|  |
| --- |
| Atos |
| Workshop Javascript |
| jQuery, jQueryUI, KnockoutJS |



|  |
| --- |
| Marco Franssen, Jan Saris, Sander van de Velde  23-9-2011 |



Inhoudsopgave

[1. Basic jQuery 3](#_Toc305179739)

[1.1 Create a MVC3 project 3](#_Toc305179740)

[2. Ajax binnen jQuery bereik 20](#_Toc305179741)

[2.1 Terminologie 20](#_Toc305179742)

[2.2 Implementatie 21](#_Toc305179743)

[2.3 Testen 26](#_Toc305179744)

[2.3.1 Request 26](#_Toc305179745)

[2.3.2 Response 27](#_Toc305179746)

[3. jQuery UI 29](#_Toc305179747)

[3.1 Scripts 29](#_Toc305179748)

[3.2 Dialog 29](#_Toc305179749)

[3.3 Datepicker 31](#_Toc305179750)

[3.4 Buttons 32](#_Toc305179751)

[3.4.1 Checkbox buttons 32](#_Toc305179752)

[3.4.2 Radio buttons 32](#_Toc305179753)

[3.4.3 Group buttons 32](#_Toc305179754)

[3.5 Animations / effects 32](#_Toc305179755)

[4. Advanced Javascript 33](#_Toc305179756)

[4.1 Create your own methods 33](#_Toc305179757)

[4.2 Create a javascript object 34](#_Toc305179758)

[4.3 Communicate between objects with events 34](#_Toc305179759)

[5. MVVM met Knockout JS 35](#_Toc305179760)

[5.1 Simple bindings 35](#_Toc305179761)

[5.2 Advanced bindings 35](#_Toc305179762)

[5.3 Cascading dropdown 35](#_Toc305179763)

[5.4 Animations with jQueryUI 37](#_Toc305179764)

[6. References 38](#_Toc305179765)

# Basic jQuery

Welkom bij deze introductie op jQuery. Dit hoofdstuk zal in het teken`staan van browser DOM manipulatie. Voor diegene die niet weet war de browser DOM is, dis is een afkorting voor Document Object Model, oftewel een boomstructuur van alle html elementen op het scherm. Deze elementen kunnen genest zijn.

## Create a MVC3 project

Open een Kaal mvc3 project genaamd JqueryWorkShopMvcApplication aan in VS2010   
(met razor en HTML5 ondersteuning)



We kunnen jQuery met de hand ophalen van de website (<http://docs.jquery.com/Downloading_jQuery>) maar we kunnen ook Nuget gebruiken. Maar standaard is er al een (oudere) versie van jQuery aanwezig in een MVC3 project. Die passen we toe.

Controleer dus dat de \_Layout.cshtml (de masterpage equivalent in Razor) (in \Views\Shared) met de juiste referenties gevuld is.

|  |
| --- |
| <!DOCTYPE html> <html> <head>     <meta charset="utf-8" />     <title>@ViewBag.Title</title>     <link href="@Url.Content("~/Content/Site.css")" rel="stylesheet" type="text/css" />     <script src="@Url.Content("~/Scripts/jquery-1.5.1.min.js")" type="text/javascript"></script> </head>   <body>     @RenderBody() </body> </html> |



**Tip1**: de standaard aanwezige modernzr javascript bibliotheek is weggelaten (buiten scope voor onze workshop)

**Tip2**: Eventueel kan /Scripts/jquery-1.5.1.js gerefereerd worden. Dit bestand met exact dezelfde logica is niet geminified. Dwz. De minified versie is geoptimaliseerd voor het internet door deze zo klein mogelijk te maken (commentaar verwijderd, functienamen verkort, etc.) De niet-minified versie is prettiger bij javascript debug werkzaamheden.

Voeg een HomeController toe (rechtermuisclick op de de controllers map)



Noem deze HomeController



Zie dat er een Index methode aanwezig is. Dit is een MVC action.

Maak een View aan voor deze action (rechtermuisclick op de methode)



Gebruik de standaard instellingen:



De View is aangemaakt:

|  |
| --- |
| @{     ViewBag.Title = "Index"; }   <h2>Index</h2> |

Maak een home.index.js aan in de scripts map via add New item. Hier gaan we de jQuery statements in schrijven, deze code is namelijk te debuggen via breakpoints in de ‘gutter’.





Het javaScript is nu geopend in de Editor.   
Sleep nu de jquery-1.5.1-vsdoc.js in de text editor . Er komt regel ‘commentaar’ beschikbaar:

|  |
| --- |
| /// <reference path="jquery-1.5.1-vsdoc.js" /> |

Dit maakt het voor VS2010 mogelijk om code inside aan te bieden voor jQuery. Erg handig.

Voeg een lege functie toe:

|  |
| --- |
| /// <reference path="jquery-1.5.1-vsdoc.js" />   function ExecuteOnStartup() {   // Add your script here } |

Refereer nu vanuit de index.cshtml naar het javascript bestand:

|  |
| --- |
| @{     ViewBag.Title = "Index";             <script src="@Url.Content("~/Scripts/home.index.js")" type="text/javascript"></script> }   <h2>Index</h2> |

Alles staat nu klaar om met de workshop te beginnen.   
**Compileer even om te controleren of alles werkt ;-)**

Dit moet de volgende pagina geven:



De gegenereerde html is eenvoudige maar duidelijke html 5 opmaak.

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <title>Index</title>  <link href="/Content/Site.css" rel="stylesheet" type="text/css" />  <script src="/Scripts/jquery-1.5.1.js" type="text/javascript"></script>  </head>  <body>  <script src="/Scripts/home.index.js" type="text/javascript"></script>    <h2>Index</h2>  </body>  </html> |

Deze is gegenereerd door de combinatie van \_Layout.cshtml en de index.cshtml

Voeg nu een aantal html objecten en een stukje javascript toe aan de index.cshtml:

|  |
| --- |
| @{     ViewBag.Title = "Index";             <script src="@Url.Content("~/Scripts/home.index.js")" type="text/javascript">  </script> }   <h2>Index</h2>   <a id="link1" href="">Link1</a>  <br />  <a id="link2" href="">Link2</a>   <script type="text/javascript">     $(document).ready(function () {         ExecuteOnStartup();     }); </script> |

We zien hier twee links zonder echte logica (er zal niks gebeuren als je er op klikt).

Het stukje script zal uitgevoerd worden als deze pagina in de browser laadt. De opdracht daarin is niks anders dan een stukje jQuery code wat zegt:

“Zodra de pagina geheel in de browser geladen is en de gebruiker bijna de eerste controls kan gaan aanklikken, dan moet de ExecuteOnStartup methode uitgevoerd worden”.

Dit is essentieel! De .ready(); gaat pas af op dat moment af en dit kan per merk browser verschillen! jQuery lost specifieke browser afhankelijkheden voor ons op.

We passen de ExecuteOnStartup() iets aan:

|  |
| --- |
| function ExecuteOnStartup() {       // add to all links (the a elements) a click function     $("a").click(function () {         alert("Hello jQuery workshop (from " + this.id + ")");     }); } |

Hier zien we het eerste voorbeeld van een jQuery selector. Voor alle elementen die voldoen aan de eis dat ze simpelweg een link zijn, wordt aan hun click event een functie toegevoegd. Hierin wordt een dialoogje getoond met daarin een tekst en de id van het element waarop geclicked is.

**Start nu de applicatie en bekijk het resultaat:**



We klikken op ieder element:

|  |  |
| --- | --- |
| De H2 | Er gebeurt niks |
| De eerste link |  |
| De tweede link |  |

Iedere link voerst dus zijn eigen functie uit. Een selector kan dus 0, 1 of meerdere elementen opleveren waar iets mee gedaan moet worden.

jQuery bevat vele varianten op selectors. Er staat een uitputtende lijst op <http://api.jquery.com/category/selectors/>

We kunnen ook functies aan andere DOM element events hangen. Zie ook <http://api.jquery.com/category/events/>

Hier gaan we dieper op in.   
We passen dus de ExecuteOnStartup weer iets aan:

|  |
| --- |
| function ExecuteOnStartup() {       // add to all links (the a elements) a click function     $("#link1").mouseenter(function () {         $("#link2").hide(500);     });       $("#link1").mouseleave(function () {         $("#link2").show(1000);     }); } |

Wat we nu zien is data an het specifieke element link1 (en alleen deze; dat komt door het # wat jQuery vertelt alleen naar elementen met deze id te zoeken) maar liefst twee functies te koppelen.

Als de muis naar link1 gaat, dan wordt specifiek link2 onzichtbaar gemaakt. Dit wordt uitgevoerd in 500 milliseconden en dit geeft dus een animatie. Als de muis link1 weer verlaat, dan moet link2 weer zichtbaar gemaakt worden in 1000 milliseconden. Ook dit geeft dus een animatie.

**Probeer het maar eens:**



Zie dat de tweede link langzamer terugkomt dan verdwijnt.

We zien hier twee belangrijke aanwinsten met jQuery. De selectors maken het veel eenvoudiger om specifieke elementen te vinden, zelfs als deze genest zijn zoals rijen in een tabel. En jQuery beidt simpele animaties wat een prettige beleving van de pagina voor de gebruiker is.

Er zijn flink wat effecten mogelijk deze gaan we in hoofdstuk 3.5 bespreken.

Overigens, wie een breakpoint in de [home.index.js](http://home.index.js) heeft opgenomen, ziet dat VS2010 netjes stopt op deze breakpoint (dit werkt alleen als je in Internet Explorer test):



Met bv. het Immeadiate scherm is de inhoud van de javascript elementen te acherhalen.

Laten we eens wat meer proberen met de selectors.   
Voeg eerst een aantal paragraph elementen toe achter de a elementen:

|  |
| --- |
| ...  <p>Hello</p>  <p>and</p>  <p>Goodbye</p>  ... |



En we maken een stylesheet bestand aan in de Content map:



Noem deze home.index.css



En voeg deze toe aan de index.cshtml:

|  |
| --- |
| @{     ViewBag.Title = "Index";             <script src="@Url.Content("~/Scripts/home.index.js")" type="text/javascript"></script>     <link href="@Url.Content("~/Content/home.index.css")" rel="stylesheet" type="text/css" /> }   <h2>Index</h2>   <a id="link1" href="">Link1</a> <br /> <a id="link2" href="">Link2</a>   <p>Hello</p>  <p>and</p>  <p>Goodbye</p>    <script type="text/javascript">     $(document).ready(function () {         ExecuteOnStartup();     }); </script> |



In de css moeten de volgende ‘styles’ toegevoegd worden:

|  |
| --- |
| p { margin: 8px; font-size:16px; }       .selected { color:blue; } |

En aan de ExecuteOnStartup voegen we de volgende regel toe:

|  |
| --- |
| …  var lastParagraph = $("p:last"); alert("the text within the last paragraph was:" + lastParagraph.text()); lastParagraph.addClass("selected").text("BYE BYE");  … |



Voer dit eens uit. Eerst volgt er een alert:



Dan volgt de volgende weergave in de browser:



Wat is er gebeurd? Deze volgende situatie is ontstaan:

* Alle laatste paragrafen (eentje dus) worden door een variabele gerepresenteerd.
* We geven een alert af met de inhoud van de laatste paragraaf
* Alle drie de paragrafen hebben een andere marge en lettergrootte gekregen (dit is standaard css gedrag)
* En we hebben gecodeerd dat de LAATSTE paragaaf de extra selected stijl heeft gekregen
* En we hebben de tekst in de paragraaf overschreven

Tip: Zag je dat we in de laatste regel dezelfde paragraaf twee maal manipuleerden? Alle functies (zoals de .addClass(“…”) en de .text(“…”)) manipuleren de aangeboden lijst van objecten en geven die daarna weer terug. De elementen die dus door de .addClass zijn gemanipuleerd, worden dus ook door de .text gemanipuleerd. Dit noemen we een *Fluent* notatie en dat zijn we ook terug bij bv. LinQ en EF 4.1 Code First.

De .ready() is was dus ook al een selector, wat grappig.

We kunnen dus met jQuery ook de attributen van (meerdere of afzonderlijke) elementen uitlezen en aanpassen. En we kunnen de CSS stijlen uitlezen en aanpassen.

Het is zelfs mogelijk op te selecteren op het wel of niet hebben van een bepaalde CSS stijl. Dit is erg krachting. Zo kunnen heel snel afzondelijke elementen een nieuwe CSS look krijgen.

Laten we nog iets meer met selectors spelen.   
Voeg aan de index.cshtml een tabel toe met een kolomnaam en view regels

|  |
| --- |
| <table border="1">  <thead>     <tr><td>Header</td></tr>  </thead> <tbody>     <tr><td>Row with Index #0</td></tr>      <tr><td>Row with Index #1</td></tr>      <tr><td>Row with Index #2</td></tr>      <tr><td>Row with Index #3</td></tr>  </tbody> </table> |

Voeg wederom aan de ExecuteOnStartup wat javascript toe:

|  |
| --- |
| …  $("table thead tr:even").css("background-color", "#bb00ff");   $("table tbody tr:even").css("background-color", "#bbbbff");  … |



En we hebben nog wat extra styling nodig voor de table in de home.index.css :

|  |
| --- |
| …  table {    background:#eeeeee;  }  … |

Dit geeft de volgende tabel invulling:



Wat is er gebeurd:

* De totale tabel heeft een achter achtergrond kleurtje gekregen (standaard css gedrag)
* De even regels in de tabel head krijgen een ander kleurtje
* De even regels in de tabel body krijgen een ander kleurtje

We kunnen dus bij de selector ook specifiek genest gaan zoeken. Dit heeft de voorkeur bij grote documenten met veel DOM elementen waar slechts enkele objecten moeten worden aangepast. Het zoeken naar elementen is gewoon het aflopen van eigenlijk alle elementen en dit kost tijd en energie (het verbruikt tenslotte stroom van je PDA en het is toch jammer als merkbaar minder kunt bellen na het bezoek van een website).

Overigens zal de header row NIET gemanipuleerd worden als de de oneven regels kleuren in de header:

|  |
| --- |
| $("thead tr:odd").css("background-color", "#bb00ff"); |



Dit geeft:



De regel met daarin “Header” is de nulde regel, dus een even regel, niet een oneven regel…

Conclusie met slechts enkele regels jQuery code is het heeft eenvoudig geworden om een aantrekkelijke (..) pagina te creeeren. Had jij dit ook met standaard javascript kunnen schrijven en had die code ook in alle browsers gewerkt?

Overigens is De DOM, dus de boomstructuur aan elementen zoals de browser die zeit, ook in te zien in moderne browsers of via plugins zoals Firebug:



Als laatste gaan we nog de items in een ordered list manipuleren.

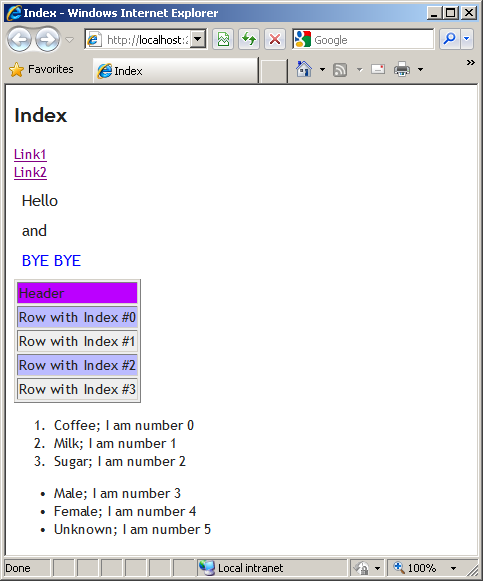
We voegen eerst twee ‘ordered’ lists (in html worden volgnummers of bullits voor de items gezet) toe onderaan de index.cshtml

|  |
| --- |
| <ol class="orderedlist">     <li>Coffee</li>     <li>Milk</li>     <li>Sugar</li> </ol>    <ul class="orderedlist">     <li>Male</li>     <li>Female</li>     <li>Unknown</li> </ul> |

Daarna voegen we de volgende regels JavaScript toe aan de ExecuteOnStartup:

|  |
| --- |
| $(".orderedlist").find("li").each(function (i) {         $(this).append("; I am number " + i);     }); |

Als we dit uitvoeren dan zien we dat achter iedere *list item* extra text is toegevoegd:



De ‘ordered’ lists zijn overigens gekozen via zijn implementatie van een CSS class. En het interessante is dat de telling doorloopt. Hoe komt dat? Waarom is er niet opnieuw geteld (dus 0,1,2,3,4,5 ipv. 0,1,2,0,1,2)?

Dit komt omdat de $(".orderedlist") simpelweg een stukje DOM representeerd, hier de twee lijstjes. Daarin gaat de find("li") alle list items achterhalen. Dus er worden zes list items aangetroffen en niet twee keer drie items. Let hier dus goed op.

***Bonus opdracht 1:***

|  |
| --- |
| Hoe zou jij de vraag naar telling 0,1,2 0,1,2 in de laatste opdracht oplossen? |

***Bonus opdracht 2:***

|  |
| --- |
| Voeg een checkbox toe aan de pagina en als ik die van waarde verander, toon of verberg dan de tabel. (hints: input, type , checkbox, .change() .toggle()) |

# Ajax binnen jQuery bereik

In het vorige hoofdstuk hebben we gezien hoe we de DOM kunnen onderzoeken en manipuleren via jQuery. In dit hoofdstuk willen we laten zien hoe makkelijk het is geworden om ajax calls te doen binnen een website[[1]](#footnote-1).

## Terminologie

We kennen allemaal wel de mogelijkheid om een Post uit te voeren in de browser. Veel schermen met invoer, zelfs inlogschermen werken met enkele invoervelden en een ‘submit’ button. Over het algemeen wordt dan alle invoer naar de server gestuurd, deze wordt afgehandeld door de achterliggende serverlogica en er wordt een nieuw scherm gegenereerd. Dit heet een **post-back**.

In dit hoofdstuk gaan we een **call-back** uitvoeren. De gebruiker (of zoiets als een timer of in moderne browesers kan dit een gebeurtenis op het OS zijn) initieert de uitvoer van een stukje JavaScript. Er is bv. wat data ingevoerd. Vervolgens wordt deze informatie naar de server doorgegeven, alleen deze data. Die data moet op de server worden verwerkt en de call-back geeft weer informatie terug. Dat is in het algemeen alleen data, geen stukken schermopmaak.

Die callback kan wellicht mislukken of lang op zich laten wachten. Daarom gebeurt dit **asynchroon**. De browser vuurt de aanroep af en blijft niet wachten op een antwoord. Nee, er wordt alleen een voorschot genomen in de vorm van: als er ooit een antwoord komt, dan moet bij terugkomst nog wat extra code uitgevoerd worden. Dit zijn de callback functies. En de attributen van die functies zijn de antwoorden van de server.

Ok, simpel. Maar hoe ziet die communicatie tussen browser en server er uit? Dit is in het algemeen opgelost met een query string voor de aanroep en het antwoord is verpakt in **Json**. Json is niks anders dan een alternatieve notatie, net zoals xml dat is. Xml is gewoon minder toepasselijk om dat deze tot vrij grote berichten kan leiden (de belangrijkste reden zijn die dubbele omarmingen). Json is lichtgewicht en het heeft nog een ander voordeel. Json data is heel simpel naar JavaScript variabelen om te zetten, dit kost amper processor tijd.

## Implementatie

Laten we beginnen met het aanbieden van data voor Ajax calls.   
We maken een extra action (methode) aan op de server.

|  |
| --- |
| [AcceptVerbs(HttpVerbs.Post)] public ActionResult DoSomething(string name, string phone, int? zip, string email) {             var myJsonResult = new MyJsonResult();             try             {                 // do some filtering                  Thread.Sleep(2000);                   List<Person> persons = new List<Person>                 {                     new Person{ID=1, Name="Donald Duck", Age=77},                     new Person{ID=2, Name="Kwik", Age=74},                     new Person{ID=3, Name="Kwek", Age=74},                     new Person{ID=4, Name="Kwak", Age=74}                 };                   myJsonResult.Persons = persons.ToArray();                   myJsonResult.Succeeded = true;             }             catch (Exception)             {                 myJsonResult.Succeeded = false;             }               return new JsonResult { Data = myJsonResult };         } |

Hier wordt dus een methode aangeboden om name/phone/zip en email door te geven. Omdat dit met een Post is geïnstrumenteerd, zou dit kunnen duiden op het feit dat we hier data gaan wijzigen.

***Tip***: de nullable int is nullable om te voorkomen dat we een 500 (internal server errror) krijgen als de aangeboden data ‘undefined’ is in JavaScript. We moeten nu wel testen of alles correct doorgegeven wordt maar dat is het altijd met Ajax: vertrouw nooit wat je van buiten aangeleverd krijgt!

Voor onze workshop is de functionaliteit iets simpeler: we wachten twee seconden en geven een lijstje met niet al te jeugdigden terug. De C# objecten die we teruggeven zijn:

|  |
| --- |
| public class MyJsonResult {         public bool Succeeded { get; set; }         public Person[] Persons { get; set; }     }       public class Person {         public int ID { get; set; }         public string Name { get; set; }         public int Age { get; set; }     } |

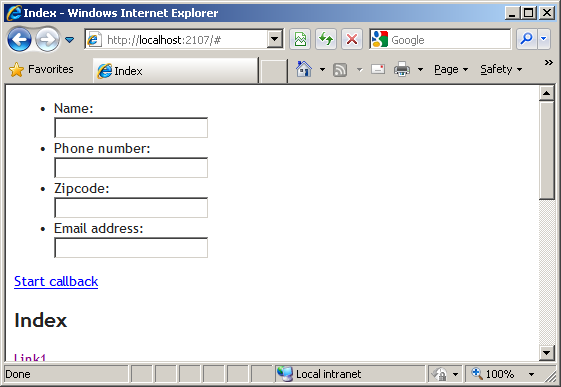
Maak dus deze classen aan in de Models map van onze MVC applicatie.

Het belangrijkste van deze logica op de server is geschreven op de laatste regel van de action. Hier geven we een speciale overerving van ActionResult terug, namelijk onze objecten verpakt als JsonResult.

In de index.cshtml moeten we de aanroep gaan voorbereiden. Eerst zorgen we dat er een aantal velden beschikbaar komen waarin we de gebruiker eea. laten invoeren. Voeg dit bij voorkeur BOVENAAN in de index.cshtml toe. Dit geeft later een beter effect…

|  |
| --- |
| …  <hr/>   <ul>       <li>         Name:         <br />         <input type="text" name="name" placeholder="name" id="name" />     </li>       <li>         Phone number:         <br />         <input type="tel" name="phone" placeholder="Phone" id="phone" />     </li>       <li>         Zipcode:         <br />         <input type="number" name="zip" placeholder="Numbers" id="zip" />     </li>     <li>         Email address:         <br />         <input type="email" name="Email" placeholder="Please enter your email address" id="email" />     </li>   </ul>   <a href="#" id="btnStartCallback">Start callback</a>  <div id="result"></div>  <div id="errorInfo"></div>  … |

 Dit zal er dan zo uit gaan zien.



Hier kan de gebruiker eea. invullen. We gebruiken hier een link om een click op uit te laten voeren. Dat had beter een echte submit butten kunnen zijn. Dit voorkomt dat spiders per ongeluk de ajaxcall aftrappen en zo onze database ongewild vullen of legen.

Nu moet dus deze ingevulde informatie (de zipcode is een integer) naar een action “DoSomething” op de Home controller verstuurd worden. We laten op de server het adres samenstellen door de viewengine. Daarvoor laten de url van de ajax call samenstellen op de index.cshtml en het adres wordt dan op de client beschikbaar als javascript parameter van de ExecuteOnStartup:

|  |
| --- |
| <script type="text/javascript">     $(document).ready(function () {         ExecuteOnStartup('@Url.Action("DoSomething", "Home")');     }); </script> |

Tip: Deze manier van URL samenstellen zorgt ervoor dat je ajax call ook blijft werken als de code uiteindelijk op de server (lees: andere machine) wordt uitgerold.

De parameter moet dus doorgegeven gaan worden aan de functie:

|  |
| --- |
| function ExecuteOnStartup(callbackUrl) {  … |

Wat rest is dan nog de call zelf. We gebruiken hiervoor de (ietswat uitgebreide) .ajax(). Voeg de volgende code toe aan de ExecuteOnStartup

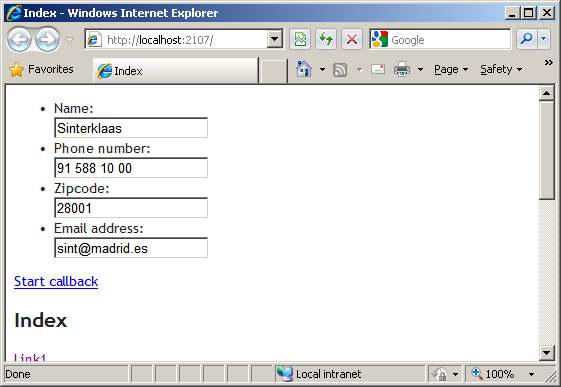
|  |
| --- |
| $('#errorInfo').hide(); $('#name').val("Sinterklaas"); $('#phone').val("91 588 10 00"); $('#zip').val("28001"); $('#email').val("sint@madrid.es");  $('#btnStartCallback').bind('click', function (e) {     var name = $('#name').val();     var phone = $('#phone').val();     var zip = $('#zip').val();     var email = $('#email').val();       var dataToSend = {   'name': name,         'phone': phone,         'zip': zip,         'email': email     };       $.ajax({         url: callbackUrl,         data: dataToSend,         dataType: 'json',         type: 'POST',           beforeSend: function () {             // before callback              $('#errorInfo').show().text("Busy...");             $('#result').empty();         },          success: function (result) {             if (result.Succeeded) {                 // after a successfull callback                 $.each(result.Persons,function () {                     $('#result').append(this.Name + ";");                 });                                      $('#errorInfo').text("Ready!").hide(2000);             }             else {                 // after an unsuccessful callback                 $('#errorInfo').text("Invalid input");             }         },           error: function (xhr, err) {             // after an error             $('#errorInfo').text("Status: " + xhr.status);         }     }); }); |

Wat zien we hier? Er is een div-je waarin errors en andere ondersteunende boodschappen getoond worden. Die verstoppen we tijdens de ready.

Vervolgens vullen we de invoer controls met verschillende waarden (dit gaat via de .val(‘some value’) functie).

Dat wordt er aan de start url een binding gelegd tussen het click event en de een functie. Dit is een alternatieve notatie voor de .click() (dit had ook een ‘tap’ op een iPhone kunnen zijn ☺) De .bind() is dus een generieke manier van functies aan events *binden*.

Uiteindelijk wordt nog vóór het uitvoeren van de callback het scherm zo getoond:

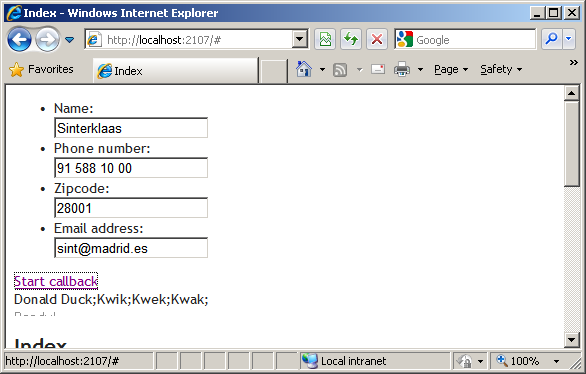


Als er dus een click event optreedt, dan wordt de volgende code uitgevoerd:

1. Eerst lezen de values van de verschillende invoer controls uit via .val(). Deze waarden worden tot één JavaScript variable samengevoegd.
2. Daarna volgt de ajaxcall waarbij we de volgende zaken instellen:

|  |  |
| --- | --- |
| url | De aan te roepen url voor de callback. Hier is dant /Home/DoSomething |
| data | De door te geven data |
| dataType | Het formaat waarin gecommuniceerd gaat worden. Dit is hier json |
| type | De type HTTP aanroep, in dit geval een POST (gelijk aan de action op de controller) |
| beforeSend | De functie die uitgevoerd moet worden voordat de aanroep plaats vind. Ik gebruik het om de text “Busy” te tonen. Dit is gedaan met een Fluent notatie. En ik leeg het te vullen resultaten div-je met .empty(). Dit verwijderd alle tekst en html elementen binnen die div. |
| success | De functie die uitgevoerd moet worden als de ajax call succesvol wordt uitgevoerd.  Het aangeleverde resultaat (de myJsonResult) wordt hier als ‘result’ doorgegeven. Hiervan kunnen we direct de result.Succeeded uitlezen en daarna itereren we door de lijst van result.Persons via de $.each(). Hier zal voor ieder item een funtie uitgevoerd worden waarbij de ‘this’ het betreffende ‘person’ voorsteld |
| error | Deze functie wordt uitgevoerd als er onverhoopt een probleem optreedt tijdens de communicatie. Dit komt nog wel eens voor als de verkeerde actie aangeroepen wordt of als de verkeerde parameters meegegeven worden. |

Voer nu deze code uit. Zie dat de velden gevuld zijn. Pas eventueel de inhoud aan en druk op de Start Callback url. We komen nu op de server waar de sleep even op zich laat wachten. Na teurgkomst worden de eendennamen getoond en zal de tekst “Ready” getoond worden (die daarna langzaam verdwijnt).

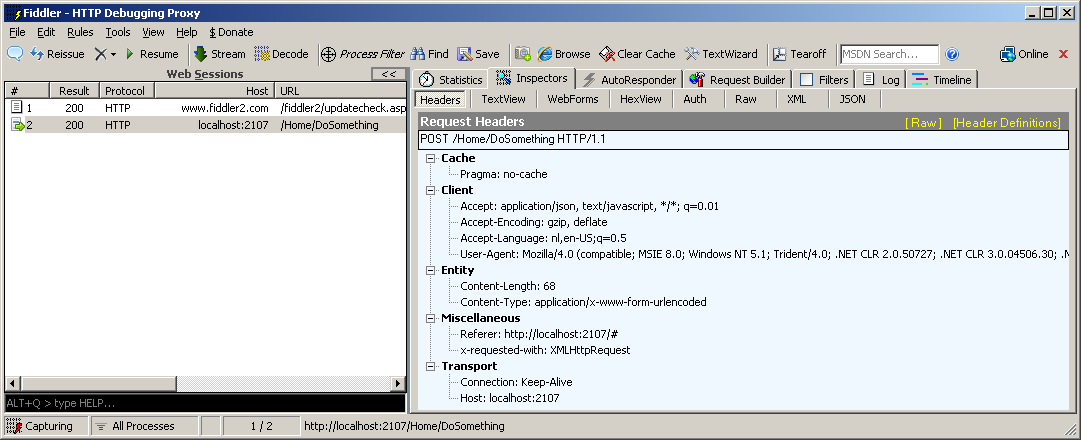


## Testen

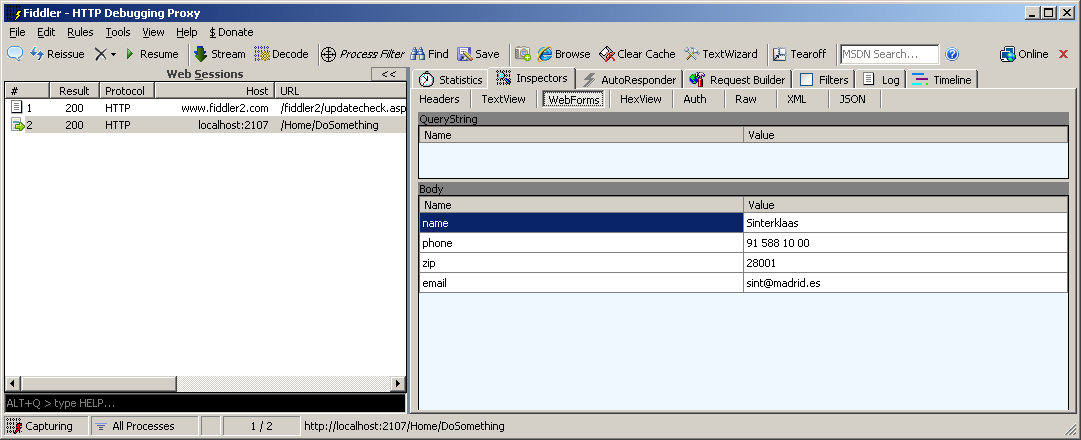
Het is een goede gewoonte om ajax logica te bouwen EN te testen samen met een tool als Fiddler (moderne browsers hebben dezelfde functionaliteit ook ingebouwd). Fiddler werkt als een extra lokale proxy waar al het webverkeer van de browser doorheen gevoerd wordt. Het is dus perfect om hiermee te ‘luisteren’ naar de vragen van de browser en de antwoorden van de server. Kijk maar eens mee wat Fiddler weet te vertellen over de callback.

### Request

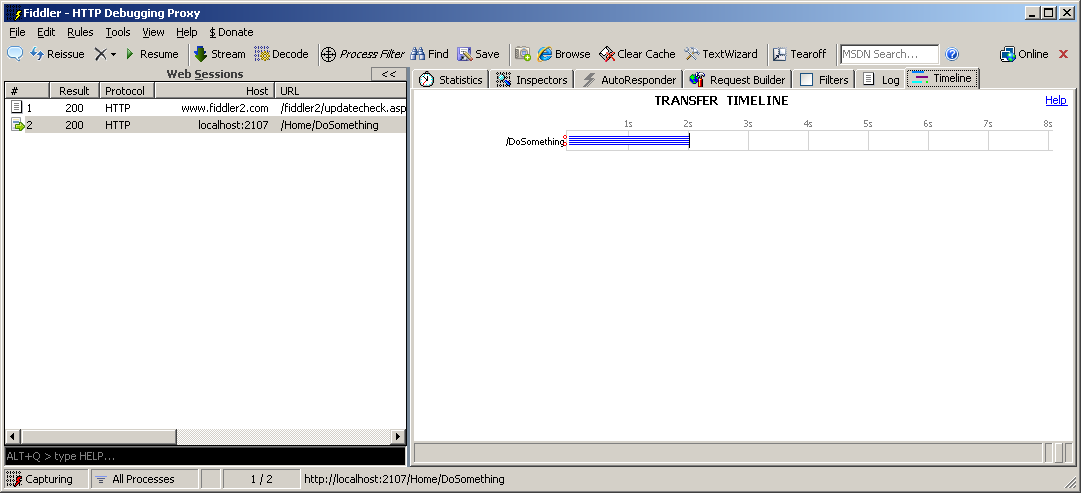
De call was een POST naar /Home/DoSomething



De meegegeven waarden zijn de volgende:

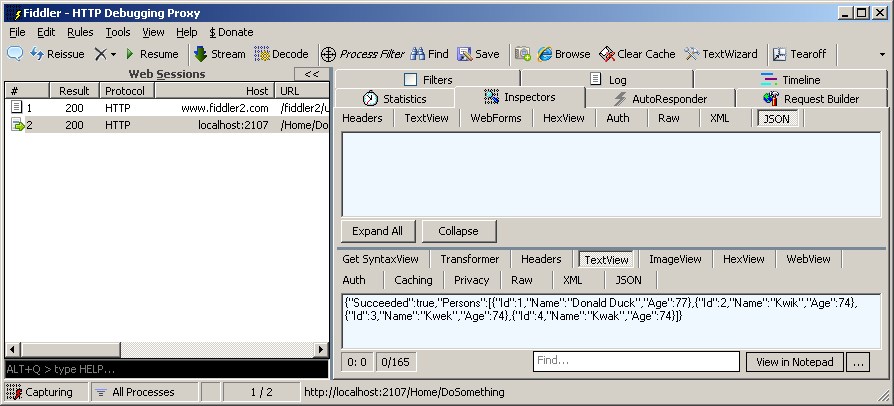


De call duurde zo’n 2 seconden:

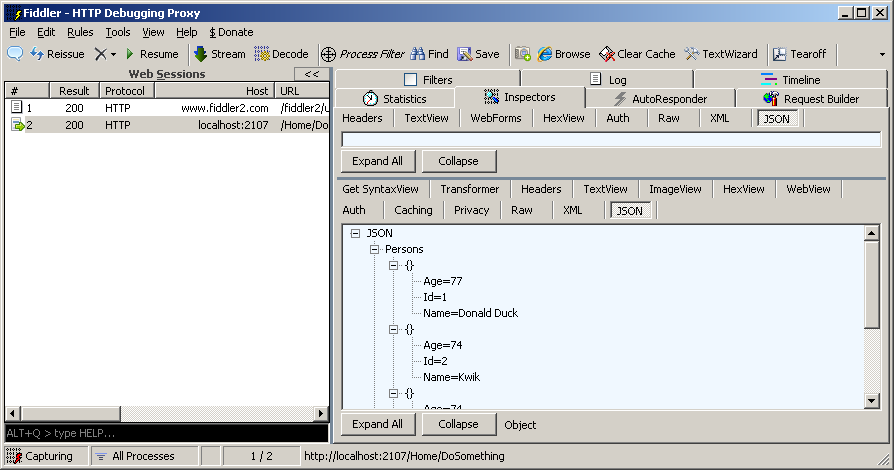


### Response

Dat de call met een 200 terug kwam was al zichtbaar. Hier zien de we ruwe Json terugkomen.



Voor complexere constructies is ook een andere notatie mogelijk:



***Extra opdracht 1:***

|  |
| --- |
| Probeer ook eens met de code te spelen zodat ook de error functie en de andere situatie in de Success functie optreden |

***Extra opdracht 2:***

|  |
| --- |
| doe ook eens een GET |

TODO: verwijzing naar $.Get() en $.Post()

Url jquery ajax opnemen

# jQuery UI

We weten inmiddels hoe de basis van jQuery werkt. We kunnen elementen aanpassen en al wat gegevens ophalen van een server. Nu wordt het tijd voor het echte werk.

We gaan controls gebruiken die op die in de browser draaien. We willen tenslotte het gevoel krijgen alsof we met een echte windows applicatie aan het werken zijn.

We beginnen ook hier weer met de basis. Wat is een dedicated Windows applicatie zonder Error box???

## Scripts

jQuery heeft al een heleboel werk voor je gedaan en biedt daarom al verschillende controls aan. Hiervoor moeten we natuurlijk wel eerst een javascript file en wat css inlezen.

Laad het script en css in door binnen @section Head de volgende regels op te nemen:

|  |
| --- |
| <link href="@Url.Content("~/Content/themes/base/jquery.ui.all.css")"rel="stylesheet" type="text/css" />  <script src="@Url.Content("~/Scripts/jquery-ui-1.8.16.min.js")" type="text/javascript"></script> |

We hebben nu de jQuery UI library en de bijbehorende CSS ingelezen. Nu kunnen we echt aan de slag!

In de code zag je al dat we het javascript “jQueryUI.js” al hadden toegevoegd. Hier gaan wij zo meteen in werken. Bekijk eens de eerste paar regels, waar we het starten van het script iets verkort opschrijven.

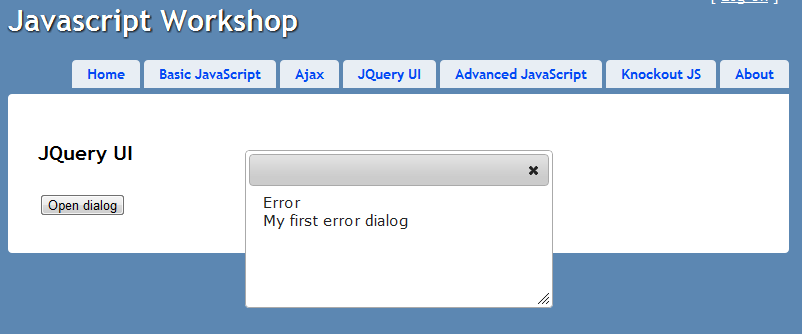
## Dialog

Ok, laten we nu eens onze pagina interactief en mooi gaan maken!  
We gaan een <div> element omtoveren tot een dialog-box die je op je scherm gaat zien.

De html is al voorbereid zoals je had gezien.   
Ga nu naar het javascript bestand (jQueryUI.js) en voeg aan de functie: ExecuteOnStartup de volgende regel toe:

|  |
| --- |
| $('#myDialog').dialog(); |

  
**Start de applicatie en bekijk het resultaat:**



Dankzij de UI library van jQuery hebben we nu een standaard dialog plugin tot onze beschikking, die van een element op je pagina een dialog maakt. Dit alles door alleen de dialog() functie aan te roepen.

Maar je zag al dat het dialog erg kaal was. Zo had het dialog geen titel en misten we de “Ok” knop voor onze foutmelding te sluiten. En het dialog ging automatisch open, dit willen we natuurlijk zelf doen.

Met andere woorden, we willen onze dialog configureren.  
Maak een config object aan en geef deze mee aan de initialisatie van het dialog.  
Dit doe je door de volgende code boven de intialisatie van het dialog te zetten:

|  |
| --- |
| var config = {          autoOpen: false,          resizable: false,          draggable: false,          modal: true,  height: 500,        title: 'My dialog title',          buttons: { Ok: function () { $(this).dialog("close"); } }      }; |

  
We hebben nu een config object aangemaakt met een aantal properties die allemaal redelijk voor zichzelf spreken. De property “buttons” bevat weer een nieuw object met properties. Als je gaat debuggen zul je dit ook zien. In het geval van de “ok” button hebben we er geen primitieve waarde maar een hele functie aan toegekend. Deze wordt uitgevoerd zodra op de “ok” button wordt geklikt.  
  
We gaan nu eerst de config meegeven aan de initialisatie van het dialog en we koppelen het openen van het dialog aan de button click van de button die op het scherm staat.

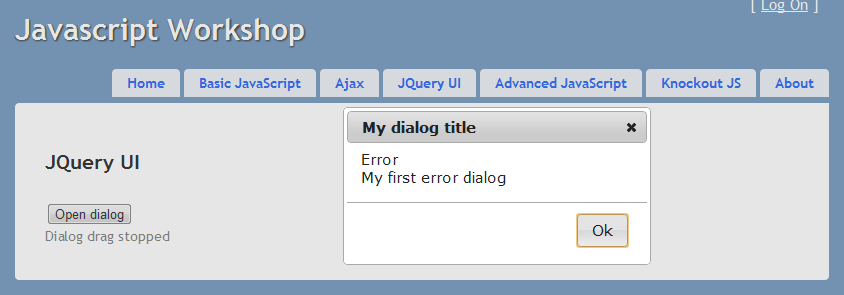
|  |
| --- |
| $('#myDialog').dialog(config); $('#openDialogButton').click(function () { $('#myDialog').dialog('open'); }); |

  
Een dialog genereert ook verschillende events. Hier kun je je op abonneren en daar je voordeel uit halen. We gaan in dit geval de status van het dialog laten zien op de pagina net onder de button.

Voeg nu de volgende regels code toe om de events af te vangen:

|  |
| --- |
| $('#myDialog').bind('dialogopen', function (){ $('#dialogState').html('Dialog opened'); });  $('#myDialog').bind('dialogdragstart',function() { $('#dialogState').html('Dialog drag started'); });  $('#myDialog').bind('dialogdragstop', function() { $('#dialogState').html('Dialog drag stopped'); });  $('#myDialog').bind('dialogclose', function () { $('#dialogState').html('Dialog closed'); }); |

  
 **Start de applicatie en bekijk het resultaat:**

  
  
**Opschoon tips:**  
Nu abonneren we ons na de initialisatie pas op de events, maar zoals je bij de ok button al had gezien kun je het ook gewoon in de config opnemen. De configuratie zou er dan als volgt uit komen te zien:

|  |
| --- |
| var config = {          autoOpen: false,          resizable: false,          draggable: true,          modal: true,          title: 'My dialog title',          buttons: { Ok: function () { $(this).dialog("close"); } },          open: function () { $('#dialogState').html('Dialog opened'); },          close: function () { $('#dialogState').html('Dialog closed'); },          dragStart: function () { $('#dialogState').html('Dialog drag started'); },          dragStop: function () { $('#dialogState').html('Dialog drag stopped'); }      }; |

**Extra info:**  
Voor een mooi overzicht van alle functionaliteit zie: <http://jqueryui.com/demos/dialog/>

## Datepicker

Nu we hebben gezien hoe eenvoudig het is om een dialog te tonen gaan we nog eens kijken wat jQuery UI nog meer voor ons in petto heeft?

We gaan nu aan ons dialog een Datepicker toevoegen. Hiervoor moeten we een stukje html toevoegen en een extra regel javascript.

Maak de inhoud van de <div> met id:”myDialog” leeg.  
Zet daar een input element van het type tekst in met het id: “myDate”

|  |
| --- |
| <input type="text" id="myDate" /> |

Voeg onderaan de ExecuteOnStartup de initialisatie van de Datepicker toe:

|  |
| --- |
| $('#myDate').datepicker(); |

  
**Start de applicatie, open de dialog en zie het resultaat:**



Dus door letterlijk 1 regel code toe te voegen kunnen we van een tekstvak een Datepicker maken. Ook hier geldt hetzelfde als voor het dialog. We kunnen de Datepicker helemaal configureren door options mee te geven of ons te abonneren op een event.

Als je een datum selecteert zie je dat hij niet de standaard Nederlandse notatie gebruikt. Verder zagen we dat de kalender standaard op zondag begint. Dit gaan we her-configureren.

We gaan dus nu de firstDay en de datum notatie wijzigen. Om onnodige regels code te gebruiken gaan we nu het object inline aanmaken en vullen. We kunnen dan de initialisatie die we zojuist hadden toegevoegd aan het javascript bestand vervangen met de volgende code:

|  |
| --- |
| $('#myDate').datepicker({ firstDay: 1, dateFormat: 'dd-mm-yy' }); |

  
**Start de applicatie en bekijk het resultaat:**



Ook voor de Datepicker geldt dat het veel meer mogelijkheden heeft. Daarvoor kun je alle demo’s bekijken en technische documentatie raadplegen op: <http://jqueryui.com/demos/datepicker/>

## Buttons

Inleiding tot buttons

### Checkbox buttons

### Radio buttons

### Group buttons

## Animations / effects

# Advanced Javascript

## Create your own methods

Van <http://sandervandevelde.wordpress.com/2011/06/15/schrijf-eens-je-eigen-jquery-functies/>

Zie een pagina met html:

// this code contains a bug

<html>

<head>

<script type="text/javascript"

src="http://ajax.googleapis.com/ajax

/libs/jquery/1.6.1/jquery.min.js"></script>

</head>

<body>

<a id="nupuntnl"

href="http://www.nu.nl">Nu</a>

<script type="text/javascript">

$(document).ready(function() {

$('#nuputnl').text('Public link');

});

</script>

</body>

</html>

De .text manipuleert geen enkel object.

Dus we schrijven een functie die piept als nul of meerdere objecten gevonden zijn:

(function( $ ){

$.fn.onlyOne = function(mode) {

var mode = mode || 'off';

if (mode != 'off') {

if (this.length == 1) {

if (mode != 'on') {

alert('Information: one object found.' );

}

}

else {

alert('Warning: not one but '

+ this.length + ' objects expected.');

}

}

// do not break the normal flow;

return $(this);

};

})( jQuery );

<script type="text/javascript">

$(document).ready(function() {

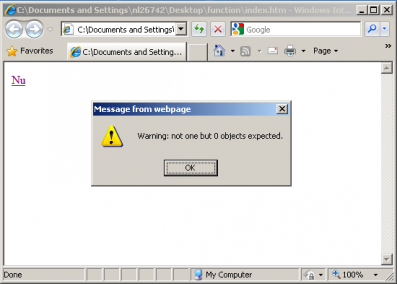
$( nuputnl').<strong>onlyOne('on')</strong>

.text('Public link');

});

</script>

Dit geeft dan namelijk :



Extra opdracht: bouw een functie die verwacht dat minstens één object wordt gevonden.

## Create a javascript object

<http://marcofranssen.nl/2011/09/17/jquery-events-contributes-to-clean-javascript/>

## Communicate between objects with events

<http://marcofranssen.nl/2011/09/17/jquery-events-contributes-to-clean-javascript/>

# MVVM met Knockout JS

Combineer

<http://sandervandevelde.wordpress.com/2011/09/23/integratie-van-knockout-in-asp-net-mvc3/>

en

<http://sandervandevelde.wordpress.com/2011/09/26/schrijf-je-in-voor-events-in-knockoutjs/>

extra opdracht:

## Simple bindings

## Advanced bindings

## Cascading dropdown

In deze oefening gaan we een cascading dropdown menu maken. Een cascading dropdown is een reeks van dropdowns die op elkaar reageren.

Voorbeeld: We selecteren een land in de eerste dropdown. Op basis van de geselecteerde waarde willen van een land de provincies ophalen en deze in de 2e dropdown tonen etc.

Op de server zijn al een aantal urls voorbereid waarmee we de verschillende dropdowns kunnen vullen aan de hand van de json response.

<http://localhost:56502/KnockoutJS/CascadingDropdown/Countries/>

<http://localhost:56502/KnockoutJS/CascadingDropdown/States/>

<http://localhost:56502/KnockoutJS/CascadingDropdown/Cities/>

We beginnen met het opbouwen van onze UI.

|  |
| --- |
| <select data-bind="options: countries, optionsCaption: 'Choose country...', optionsValue: function(item) { return item.value; }, optionsText: function(item) { return item.text; }, value: countrySelect, valueUpdate: 'change'" id="Country" name="Country"></select>  <select data-bind="options: states, optionsCaption: 'Choose state...', optionsValue: function(item) { return item.value; }, optionsText: function(item) { return item.text; }, value: stateSelect, valueUpdate: 'change'" id="State" name="State"></select>  <select data-bind="options: cities, optionsCaption: 'Choose city...', optionsValue: function(item) { return item.value; }, optionsText: function(item) { return item.text; }, value: city, valueUpdate: 'change'" id="City" name="City"></select>  <span data-bind="text: result"></span> |

In de UI hebben we een 3-tal dropdowns toegevoegd die gebind worden aan waardes uit het viewmodel. Tevens hebben we een span waar het resultaat van onze selectie in willen gaan tonen.

Om de data voor onze bindings te kunnen aanbieden hebben we een viewmodel nodig. Dit definiëren we in javascript.

|  |
| --- |
| <script>          var viewModel = {              country: ko.observable(),              countries: ko.observableArray(),              state: ko.observable(),              states: ko.observableArray(),              city: ko.observable(),              cities: ko.observableArray(),              result: ko.observable()          };      </script> |

Zoals je ziet komen de namen van de properties in ons viewmodel overeen met de bindings die we eerder in de UI hebben gedefiniëerd. Het enige wat ons nu nog te doen staat is deze properties vullen met data van de server.

We beginnen met het vullen van de Country dropdown. Dit doen we door een call te doen naar de server en onze countries property te vullen met de response van de server. Verder applyen we de bindings op ons viewModel.

|  |
| --- |
| $(function () {              $.getJSON('http://localhost:56502/KnockoutJS/CascadingDropdown/Countries/', null, function (response) {                  viewModel.countries(response);              });              ko.applyBindings(viewModel);          }); |

Als je nu je pagina ververst zul je zien dat de countries dropdown is gevuld met de waarden van de server.

Nu kunnen we ervoor gaan zorgen dat de verschillende dropdowns op elkaar gaan reageren. Hiervoor voegen we een stukje code aan ons viewmodel toe. We beginnen met het vullen van de states dropdown op het moment dat een land geselecteerd is.

|  |
| --- |
| viewModel.countrySelect = ko.dependentObservable({              read: viewModel.country,              write: function (country) {                  this.country(country);                  $.getJSON('http://localhost:56502/KnockoutJS/CascadingDropdown/States/' + country.value, null, function (response) {                      viewModel.states(response);                  });              },              owner: viewModel          }); |

Hetzelfde kunnen we doen voor de steden op basis van een selectie van een state. Probeer nu zelf op basis van je UI de juiste property te implementeren op je viewmodel. Maak gebruik van de volgende server url (vergeet je parameter niet toe te voegen aan het einde van de url).

<http://localhost:56502/KnockoutJS/CascadingDropdown/Cities/>

Tip: Bestudeer de properties in de UI die je gebruikt voor de binding.

Last but not least willen we het resultaat van onze selectie nog tonen in de span die we binden aan de property result.

|  |
| --- |
| viewModel.result = ko.dependentObservable(function () {     var result = '';     result += this.country() != undefined ? 'Country: ' + this.country().text + ', ' : '';     result += this.state() != undefined ? 'State: ' + this.state().text + ', ' : '';     result += this.city() != undefined ? 'City: ' + this.city().text : '';     return result;  }, viewModel); |

Nu zul je zien dat na het selecteren van een waarde in de dropdowns de waardes netjes getoond worden in de span die gebind is aan je result property op je viewmodel.

## Animations with jQueryUI

# References

<http://marcofranssen.nl>

<http://sandervandevelde.wordpress.com/>

<http://addyosmani.com>

<http://jquery.com>

<http://jqueryui.com>

<http://knockoutjs.com>

<http://github.com/atosorigin/javascriptWorkshop>

1. Communicatie tussen verschillende websites wordt als een beveiligingsrisico gezien. Er is wel een mogelijk in de vorm van JsonP. Deze geeft de mogelijkheid om de aanvragen een callback functie op te geven. De opgevraagde site zal zijn antwoord dan via die callback functie doorgeven. Dit valt buiten scope voor deze workshop. [↑](#footnote-ref-1)