
var inJSONFormaat = ko.toJSON(persoon);  
window.localStorage.setItem("persoon", inJSONFormaat);
```

 (1)

- (1) De functie `toJSON` geeft een string terug.
Deze string is het JavaScript object (in de parameter) in JSON formaat.
Hier is dit de string `{ "voornaam" : "Jean", "familienaam" : "Smits" }`

Bij het teruglezen lees je de string in JSON formaat
en zet je die string terug om naar een JavaScript object.

```
var inJSONFormaat = window.localStorage.getItem("persoon");  
var persoon = JSON.parse(inJSONFormaat);
```

 (1)

- (1) Je geeft aan de functie `parse` een string met JSON formaat mee.
De functie ontleedt deze string en geeft je het bijbehorend JavaScript object terug. Hier is dit object `{voornaam : "Jean", familienaam : "Smits"}`

Je wijzigt de drie laatste regels in `todo.js`

```
window.onbeforeunload = function() {  
    opslaan();  
};
```

Je voegt een functie `opslaan` toe

```
function opslaan() {  
    window.localStorage.setItem("todos", ko.toJSON(viewModel.todos));  
}
```

 (1)

- (1) Je zet de array `todos` om naar een string in JSON formaat
en je schrijf deze string weg in local storage onder de key `todos`.

Je voegt ook nog volgende code toe onder in `todo.js`.

De browser voert deze code uit bij het laden van de pagina.

```
laden();
```

Je voegt ook nog volgende functie toe onder in `todo.js`.

```
function laden() {  
    var index;  
    var todos = window.localStorage.getItem("todos");  
    var todo;  
    var todoMetObservables;  
    if (todos !== null) {  
        todos = JSON.parse(todos);  
        for (index = 0; index != todos.length; index++) {  
            todo = todos[index];  
            todoMetObservables = {omschrijving:ko.observable(todo.omschrijving),  
                                  afgewerkt:ko.observable(todo.afgewerkt)};  
            viewModel.todos.push(todoMetObservables);  
        }  
    }  
}
```

 (1)
(2)

- (1) Als de local storage nog geen data bevat met de key `todos`, zal deze variabele gelijk zijn aan `null`.
- (2) Je zet de string met JSON formaat om in een JavaScript object.

Je kan de applicatie uitproberen. Ook als je de browser verlaat en daarna de pagina in de browser terug laadt, lees je de todo items terug uit local storage.

9.4 De data opslaan zodra de gebruiker een wijziging doet

De code slaat nu de data op bij een refresh van de pagina of bij het verlaten van de pagina. Je kan de data ook opslaan bij iedere wijziging die de gebruiker doet (in textboxes, checkbox, ...) en bij iedere verwijdering die de gebruiker doet.

Als de elektriciteit uitvalt, verliest de gebruiker op die manier geen data.

Je verwijdert in `todos.js` volgende regels:

```
window.onbeforeunload = function() {  
    opslaan();  
};
```

Je voegt op het einde van de functie toevoegen volgende opdrachten toe

```
nieuweLijn.omschrijving.subscribe(function() {  
    opslaan();  
});  
nieuweLijn.afgewerkt.subscribe(function() {  
    opslaan();  
});
```

 (1)

- (1) Je kan op een observable de functie `subscribe` oproepen. Je geeft als parameter een function mee. Iedere keer de gebruiker deze observable wijzigt, roept Knockout deze function op. In deze function roep je de function `opslaan` op, die de data opslaat.

Je voegt in de functie laden na de opdracht

```
viewModel.todos.push(todoMetObservables);
```

dezelfde opdrachten toe

```
todoMetObservables.omschrijving.subscribe(function() {  
    opslaan();  
});  
todoMetObservables.afgewerkt.subscribe(function() {  
    opslaan();  
});
```

Je voegt volgende opdracht toe achteraan in de functie verwijderen

```
opslaan();
```

Je voegt volgende opdracht toe achteraan in de functie `afgewerkteVerwijderen`

```
opslaan();
```

Je kan de applicatie uitproberen.



Taak: Verjaardagen: zie takenbundel

10 DATA OPHALEN VAN DE SERVER APPLICATIE

10.1 Algemeen

De meeste data die je nodig hebt in een SPA codeer je niet hard in je JavaScript, maar haal je op van de server applicatie.

Je doet dit door een Ajax request te sturen met de method GET naar een URL. De server applicatie verwerkt deze request, leest data uit de database en geeft deze data in JSON formaat terug als response op de request.

Het is mogelijk dat je het client gedeelte van een SPA maakt terwijl het server gedeelte nog niet geschreven is. Je kan dan ook nog geen Ajax requests versturen naar dit server gedeelte. Je stuurt deze requests dan voorlopig naar zelfgemaakte JSON bestanden met voorbeelddata.

Het bestand `genres.json` is zo'n bestand

```
[
  { "nummer" : 1, "naam" : "Rock",
    "link": { "rel" : "artists", "href" : "genres.1.artists.json" } },
  { "nummer" : 2, "naam" : "Jazz",
    "link": { "rel" : "artists", "href" : "genres.2.artists.json" } },
  { "nummer" : 7, "naam" : "Bossa Nova",
    "link": { "rel" : "artists", "href" : "genres.7.artists.json" } }
]
```

 (1)

- (1) De voorbeelddata bestaat uit drie muziekgenres. Per genre wordt het nummer en de naam vermeld. Ieder genre heeft ook een property `link`. In deze property wordt vermeld dat de artists (bij `rel`) kunnen opgehaald worden door een Ajax request te doen naar de URL vermeld bij `href`.

De URL, vermeld bij `href` van het genre Rock, verwijst naar het bestand `genres.1.artists.json`

```
["Aerosmith", "Foo Fighters", "Led Zeppelin", "Pearl Jam", "Red Hot Chilli Peppers"]
```

De URL, vermeld bij `href` van het genre Jazz, verwijst naar het bestand `genres.2.artists.json`

```
["Aaron Goldberg", "Miles Davis"]
```

De URL, vermeld bij `href` van het genre Bossa Nova, verwijst naar het bestand `genres.7.artists.json`

```
["Astrud Gilberto", "Sergio Mendes", "Caetano Veloso"]
```

Je maakt een applicatie waarin je deze data leest en toont aan de gebruiker.

Je maakt een pagina `muziek.html`

```
<!doctype html>
<html>
<head>
<meta charset="UTF-8">
<title>Muziek</title>
<link rel="stylesheet" href="default.css">
</head>
<body>
<h1>Muziek</h1>
<h2>Genres</h2>
```

```
<ul data-bind="foreach:genres">
  <li data-bind="text:naam"></li>
</ul>
<script
  src="http://cdnjs.cloudflare.com/ajax/libs/knockout/3.0.0/knockout-
min.js"></script>
<script src="muziek.js"></script>
</body>
</html>
```

Je maakt een pagina muziek.js

```
var viewModel = {
};
var xmlHttpRequest = new XMLHttpRequest(); (1)
xmlHttpRequest.open("GET", "genres.json", true); (2)
xmlHttpRequest.onreadystatechange = function() { (3)
  if (xmlHttpRequest.readyState === 4 && xmlHttpRequest.status === 200) { (4)
    viewModel.genres = JSON.parse(xmlHttpRequest.responseText); (5)
    ko.applyBindings(viewModel); (6)
  }
};
xmlHttpRequest.send(); (7)
```

- (1) Je maakt een object van het type XMLHttpRequest. Met zo'n object kan je Ajax requests versturen.
- (2) Je geeft aan dat je een request met de method GET zal versturen naar de URL genres.json
- (3) Je koppelt een functie aan het onreadystatechange event van dit object.
- (4) Als dit event optreedt én de property readyState gelijk is aan 4, is de response (op de Ajax request) aangekomen in de browser. Deze response bevat geen fouten als de status gelijk is aan 200.
- (5) Je zet de JSON in de response om naar een Java object (een array met genres). Je plaatst dit Java object in het ViewModel.
- (6) Je verbindt het ViewModel met de View.
- (7) Je verstuurt de Ajax request.

Je kan de applicatie uitproberen.

Als volgende stap maak je van de genres hyperlinks.

Wanneer de gebruiker zo'n genre aanklikt, toon je dit genre in een h2 element.

Je wijzigt in muziek.html de regel ...

```
<li><a href="#" data-bind="text:naam, click:$root.kiesGenre"></a></li>
```

Je voegt onder volgende regel toe

```
<h2 data-bind="with:genre()"><span data-bind="text:naam"></span></h2>
```

Je wijzigt in muziek.js de initialisatie van de variabele viewModel

```
var viewModel = {
  genre: ko.observable()
};
```

Je voegt na deze code volgende regels toe

```
viewModel.kiesGenre=function(gekozenGenre) {
  viewModel.genre(gekozenGenre);
};
```


Je kan de applicatie uitproberen.

Als volgende stap toon je onder dit h2 element de artiesten van het gekozen genre

Je voegt in muziek.html na </h2> volgende regels toe

```
<ul data-bind="foreach:artiesten">
  <li data-bind="text:$data"></li>
</ul>
```

Je wijzigt in muziek.js de initialisatie van de variabele viewModel

```
var viewModel = {
  genre : ko.observable(),
  artiesten: ko.observableArray()
};
```

Je breidt de functie kiesGenre uit

```
viewModel.kiesGenre = function(gekozenGenre) {
  viewModel.genre(gekozenGenre);
  xmlHttpRequest.open("GET", gekozenGenre.link.href, true);
  xmlHttpRequest.onreadystatechange = function() {
    if (xmlHttpRequest.readyState === 4 && xmlHttpRequest.status === 200) {
      var artiesten = JSON.parse(xmlHttpRequest.responseText);
      viewModel.artiesten(artiesten);
    }
  };
  xmlHttpRequest.send();
};
```

- (1) Je geeft aan dat je een Ajax request zal versturen met de method GET naar de URL in de property link.href van het gekozen genre.
- (2) Je brengt de array uit de response (op de Ajax request) over naar de observable array in het ViewModel.

Je kan de applicatie uitproberen.



Taak: Producten: zie takenbundel

11 TEMPLATES

11.1 Algemeen

Bij een SPA bestaat de volledige applicatie uit één HTML pagina.

Je zal in bepaalde delen van deze HTML pagina, afhankelijk van handelingen van de gebruiker, andere informatie tonen.

Voorbeeld : je toont initieel (in het gestreept omrande deel van de pagina) het adres van de hoofdzetel:



Als de gebruiker Foto aanklikt, toon je in hetzelfde pagina deel de foto van de hoofdzetel

Hoofdzetel

[Adres](#) [Foto](#)



Als de gebruiker Adres aanklikt, toon je terug het adres.

- De plaats in de pagina waarin je het ene veranderlijke gedeelte (adres) of het andere veranderlijke gedeelte (foto) toont, heet een placeholder.
- De veranderlijke gedeelten heten templates.

Je maakt een pagina `hoofdzetel.html`

```
<!doctype html>
<html lang="nl">
<head>
<meta charset="UTF-8">
<title>Hoofdzetel</title>
<link rel="stylesheet" href="default.css">
</head>
<body>
  <h1>Hoofdzetel</h1>
  <a href="#" data-bind="click:adres">Adres</a> (1)
  <a href="#" data-bind="click:foto">Foto</a> (2)
  <div data-bind="template:template()"></div> (3)
  <script id="adres" type="text/html"> (4)
    <h2>Adres</h2>
    <dl>
      <dt>Straat</dt>
      <dd data-bind="text:straat"></dd>
      <dt>Huisnummer</dt>
      <dd data-bind="text:huisnummer"></dd>
      <dt>Postcode</dt>
      <dd data-bind="text:postcode"></dd>
      <dt>Gemeente</dt>
      <dd data-bind="text:gemeente"></dd>
    </dl>
  </script>
  <script id="foto" type="text/html"> (5)
    <h2>Foto</h2>
    
  </script>
  <script src="http://cdnjs.cloudflare.com/ajax/libs/knockout/3.0.0/knockout-
min.js"></script>
  <script src="hoofdzetel.js"></script>
</body>
</html>
```

- (1) Als de gebruiker deze hyperlink aanklikt, voer je de functie `adres` uit in het ViewModel.
- (2) Als de gebruiker deze hyperlink aanklikt, voer je de functie `foto` uit in het ViewModel.
- (3) Deze `div` speelt de rol van placeholder. Hij zal in zich de ene template of de andere template tonen. Hij roept in het ViewModel de functie `template` op. Hij krijgt de naam van de af te beelden template terug. Dit zal ofwel `adres`, ofwel `foto` zijn.
- (4) Dit is de eerste template. Een template is een script met het type gelijk aan `tekst/html` en een `id` die de naam van de template aangeeft. De browser toont de inhoud van een template niet in de browser. Knockout zal ten gepasten tijde de inhoud van de template tonen in de placeholder.
- (5) Dit is de tweede template.

Je maakt een pagina `hoofdzetel.js`

```
var viewModel = {straat : "Keizerslaan", huisnummer:"10", postcode:1000,  
gemeente:"Brussel",  
  template:ko.observable("adres"),  
  adres : function() {  
    this.template("adres");  
  },  
  foto : function() {  
    this.template("foto");  
  }  
};  
ko.applyBindings(viewModel);
```

- (1) De observable `template` verwijst naar het id van de eerste template: `adres`. Knockout toont dan de inhoud van deze template in de placeholder.
- (2) Knockout voert deze functie uit als de gebruiker `Adres` aanklikt. Je verwijst in de observable naar het id van de eerste template: `adres`. Knockout toont dan de inhoud van deze template in de placeholder.
- (3) Knockout voert deze functie uit als de gebruiker `Foto` aanklikt. Je verwijst in de observable naar het id van de tweede template: `foto`. Knockout toont dan de inhoud van deze template in de placeholder.

Je kan de applicatie uitproberen.



Opmerking

In een uitgebreidere applicatie kan een template op zich één of meerdere placeholders bevatten, die verwijzen naar andere templates.



Taak: Personeel: zie takenbundel

12 BACK, FORWARD EN BOOKMARKS

12.1 Algemeen

Als de gebruiker een template ziet, krijgt hij met de back knop van de browser niet de vorige template (van dezelfde pagina) te zien, maar de vorige pagina.

Ook de forward knop van de browser brengt de gebruiker niet op de volgende template, maar op de volgende pagina.

De gebruiker kan ook één template niet bookmarken, want alle templates hebben dezelfde URL.

Je kan dit bijvoorbeeld uitproberen in de pagina `hoofdzetel.html`.

Je leert in dit hoofdstuk hoe je deze problemen oplost.

12.2 URL fragment

Het URL fragment is het laatste deel van een URL dat begint met het teken #.

Je verbindt iedere template met een eigen URL fragment.

In de pagina `hoofdzetel.html` verbind je

- de template `adres` met het URL fragment `#adres`
- de template `foto` met het URL fragment `#foto`

Als de gebruiker de template `adres` ziet, is de URL `hoofdzetel.html#adres`.

Bij de foto template is de URL `hoofdzetel.html#foto`.

De back en forward knoppen van de browser wisselen nu tussen deze twee URL's. De browser blijft daarbij op dezelfde pagina, omdat enkel het URL fragment wijzigt.

Je kan ook de URL `hoofdzetel.html#adres` of de URL `hoofdzetel.html#foto` bookmarken.

Je wijzigt in `hoofdzetel.html` de hyperlinks

```
<a href="#adres">Adres</a>
<a href="#foto">Foto</a>
```

Bemerk dat de hyperlinks geen data-bind meer hebben op het click event.

12.3 Sammy

De volgende stappen zijn:

- als het URL fragment `#adres` is, moet je de template `adres` tonen.
- als het URL fragment `#foto` is, moet je de template `foto` tonen.

Je gebruikt de library `Sammy` om deze stappen uit te coderen.

Je kan deze library downloaden vanaf de website `sammy.js`.

Je voegt in `hoofdzetel.html` onder de eerste `</script>` volgende regel toe

```
<script src="sammy-latest.min.js"></script>
```

`Sammy` gebruikt op zijn beurt de library `jQuery`.

Je voegt boven de vorige regel nog een regel toe die naar deze library verwijst

```
<script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"></script>
```

Je wijzigt hoofdzetel.js

```
var viewModel = {straat : "Keizerslaan", huisnummer : "10", postcode : 1000,  
  gemeente : "Brussel",  
  template : ko.observable()  
};  
Sammy(function() { (1)  
  this.get("#adres", function() { (2)  
    viewModel.template("adres");  
  });  
  this.get("#foto", function() { (3)  
    viewModel.template("foto");  
  });  
  this.get("", function() { (4)  
    viewModel.template("adres");  
  });  
}).run(); (5)  
ko.applyBindings(viewModel);
```

- (1) Je moet Sammy één keer in je applicatie initialiseren. Je doet dit door de functie `Sammy` op te roepen en als parameter een functie mee te geven.
- (2) Binnen deze functie verwijst `this` naar de Sammy library.
Je roept daarop de functie `get` op en je geeft twee parameters mee.
De eerste parameter is een URL fragment.
De tweede parameter is een functie. Als het URL fragment in de adresbalk van de browser gelijk is aan het URL fragment in de eerste parameter, voert Sammy deze functie uit. Je bepaalt in deze functie de af te beelden `template`. Je kan (2) lezen als *wanneer het URL fragment van de huidige URL gelijk is aan #adres, toon dan de template adres*.
- (3) Je kan dit lezen als *wanneer het URL fragment van de huidige URL gelijk is aan #foto, toon dan de template foto*
- (4) Je kan dit lezen als *wanneer het URL fragment van de huidige URL leeg is, toon dan de template adres*. In de `this.get` oproepen (bij 2, 3 en 4) moet je de `get` oproep met een leeg URL fragment (bij 4) als laatste vermelden.
- (5) Je voert op het resultaat van de Sammy functie de functie `run` uit om de initialisatie van Sammy af te ronden.

Je kan de applicatie uitproberen. De back en forward toetsen van je browser brengen je van de vorige naar de volgende template.



Taak: Pannenkoeken: zie takenbundel

12.4 Een URL fragment met een parameter

Je kan in `muziek.html` kiezen uit drie genres. De back en forward toetsen van je browser brengen je echter niet van het vorige naar het volgende genre. Je zal dit hier oplossen.

Je zorgt er eerst voor dat iedere genre hyperlink verwijst naar een URL met een uniek URL fragment. Dit URL fragment wordt `#genres/`, gevolgd door het nummer van het genre. De eerste hyperlink verwijst naar het URL fragment `#genres/1`, de laatste hyperlink verwijst naar het URL fragment `#genres/7`

Je wijzigt daartoe in `muziek.html` de eerste `<a>` tag

```
<a data-bind="text:naam,attr:{href:'#genres/'+nummer}">
```

Je voegt na de eerste `</script>` tag volgende regels toe

```
<script  
src="http://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"></script>  
<script src="sammy-latest.min.js"></script>
```

Als volgende stap configureer je Sammy in `muziek.js`. Je kan hierbij de URL fragmenten, waarop Sammy moet reageren niet hard coderen in JavaScript. Dit zijn tot nu de URL fragmenten `#genres/1`, `#genres/2` en `#genres/7`. Maar als er genres bijkomen, zou je ook de configuratie van Sammy moeten uitbreiden met bijkomende URL fragmenten.

De oplossing bestaat er in Sammy te configureren met een URL fragment met een parameter. Deze parameter staat voor het veranderlijke genrenummer. Het geconfigureerde URL fragment zal `#genres/:nummer` zijn. Dit URL fragment staat voor alle URL's waarvan het URL fragment begint met `#genres/`.

De parameter (die je voorafgaat met `:`) `nummer` staat voor de rest van dit URL fragment. Dit is bij ons het genrenummer.

Je wijzigt `muziek.js`

```
var viewModel = {  
  genre : ko.observable(),  
  artiesten : ko.observableArray()  
};  
Sammy(function() {  
  this.get("#genres/:nummer", function() { (1)  
    var genreNummer = parseInt(this.params["nummer"]); (2)  
    vulArtiesten(genreNummer);  
  });  
  this.get("", function() {  
    if (viewModel.genres === undefined) {  
      vulGenres();  
    }  
    viewModel.artiesten.removeAll();  
  });  
}).run();  
  
function vulGenres() {  
  var xmlHttpRequest = new XMLHttpRequest();  
  xmlHttpRequest.open("GET", "genres.json", true);  
  xmlHttpRequest.onreadystatechange = function() {  
    if (xmlHttpRequest.readyState === 4 && xmlHttpRequest.status === 200) {  
      viewModel.genres = JSON.parse(xmlHttpRequest.responseText);  
      ko.applyBindings(viewModel);  
    }  
  };  
  xmlHttpRequest.send();  
}  
  
function vulArtiesten(genreNummer) {  
  var xmlHttpRequest; = new XMLHttpRequest();  
  xmlHttpRequest.open("GET", "genres." + genreNummer + ".artists.json", true);  
  xmlHttpRequest.onreadystatechange = function() {  
    if (xmlHttpRequest.readyState === 4 && xmlHttpRequest.status === 200) {  
      var artiesten = JSON.parse(xmlHttpRequest.responseText);  
      viewModel.artiesten(artiesten);  
    }  
  };  
  xmlHttpRequest.send();  
}
```

- (1) Je kan dit lezen als *wanneer het URL fragment van de huidige URL begint met #genres/, voer dan een function uit.*
- (2) Je leest de inhoud van de parameter :nummer met `this.params["nummer"]`.

De pagina werkt als de gebruiker eerst naar `muziek.html` surft en binnen die pagina de hyperlinks gebruikt. De pagina werkt niet als de gebruiker direct naar één van de URL fragmenten met een genrenummer (bvb. `muziek.html#genres/7`) surft.

In dat geval roept Sammy de function `vu1Artiesten` op, maar is de variabele `genres` in het ViewModel nog niet ingevuld. De gebruiker ziet dan een lege lijst met genres.

Je lost dit op door na de opdracht

```
var genreNummer = parseInt(this.params["nummer"]);
```

volgende opdrachten toe te voegen

```
if (viewModel.genres === undefined) {  
    vu1Genres();  
}
```

Je kan de pagina uitproberen.



Taak: Personeel 2: zie takenbundel

13 KNOCKOUT VALIDATION

13.1 Algemeen

Als de gebruiker een textbox wijzigt, moet je meestal de invoer valideren.

Het project Knockout Validation is een Knockout plugin om invoer te valideren.

Je vindt dit project op <https://github.com/Knockout-Contrib/Knockout-Validation>.

Je opent in deze pagina de map `Dist`

en je slaat `knockout.validation.min.js` op in je werkdirectory.

Je keert één niveau terug. Je opent de map `Localization` en je slaat `nl-NL.js` op in je werkdirectory. Dit bestand bevat nederlandstalige foutmeldingen.

Je maakt als voorbeeld een programma om een wiskundige deling te doen.

Je maakt een bestand `delen.html`

```
<!doctype html>
<html lang="nl">
<head>
<meta charset="UTF-8">
<title>Delen</title>
<link rel="stylesheet" href="default.css">
</head>
<body>
  <h1>Delen</h1>
  <div><label>Teller:
  <input data-bind="value:teller, valueUpdate:'keyup'" type="number"
required></label></div> (1)
  <div><label>Noemer:
  <input data-bind="value:noemer, valueUpdate:'keyup'" type="number" required
min="1"></label></div> (2)
  <h2>Deling:<span data-bind="text:deling()"></span></h2>
  <script src="http://cdnjs.cloudflare.com/ajax/libs/knockout/3.0.0/knockout-
min.js"></script>
  <script src="knockout.validation.min.js"></script> (3)
  <script type="text/javascript" src="nl-NL.js"> </script> (4)
  <script src="delen.js"></script>
</body>
</html>
```

- (1) Je geeft met het attribuut `required` aan dat dit veld verplicht in te vullen is.
- (2) Je geeft met het attribuut `required` aan dat dit veld verplicht in te vullen is.
Je geeft met het attribuut `min` aan dat de minimum waarde 1 is.



Opmerking

Naast `required` bestaan ook `max` en `pattern`. Je geeft aan `pattern` een regular expression mee. De tekst in de textbox moet overeenkomen met deze expressie.

Je maakt een bestand `delen.js`

```
var viewModel = {
  teller:ko.observable(0),
  noemer:ko.observable(1),
  deling:function() {
    if (this.isValid()) { (1)
      return this.teller() / this.noemer();
    }
  }
};
```

```
    }  
    else {  
        return "";  
    }  
}  
};  
ko.validation.init({  
    parseInputAttributes:true  
});  
ko.applyBindings(ko.validatedObservable(viewModel));
```

(2)
(3)
(4)

- (1) De functie `isValid` geeft enkel `true` terug als alle textboxes een geldige waarde bevatten.
- (2) Je kan de validatieinstellingen wijzigen met de functie `init`.
- (3) De plugin verwerkt de HTML validatie attributen (`min`, ...) standaard niet. De plugin doet dit wel als je `parseInputAttributes` op `true` plaatst.
- (4) De plugin werkt enkel als je het `ViewModel` aanbiedt aan de functie `validatedObservable`.

Je kan de pagina uitproberen.

13.2 Plaats van de foutmeldingen

De plugin toont de foutmeldingen standaard na de textboxes.
Je kan ook zelf beslissen waar je de foutmeldingen wil tonen.
Je toont als voorbeeld de foutmeldingen naast de labels.

Je voegt in `delen.html` voor de eerste `<input ...>` volgende code toe

```
<span data-bind="validationMessage:teller" class="fout"></span>
```

Je voegt voor de tweede `<input ...>` volgende code toe

```
<span data-bind="validationMessage:noemer" class="fout"></span>
```

Je wijzigt in `delen.js` de oproep van de functie `init`

```
ko.validation.init({  
    parseInputAttributes:true,  
    insertMessages:false  
});
```

(1)

- (1) Als je `insertMessages` op `false` plaatst, toont de plugin de foutmeldingen niet meer op hun standaard plaats (na de textboxes).

Je kan de pagina uitproberen.



Opmerkingen

- Het `ViewModel` kan JavaScript objecten bevatten die op hun beurt observables bevatten. Om deze observables te valideren moet je in de functie `init` de property `grouping` op de waarde `{deep:true}` plaatsen.
- De plugin valideert de textbox standaard enkel als je de textbox wijzigt. Om de textbox altijd te valideren na het verlaten van de textbox moet je in de functie `init` de property `messagesOnModified` op `false` plaatsen.

14 COLOFON

Domeinexpertisemanager: Rita Van Damme

Moduleverantwoordelijke: Hans Desmet

Medewerkers: Hans Desmet

Versie: 28/10/2013

Nummer dotatielijst: