JQUERY 2.0.2
UI 1.10

Inhoudsopgave

1 jQuery introductie

1.1 Versie

Deze cursus is gebaseerd op jQuery versie 2.0.2 of 1.10.1 (die identiek zijn, enkel zonder/met <IE8 ondersteuning) en de UI 1.10.

Moet je nog steeds oudere browsers ondersteunen, gebruik dan JQ 1.10.1.

1.2 Wat is jQuery?

jQuery is een gratis, open-source Javascript library die het gebruik van Javascript vereenvoudigt. Deze library werd voor het eerst gepubliceerd in januari 2006 door **John Resig**.

*Voor de front-end developer is kunnen werken met jQuery een **must**!*

Er zijn nog heel wat andere Javascript libraries, maar we kiezen voor jQuery omdat die heel versatiel is, frequent ge-update wordt en erg veel gebruikt wordt.

Daarnaast zijn er zeer veel populaire **plug-ins** die *dependent* (afhankelijk) zijn van jQuery, m.a.w. jQuery moet eerst gekoppeld zijn om de plug-in te doen werken.

JQ maakt het ook veel gemakkelijker om Ajax applicaties te maken: interactieve applicaties tussen client- en serverside.

Dit zijn de sterke puntenvan jQuery: ·

- DOM selectie
- CSS manipulatie
- Chaining
- Ajax calls
- browserverschillen oplossen
- UI library met veel effecten en widgets
- talloze populaire plug-ins

Maar vergis je niet!

jQuery vervangt Javascript niet, het is een laag erbovenop!
Je hebt minstens een goede basiskennis van Javascript nodig om efficiënt met jQuery te kunnen werken.

Je moet nog steeds Javascript en DOM goed onder de knie hebben om vlot JQ scripts te schrijven.

2 De cursus

2.1 Over deze handleiding

In tegenstelling tot de Javascript cursus maken we in deze handleiding geen onderscheid tussen theorie en praktijk. Dat betekent soms wat meer tekst tijdens het verwerken van de projecten.Een opsomming van *JQ Selectors* vind je in bijlage.

2.2 Conventies

In deze cursus wordt de volgende opmaak gehanteerd:

- lees **JS** als *Javascript*, de taal, en **JQ** als *jQuery*, de library
- Javascript objecten, properties en methods worden in een monotype font geschreven, vb. `document.getElementById()`.
- lees `div#content` als het `div` element met de `id` “content”
- lees `div.keuzevakjes` als een `div` element met de `class` “keuzevakjes”
- Grotere stukken code worden omkaderd:

```
var getal = Number('3.5') // returnt 3.5
var probleemgetal = Number('3.5 witte muizen') // returnt NaN
```

- Bestandsnamen (*cookieBank.html*), eigen variabelen (*getal*) en Engelstalige begrippen (*closure*) worden in schuinschrift gezet.
- Het teken ⮵ wordt gebruikt om aan te duiden dat de volgende regel volgt op de huidige en dus één statement vormt. Door ruimtegebrek is het niet altijd mogelijk lange statements op één lijn af te drukken.

  Bijvoorbeeld:

```
if (nieuwSaldo<=0){
                    strBericht = "Uw saldo is onvoldoende om dit ⮵
                                  bedrag af te halen. ";
                    strBericht += "U kunt maximaal " + eval(saldo-1) + ⮵
                                  " Euro afhalen.";
                    waarschuwing(strBericht);
                    bedragVeld.value=saldo-1;
                    bedragVeld.focus()
    }
```

  Hierin moet je de eerste twee lijnen van de `if` schrijven als

```
strBericht = "Uw saldo is onvoldoende om dit bedrag af te halen. ";
strBericht += "U kunt maximaal " + eval(saldo-1) + " Euro afhalen.";
```

- Een **tip** dient om je aandacht te trekken op een nuttige levenswijsheid, veelvoorkomend probleem, een handige oplossing:

"beter een diamant met een foutje, dan een kei zonder gebreken"(*Confusius*)

2.3 Voorkennis

Sommige onderdelen van jQuery – vooral de UI widgets – vergen geen diepgaande Javascript kennis: met een minimum aan copy-paste code, krijg je zoiets in gang. Maar om zelf jQuery scripts te kunnen schrijven, laat staan plug-ins, moet je een degelijke Javascript kennis hebben.

Daarom verwachten we dat je devolgende modules achter de rug hebt:

- HTML-CSS
- *Javascript PF*

2.4 De topics

We volgen een **pragmatische aanpak**.

Een aantal veel voorkomende scripts en situaties doen we opnieuw met JQ. We overlopen ook de meest populaire UI widgets.

2.5 Software

De voorbeeldwebsite gebruikt HTML5, dat kan vandaag geen probleem meer opleveren.

Gebruik je vertrouwde editor (zelfde als voor JS) en verder de webtools die bij een browser horen:

Installeer bij *FireFox* **Firebug**, bij *IE8* de **developer tools**, bij *Chrome* de **developer tools**.

3 jQuery projecten

3.1 de voorbeeldapplicatie

Voor alle projecten in deze cursus gebruiken we een *kant-en-klare* website **'De Plantenshop**'.

Deze voorbeeldwebsite is een PHP applicatie die gekoppeld is aan een MySQL databank. Alle HTML, CSS en PHP is volledig. Je hoeft ook geen PHP of MySQL te kennen. De website gebruikt de HTML5 standaard.

Wij gebruiken deze applicatie om er client-side jQuery scripts op toe te passen, net zoals het er in een bedrijf aan toe gaat.

Als je problemen hebt of als je de procedure niet begrijpt, vraag assistentie aan je coach.

Wat heb je nodig?

- de oefenbestanden in *plantenshop.zip*
- een webserver die PHP ondersteunt (zoals een XAMPP omgeving)
- een connectie met de *MySQL* database "plantenshop"

Wat doe je?

- ☞ *unzip* het bestand *plantenshop.zip* in de root van je webserver (bv. je *localhost*) of in een map of op de webruimte die je kreeg van je coach.

  Het resultaat is een map *plantenshop_2013_basis* die alles bevat
- ☞ hernoem deze map naar *plantenshop*
- ☞ surf **via je webserver** naar *plantenshop/index.php*
  Bijvoorbeeld

  http://localhost/plantenshop/index.php
  of
  http://www.ict.teno.be/oostende/mark_decursist/oefeningen/plantenshop/index.php

  Via je bestandssysteem*file://*ernaartoe gaan, zoals,

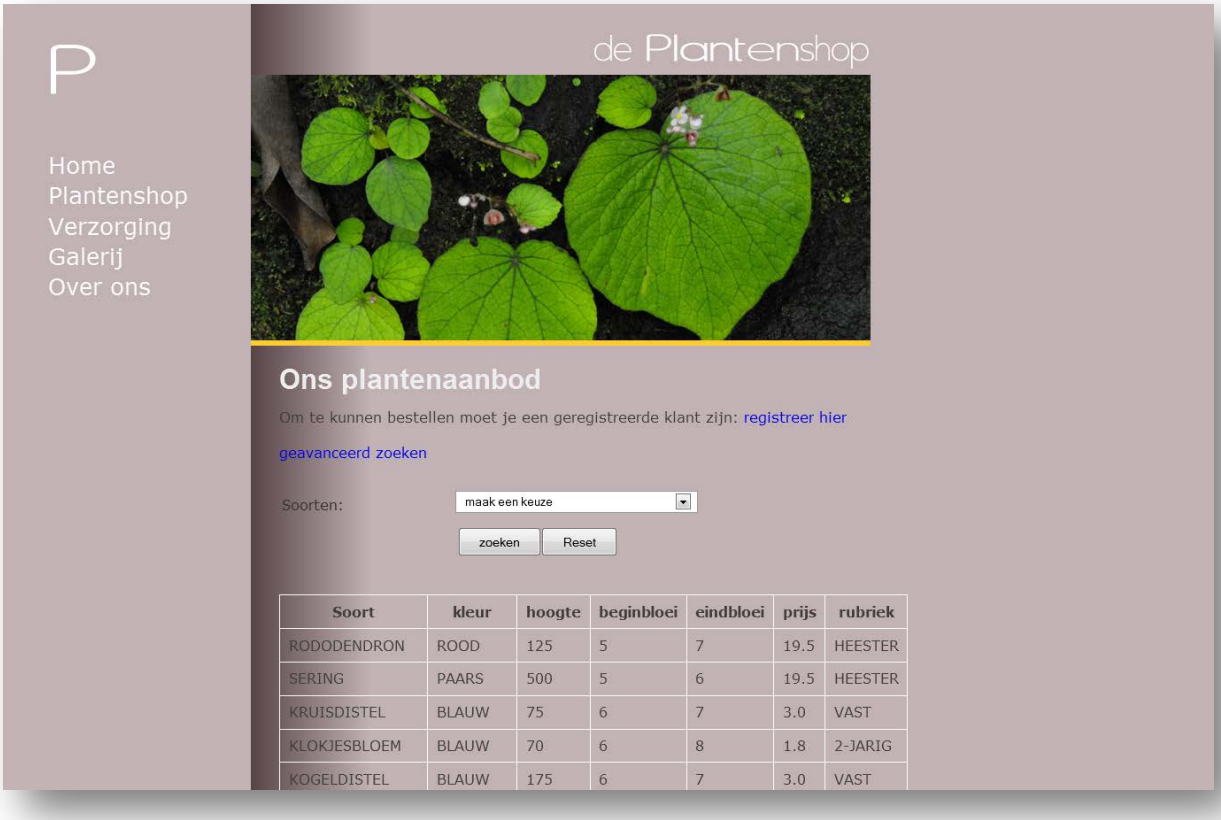
  c:/plantenshop/index.php

  zal niet lukken! Je moet via *http* werken

Nu moet je het volgende zien:

☞ test nu ook even de werking van de databaseconnectie:
klik op het menu-item *Plantenshop*

deze pagina moet er zo uit zien:



Onder de formuliervelden is een tabel te zien met planten.

De kans is klein dat je dit effectief ziet want deze pagina is gekoppeld aan de database *plantenshop*, waarvan de verbinding waarschijnlijk nog niet in orde is.

☞ **Wat moet je nagaan?**

? surf je via een webserver? dus *http://....*

? ondersteunt de webserver PHP?

? heb je de database geïnstalleerd op je MySql server?

? heb je een gebruiker aangemaakt die SELECT, INSERT, UPDATE rechten heeft op deze database?

Vraag eventueel assistentie aan je coach om de juiste connectiegegevens naar de database te helpen aanpassen en te controleren in het bestand/*DATA/DBCONFIG.PHP*:

hostname/username/password/dbName

3.2 jQuery gebruiken

Je kan jQuery in je website gebruiken op twee manieren:

- met een **lokale kopie van jQ**

  In dit geval moet je jQ **downloaden** en installeren in je website.

  voordelen: volledige controle, versie controle, beter tijdens ontwikkeling

  nadelen: trager, meer verkeer naar eigen site (betalend?)

- via een **CDN** (*Content Delivery Network*): dit is rechtstreekse koppeling naar de library op het internet.

Je hoeft jQ niet te downloaden, je koppelt gewoon naar een URL op het net.

voordelen: parallele http requests betekent grotere snelheid, CDN's caching, minder http verkeer naar je eigen hosting site, auto koppeling naar meest recentste versie mogelijk

nadelen: internet verbinding noodzakelijk, versie controle (plugins?)

Tijdens deze cursus gebruiken we een lokale kopie, maar lees nadien aandachtig de bijlage '*jQuery via een CDN*'.

3.2.1 Downloaden

Je kunt de meest recente versie van de library downloaden van *jquery.com*.

Die is beschikbaar in drie versies:

- *uncompressed*: normaal leesbaar
- *minified*: alle spaties en lijneinden zijn verwijderd, onleesbaar
- *packed*: *minified* en gecomprimeerd, onleesbaar

Als je niet van plan bent de library te lezen of er een stukje bij te schrijven, neem dan steeds de *minified* versie, die is de snelste in uitvoering.

Zo ziet een *minified* versie er uit:

Misschien een idee voor behangpapier in je stamcafé?

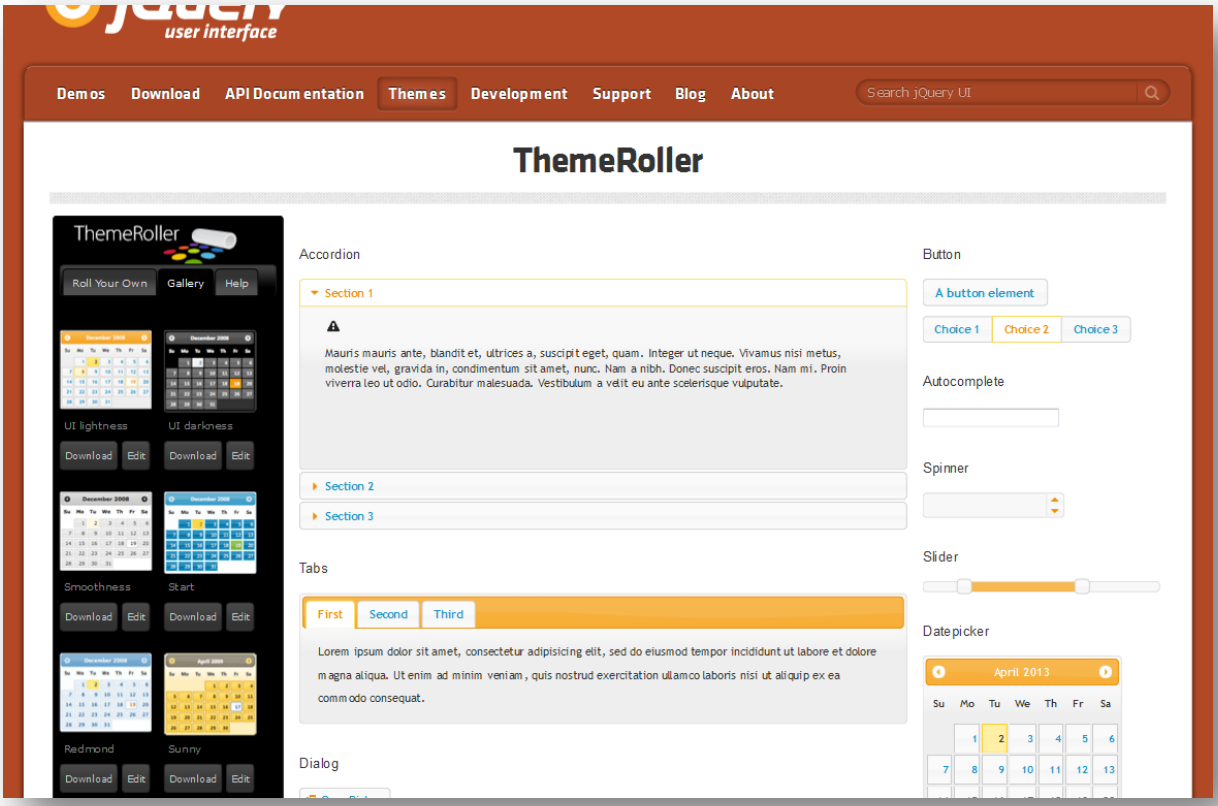
Dit is de ***core library*** die alle kernfuncties bevat samen met **enkele** *User Interface* (UI)*widgets*. Er zijn echter veel meer **UI** widgets die apart in een **UI library** zitten.

We zullen nu de *UI library* samen met de *core library* downloaden en ze beide installeren. De UI library is volledig afhankelijk van de core library en kan dus onmogelijk apart werken.

Doe het volgende:

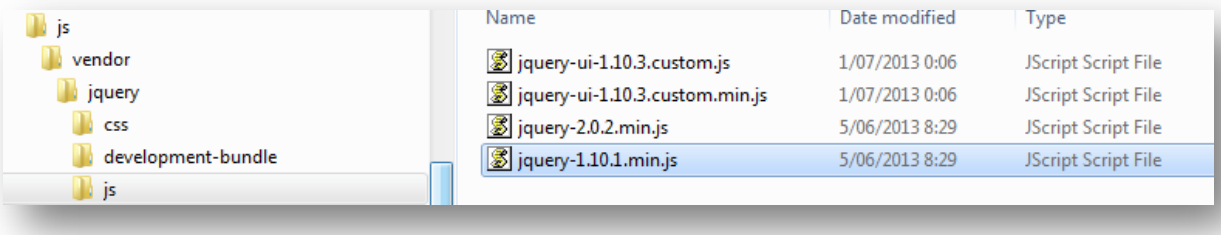
☞ ga naar *http://jqueryui.com*

- klik de link *Themes* en klik bij *Themeroller* op *Gallery*
- kies in de linkerkolom een *theme*
  (wij kiezen het standaard theme "*UI lightness*"



- de knop *Edit* onder elk *theme* brengt je op het tabblad "*Roll your own*" waar je elk aspect van een theme kunt aanpassen. Je kunt hiermee dus je eigen *themes* samenstellen
- kies een *theme* en klik op de *Download* knop er onder.
- op de volgende pagina (*Download builder*) kan je onderdelen kiezen. Kies alles en klik op *Download*
- sla het zip bestand op
- unzip het bestand in de map *PLANTENSHOP/JS/VENDOR/JQUERY*

dan heb je nu de volgende mappenstructuur:



- De hoofdmap JS zal alle eigen javascripts bevatten, terwijl de map VENDOR bedoeld is voor alle *third party* scripts: niet zelf geschreven scripts, waarvan jquery er één is
- De map *JQUERY/JS* bevat de laatste versies van de jQ libraries, hier*:*
  - o *jquery-1.10.1.js* en
  - o *jquery-ui-1.10.3.custom.min.js.*

    Opmerking*: als je deze cursus doorneemt, kan er al een nieuwe versie van jQuery of jQuery UI verschenen zijn, kies eventueel de nieuwste versie.*
    Je kan ook kiezen om jquery-2.0.x.js te gebruiken als je <IE8 niet moet ondersteunen
- De map *CSS* bevat in een submap het gedownloadde theme:*ui-lightness*. Als je andere themes downloadt moet je deze maar ook in de map css zetten
- de map *DEVELOPMENT-BUNDLE* bevat demo's en uitleg van de verschillende effecten en extra bestanden. Deze map is niet essentieel en moet op een productieserver verwijderd worden.
  Ze is voor jou momenteel misschien wel nuttig als je wat voorbeelden wil bekijken.

3.3 Kennismaking met jQuery

We maken onze eerste jQuery scripts.

3.3.1 jQuery koppeling

De *'Plantenshop'* website maakt gebruik van *meerlagenarchitectuur (MVC)* om de presentatie te scheiden van de andere functionaliteiten. Alle moderne CMS'en werken op een dergelijke manier, alleen wat ingewikkelder dan hier. Deze website gebruikt ook een eenvoudig *template* syteem en haalt verschillende onderdelen op via PHP functies. Vraag gerust wat uitleg aan je coach over hoe er tewerk gegaan wordt.

Zo zullen we ook de koppelingen naar stylesheets en jQuery libraries op één plaats zetten. Het systeem zorgt er dan voor dat alle pagina's die koppelingen krijgen.

Zoek het bestand *head.tpl* in de map *pres*:

```
<!DOCTYPE HTML>
<!--[if lt IE 7]><html class="no-js lt-ie9 lt-ie8 lt-ie7"><![endif]-->
<!--[if IE 7]><html class="no-js lt-ie9 lt-ie8"><![endif]-->
<!--[if IE 8]><html class="no-js lt-ie9"><![endif]-->
<!--[if gt IE 8]><!-->
<html class="no-js">
<!--<![endif]-->
<head>
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
<title>{$title}</title>
<meta name="description" content="de Plantenshop is een oefensite van de vdab IT
  opleidingen om jQuery functionaliteit op uit te testen. In deze plantenshop kunt u
  geen planten kopen of bestellen.">
<meta name="keywords" content="vdab opleidingen jQuery tutorial">
<meta name="viewport" content="width=device-width">


<!-- stylesheets voor alle pagina's via vaste LINK elementen-->
<link rel='stylesheet' type='text/css' href='css/plantenshop.css'  />


<!-- pagina-gebondenstylesheets -->
{$paginaStylesheets}


<!-- algemene JS scripts voor alle pagina's via vaste SCRIPT elementen-->
<script src="js/vendor/modernizr-2.6.2.min.js"></script>


<!-- pagina-gebonden JS scripts-->
{$paginaScripts}
```

```
</head>
```

Dit template bevat het bovenstuk van alle pagina's in de website. In dit bestand zijn `{$variabele}` variabelen te zien die ingevuld worden door de controller *index.php*.

Het template is gebaseerd op de HTML5 boilerplate, maar wij zullen alles betreffende jQuery er zelf in plaatsen terwijl dat in het boilerplate reeds gebeurd is. Het template bevat ook reeds een versie van *Modernizr*, die we echter niet zullen gebruiken. Zoals je weet is Modernizr een javascript dat niet afhankelijk is van jQuery.

Er zijn 4 ruimtes voorzien voor stylesheets en scripts:

- **vaste stylesheets** die op alle pagina's aanwezig zijn: hier is reeds een koppeling naar *plantenshop.css*
- **pagina-gebonden stylesheets** die slechts op één bepaalde pagina aanwezig zullen zijn: worden ingevoegd door de controller via de variabele `{$paginaStylesheets}`
- **vaste, algemene scripts** die op alle pagina's aanwezig zijn: hier is reeds een koppeling naar *Modernizr* aanwezig
- **pagina-gebonden scripts** die enkel op één bepaalde pagina zullen aanwezig zijn: worden ingevoegd door de controller via de variabele {`$paginaScripts}`

Nu voegen we de nodige koppelingen in om jQuery te doen werken:
2 jQ libraries en één jQ stylesheet.

☞ Voeg deze elementen toe:

```
<!DOCTYPE HTML>
<!--[if lt IE 7]><html class="no-js lt-ie9 lt-ie8 lt-ie7"><![endif]-->
<!--[if IE 7]><html class="no-js lt-ie9 lt-ie8"><![endif]-->
<!--[if IE 8]><html class="no-js lt-ie9"><![endif]-->
<!--[if gt IE 8]><!-->
<html class="no-js">
<!--<![endif]-->
<head>
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
<title>{$title}</title>
<meta name="description" content="de Plantenshop is een oefensite van de vdab IT
  opleidingen om jQuery functionaliteit op uit te testen. In deze plantenshop kunt u
  geen planten kopen of bestellen.">
<meta name="keywords" content="vdab opleidingen jQuery tutorial">
<meta name="viewport" content="width=device-width">


<!-- stylesheets voor alle pagina's via vaste LINK elementen-->
<link rel='stylesheet' type='text/css' href='css/plantenshop.css'  />
<link rel='stylesheet' type='text/css' href='js/vendor/jquery/css/ui-
  lightness/jquery-ui-1.10.3.custom.min.css'  />


<!-- pagina-gebondenstylesheets -->
```

```
{$paginaStylesheets}

<!-- algemene JS scripts voor alle pagina's via vaste SCRIPT elementen-->
<script src="js/vendor/modernizr-2.6.2.min.js"></script>
<script src="js/vendor/jquery/js/jquery-1.10.1.js"></script>
<script src="js/vendor/jquery/js/jquery-ui-1.10.3.custom.min.js"></script>

<!-- pagina-gebonden JS scripts-->
{$paginaScripts}
</head>
```

Bespreking:

- let er op dat het *pad* altijd vertrekt van de *root* van de website
- het ***themestylesheet***:
  - hier *jquery-ui-1.10.3.custom.min.css*
    We gebruiken bij voorkeur de *minimized* versie (alle witruimte eruit gehaald) voor snelheid
  - als je meerdere themes downloadt, kan je wisselen van theme door naar een andere theme-map te verwijzen,
    vb. *'js/vendor/jquery/css/**blitzer**/jquery-ui-1.10.3.custom.min.css'*
  - Het is ook mogelijk een theme op een andere manier te downloaden waarbij het woordje *custom* afwezig is
- de **corelibrary**:
  - *jquery-1.10.1.js* moet als **eerste** gekoppeld worden
  - in onze *custom download* zat een niet gecomprimeerde versie van jQuery.
    Download je die apart dan kan je de minimized versie downloaden en gebruiken: *jquery-1.10.1.min.js*
- de **UI library:**
  - *jquery-ui-1.10.3.custom.min.js*

Nu bevatten alle pagina's in de applicatie deze koppelingen, dat kan je controleren op meer dan één manier. Surf opnieuw naar één van de pagina's in de plantenshop of ververs de huidige pagina:

- ☞ kies dan in je browser "*View source*"
- ☞ of klik in een webdeveloper tool zoals **FireBug** de ***script*** tab en de ***CSS*** tab aan. Ook de network tab van je developer tool toont het laden van de libraries

In beide gevallen moet je de code van de gekoppelde JQ libraries zien.

3.3.2 De jQuery function

De jQuery library steunt op de `jQuery()` **factory function.**
Deze functie is een soort “Zwitsers mes” die, afhankelijk van zijn parameters, heel veel mogelijkheden biedt.
Deze function kan je op twee manieren schrijven:

- `jQuery()`
- `$()`

De `$()` function kan je gebruiken

- om DOM elementen in te 'verpakken' en er dan JQ methods op toe te passen
- om algemene JQ methods te gebruiken die niet op DOM elementen van toepassing zijn, bijvoorbeeld het lussen doorheen arrays en objecten
- om een script te starten als de DOM geladen is

3.3.3 document ready

JQ maakt het ons veel makkelijker om code te scheiden van HTML structuur: *unobtrusive JS*.
Ook hier staan we voor hetzelfde probleem als voorheen: we moeten namelijk **wachten tot de volledige DOM tree ingeladen is** vooraleer we een script kunnen starten dat werkt op een HTML-element.
In JS zouden we daar het `window.onload` event voor gebruiken, in JQ doen we hetzelfde met de `$(document).ready` handler.

☞ In de controller *index.php*, zoek in de `switch` structuur de `case` "about" en voeg de volgende lijn toe:

```
…
switch ($page){
        case "about":
                /*** About pagina ***/
                $tpl['title']         = "de Plantenshop: wie zijn we en wat doen we?";
                $tpl['body_id']       = "about";
                //content
                $tpl['rechts']        = getAbout();
                $tpl['paginaScripts'] = getScriptElements("js/about.js");
        break;
…
```

Hiermee maken we in deze pagina een koppeling naar het bestand *about.js*.
Andere pagina's zullen dit script niet hebben.

☞ Open het JS bestand *about.js* in de map *js*. Het bevat reeds een Javascript functie *walkTree()*. Laat deze staan, we maken er later gebruik van.
☞ Typ de volgende code in het bestand boven *walkTree()*:

```
// JavaScript Document
alert('dom tree nog niet geladen: onmiddellijke uitvoering');
$(document).ready(function(){
        alert('dom tree geladen: de id van het body element is: ' + $('body')[0].id);
  });//einde doc.ready
```

Bespreking:

- het eerste statement wordt uitgevoerd zodra het gelezen is, dus vooraleer de DOM gebouwd is, vooraleer images geladen zijn etc..
- De `$(document).ready()`event handler wordt pas uitgevoerd als de *DOM tree* geladen is, maar is beter dan `window.onload` want deze handler wacht niet tot alle andere resources geladen zijn (images, stylesheets, iframes…). Daardoor werkt hij sneller
- **binnen** de `ready()` functie voeren we een **anonieme functie** uit, waarin nu een tweede `alert` staat
- de code `$('body')[0].id` leggen we verder uit, het leest de `id` van het body element
- Je kan ook meerdere `$(document).ready` handlers in je script zetten, ze zullen uitvoeren in volgorde
- het is belangrijk dat je het einde van de `$(document).ready` aangeeft met commentaar: straks hebben we zoveel haakjes, dat het script onoverzichtelijk wordt

☞ We korten de code onmiddellijk in:

```
alert('dom tree nog niet geladen: onmiddellijke uitvoering');
$(function(){
  alert('dom tree geladen: de id van het body element is' + $('body')[0].id);
});//einde doc.ready
```

Bespreking:

- `$()` is hetzelfde als `$(document).ready()`

Het is dus belangrijk te bedenken op **welk moment** je een script wil uitvoeren:

- scripts die iets toepassen op een HTML-element, plaats je dus in de *document ready* functie. Doe je dat niet dan is de kans groot dat je een fout krijgt: het element is niet gevonden, want de DOM is nog niet volledig gelezen.
- scripts die geen gebruik maken van de aanwezige HTML, of waarvan je zeker bent dat ze pas later uitgevoerd worden kan je los in de script tag plaatsen. Bijvoorbeeld, het inladen van gegevens hoeft niet in `$()`. Ook een eventhandler functie (bv.voor het klikken op een knop) kan erbuiten gedefinieerd worden zolang je een geldige referentie kan leggen naar de knop: dit laatste zal dus wel in de `$()` moeten gebeuren.

3.3.4 Selecteren met de $() wrapper

Een andere toepassing van de `$()`*functie* is het **selecteren** van elementen (collection)**.** Dat gebeurt aan de hand van **CSS selectors.**

Zoals je onmiddellijk zult merken is dat veel eenvoudiger dan met de DOM methods `document.getElementById()` en `document.getElementsByTagName()`.
Het is vergelijkbaar met `document.querySelector()` en `document.querySelectorAll()`maar met toch een verschil.

☞ We proberen een eerste selectie: vervang de vorige code door het volgende:

```
$(function(){
  $('a').addClass('rood');
});//einde doc.ready
```

Alle hyperlinks in de pagina zijn rood van bij het begin.
Hoe gaat het in zijn werk:

- de `$('a')`returnt een "***wrapped set"*** met alle hyperlinks in de pagina
- de `a` in de functie is een **CSS selector**, identiek aan het **element** `a` die je in een stylesheet zou gebruiken:

  ```
  a { color: red }
  ```

- op die *wrapped set* passen we de **JQ method** `addClass()` toe die de bestaande CSS class 'rood' op de set toepast.
  Deze CSS class is reeds aanwezig in het stylesheet *plantenshop.css*.

  `addClass()`is een handige **method** die een *class* toevoegt aan alle items in de set (zonder de bestaande classes te overschrijven).

De selectie `$('a')` noemt men een ***wrapped set*** en `$()`noemt men de ***jQuery wrapper***. De *wrapped set* `$('a')` bevat **alle** hyperlinks in de pagina.
Een wrapped set is

- **gelijk** aan de zuivere Javascript methods want het bevat hetzelfde als `document.getElementsByTagName('a')` en `document.querySelectorAll('a')`
- **niet gelijk** aan een zuivere DOM *collection* want:
  - je kan er jQuery methods (zoals `addClass()`) op toepassen
  - je kan er niet onmiddellijk standaard DOM methods op toepassen

Een wrapped set is een collectie DOM elementen in een jQuery jasje: het is een jQuery object.

Wil je een standaard DOM method/property toepassen op een *wrapped set* dan moet je er de DOM node uit halen. In ons voorbeeld zal

```
alert($('a').nodeName)
```

mislukken want een *wrapped set* heeft geen `nodeName`.

```
alert($('a')[0].nodeName)
```

daarentegen haalt het DOM element uit de wrapped set en daar kan je wel de `nodeName` van vragen: het eerste DOM element.

jQuery gebruikt zowel CSS2 als CSS3 selectors om een set te maken. Daarnaast heeft het nog enkele eigen selectors. We raden je aan er even de online documentatie op na te kijken:

http://api.jquery.com/category/selectors/

Enkele voorbeelden van de verschillende selectors vind je in Appendix A.

3.3.5 Selecteren in context

Hierboven gebruikten we de algemene selector *a* om alle `a` elementen te verzamelen. CSS laat ons echter toe verder te selecteren:

```
$(function(){
  $('#bedrijf a').addClass('rood');
});//einde doc.ready
```

In deze wrapped set zitten slechts de hyperlinks die zich binnen het section element met de `id` "bedrijf" bevinden: slechts ééntje wordt rood.

- ☞ Net hetzelfde kunnen we bekomen door de selector met een *context*argument te gebruiken:

```
$('a','#bedrijf').addClass('rood');
```

Het resultaat is identiek, maar in het eerste geval zoekt jQ de DOM af vanuit de *root*, terwijl in het tweede – met *context* argument – er eerst de context bepaald wordt en die wordt dan afgezocht.

Je vraagt je waarschijnlijk af wanneer het één en wanneer het ander te gebruiken. De tweede methode zal handiger zijn als we **dynamisch** de context moeten bepalen en we die dus niet vooraf kennen.

Een derde mogelijkheid is de method `find()`gebruiken. Die returnt alle descendants van een andere selector. Om opnieuw hetzelfde te bekomen zouden we schrijven:

```
$('#bedrijf').find('a').addClass('rood');
```

Gebruik bij voorkeur één van de eerste twee manieren, omdat ze sneller zijn.

Merk ook op dat we hier een method laten volgen op een andere method: dat noemen we *chaining*.

Tenslotte een FAQ:*hoe weten we of er uberhaupt elementen in de wrapped set zitten?*

Als geen elementen kunnen geselecteerd worden, returnt jQ een lege set. Een set heeft ook altijd een `length` property:

```
alert($('#bedrijf a').length);
```

Deze returnt een integer met het aantal elementen in de set; `0` indien leeg.

3.3.6 Chaining

Elke JQ method returnt altijd opnieuw de *set*: meestal is dat dezelfde beginset, soms kleiner (want gefilterd), soms groter, misschien ook wel leeg. Maar in essentie heb je na de uitvoering van een method weer een set.

Dat betekent dat je na het uitvoeren van een JQ method er onmiddellijk een tweede method op kan uitvoeren, enz..

Dus na `$('a').addClass('rood')` heb je opnieuw `$('a')`.

Dit laat ons toe meerdere methods na elkaar uit te voeren: dit noemt men **chaining**.

☞ We testen dit uit:

```
$(function(){
  $('a').addClass('rood').filter('a[target]').addClass('groen');
});
```

De hyperlinks worden nog steeds rood maar de hyperlinks bij 'Links' worden groen.

Verklaring:

- de `filter('a[target]')` method **vernauwt** de set tot die hyperlinks die een `target` attribuut hebben en voegt daar opnieuw een `class` aan toe.

☞ en we doen nog meer aan *chaining*:

```
$(function(){
  $('a')
        .addClass('rood')
        .filter('a[target]')
              .addClass('groen')
        .end()
        .addClass('onderlijnd');
});
```

Nu worden ook alle hyperlinks onderlijnd.

Verklaring:

- de `end()` method beëindigt de `filter()` method en returnt daarmee opnieuw de oorspronkelijke set, waarop dan nogmaals een nieuwe *class* toegevoegd wordt.
- Merk de **gesplitste schrijfwijze** op: Javascript laat zonder problemen het opsplitsen van de lijnen toe als de volgende lijn begint met een operator

☞ We besluiten ons experiment met *chaining* door een *property* van de *wrapped set* op te vragen in een `alert()`:

```
$(function(){
  alert(
  $('a').addClass('rood')
         .filter('a[target]').addClass('groen')
                .end().addClass('onderlijnd').length
  );
});
```

Je krijgt een getal te zien: het aantal hyperlinks in de oorspronkelijke wrapped set

Verklaring:

- de property `length` geeft het aantal items (nodes, elementen) in de set. Welke set? De set die gereturnt wordt door de laatste method, die dus identiek is aan de oorspronkelijke set.

☞ Je mag dit scriptje in commentaar zetten. Laat de *document.ready* function echter staan.

3.3.7 Alternerende rijen

In de "Over ons" pagina zit ook een tabel, die een *beetje* opgemaakt wordt vanuit het externe stylesheet. Met CSS3 zijn we in staat de rijen alternerend te kleuren, maar met JQ ook.

☞ Voeg de volgende twee lijnen code toe aan de `document.ready()` functie:

```
…
  $('tr:odd').addClass('oneven');
  $('tr:even').addClass('even');
…
```

Even en oneven rijen zijn verschillend gekleurd!

Dit is evengoed van toepassing op vaste als dynamische gegenereerde tabellen (PHP, Java).

Hoe werkt het:

- er staan twee `class` stylerules in het stylesheet: 'even' en 'oneven' die de background kleuren
- via de JQ selectors `:odd` en `:even` kunnen we de betreffende `tr` elementen selecteren
- met `addClass()` voegen we dynamisch de opmaak toe

Er is wel een probleem ontstaan: de *header*-rij werd ook gekleurd, die mag echter niet veranderen.

☞ we passen wat aan:

```
…
  $('tbody tr:odd').addClass('oneven');
  $('tbody tr:even').addClass('even');
…
```

Door de meer precieze *selector* is ons probleempje opgelost.

3.3.8 Een click Event

JQ maakt van het werken met *events* een kinderspelletje: vergeet de cross-browser problemen, JQ vangt ze op voor jou.

☞ voeg de volgende code toe:

```
$(function(){
…
  $('a[href^="http"]').click(
        function(){
              alert('U staat op het punt de pagina te verlaten');
        });
…
});
```

Bij een klik op een 'externe' hyperlink verschijnt nu een berichtje, bij een hyperlink naar een in-pagina *bladwijzer* gebeurt dat niet.

Verklaring:

- de selector `'a[href^="http"]'` selecteert alle hyperlinks waarvan de waarde van het `href` attribuut start met 'http', de 'externe' links dus in deze voorbeeldpagina.
- de `click()` method bindt een *event handler* aan het muisklik event
- In dit geval maken we gebruik van een anonieme functie om het berichtje te doen verschijnen

3.3.9 Algemene Event binding met on()

Voorgaande `click()` method is een directe manier om het Javascript click event te koppelen aan een element. Zo zijn er ook andere event methods: `mouseover()`, `submit()`, `resize()` en nog meer.

De method `on()` is een algemene method die **eender welk event** kan koppelen aan een elementenset. Zo kunnen we de vorige toepassing herschrijven met `on()`:

```
  $('a[href^="http"]').on('click',function(){
        alert('U staat op het punt de pagina te verlaten');
  });    …
```

De method `on()` gebruiken heeft een aantal voordelen:

- je meerdere event handlers koppelen voor hetzelfde event
- je kan uiteraard handlers koppelen voor andere events aan hetzelfde element
- je kan er *user-defined events* (zelf gedefinieerde events) mee koppelen, niet enkel Javascript events
- je kan een event handler gemakkelijk 'ontkoppelen' met `off()`
- je kan er *delegated event handlers* mee koppelen: event handlers die gekoppeld worden aan *toekomstige elementen* (die nu nog niet aanwezig zijn)

Delegated Events leggen we in een verder stadium uit.

3.3.10 Elementen aanmaken

Dezelfde `$()`*wrapper* kan ook gebruikt worden om **nieuwe elementen aan te maken**. De syntax hiervoor verschilt van *selecteren* doordat je er een **HTML-string** in plaatst, geen CSS selector:

```
$("<tag>inhoud</tag>")
```

Als voorbeeld willen we een *"terug naar boven"* hyperlink plaatsen voor elke kop in de tekst. Daarvoor voeg je de volgende code toe aan de *document.ready* function:

```
$(function(){
/* vorig script */
…
$('<a href="#about"title="terug nr boven">terug nr boven</a>')
        .insertBefore('h2, h3, h4, h5, h6');
  …
});
```

Nu bemerk je hyperlinks voor elke kop.

Verklaring:

- de bookmark `#about` is de `id` van de `body` tag van deze pagina. Die wordt voornamelijk gebruikt om pagina-eigen styles toe te passen, maar we kunnen hem evengoed gebruiken om weer naar boven te surfen
- de method `insertBefore()` voegt de nieuw aangemaakte HTML in vóór elk item van de *wrapped set*: merk op dat het onnodig is te itereren doorheen de elementen van de set!
- er bestaan verschillende **alternatieve methods** om nieuwe elementen in/na/voor toe te voegen:
    - de method `before()`bijvoorbeeld is identiek en verschilt enkel op gebied van de schrijfwijze: om hetzelfde te bekomen zouden we dan noteren:

      ```
      $('h2, h3, h4, h5, h6').before('<a href="#about">terug nr
      boven</a>');
      ```

De set die na de method gereturned wordt is dus anders:

- In het eerste geval is dat de ingevoegde hyperlink,
- in het tweede de set koppen.

Bekijk nueven de Appendix " *Overzicht methodes om inhoud in te voegen/verplaatsen/vervangen* ".

☞ We kunnen de selector ook wat korter schrijven:

```
$(function(){
  …
$('<a href="#about" title="terug nr boven">terug nr
  boven</a>').insertBefore(':header');
  …
});
```

De selector `:header` is hetzelfde als `h1,…h6.`

☞ Omdat ook de eerste paar headers op de pagina een link krijgen - die geen zin hebben, passen we opnieuw aan:

```
$(function(){
  …
$('<a href="#about" title="terug nr boven">terug nr
  boven</a>').insertBefore(':header:gt(1)');
  …
});
```

De selector `:gt(n)`(*greater than*) is een JQ selector (geen CSS) die gebruik maakt van de *index* van het element binnen de *wrapped set*. Ze **reduceert** die set tot die elementen met een index **groter** dan de aangeduide.
In ons voorbeeld zullen hyperlinks pas verschijnen vanaf de derde kop: index 1 is het tweede element in de set.

3.3.11 Button widget

Je hebt daarstraks niet enkel de jQ Core library geïnstalleerd, maar ook het UI framework. Dat betekent dat we op elk moment kunnen gebruiken maken van de ***widgets*** die het bevat. Dit is een ideaal moment om een heel eenvoudige widget te gebruiken, nl. de **Button**.

De UI Button is in staat `button, a, input type=submit, input type=reset, input type=radio, input type=checkbox` elementen te stylen volgens het Framework. Daarbij worden de styles en kleuren van het gebruikte thema (bij ons *UI-Lightness*) toegepast.

☞ Het toepassen ervan is *Oh-Zo-Simpel*:

```
$(function(){

  …

$('<a href="#about" title="terug nr boven">terug nr boven</a>')
  .insertBefore(':header:gt(1)').button();

  …

});
```

Alle '*terug nr boven*' hyperlinks veranderen op slag in knoppen!

De `button()` method bouwt de hyperlink volledig om naar een complex element met meerdere UI classes. Gebruik nu even een webdeveloper tool zoals FireBug om de HTML van het *button* element te bekijken. Dan merk je op dat het `a` element meerdere ui-classes toegekend gekregen heeft en er is zelfs een `span` element aan toegevoegd!

```
<a class="ui-button ui-widget ui-state-default ui-corner-all ui-button-text-
  only" title="terug nr boven" href="#about" role="button">

<span class="ui-button-text">terug nr boven</span>

</a>
```

Om er iets aan te veranderen hoef je die classes niet manueel te veranderen: als we de button willen beïnvloeden, doen we dat aan de hand van een `options` object binnen de method.

☞ We willen een ikoontje aan de rechterkant van de knop:

```
$('<a href="#about" title="terug nr boven">terug nr boven</a>')
        .insertBefore(':header:gt(1)')
        .button( { icons: {secondary:'ui-icon-circle-triangle-n'} });
```

Bespreking:

- Het `{ }` object binnen de haakjes van de method bevat **instellingen** als properties. Je kan de mogelijkheden steeds bekijken op de relevante pagina van de widget.
- de property `secondary` plaatst het icoontje rechts van de tekst, `primary` zou dat links doen
- inspectie met je dev tool toont je dat het ikoontje automatisch als background wordt ingesteld door een CSS class: *ui-icon-circle-triangle-n*

☞ de demo van de widget is te vinden op http://jqueryui.com/button/ en
☞ uitleg over de mogelijke opties op http://api.jqueryui.com/button/
☞ om te weten te komen welke `class` je moet toepassen voor elke ikoontje, kijk je op de theme pagina http://jqueryui.com/themeroller/ bij *Frameworks icons.* Dit is een *sprite* met alle ikoontjes: laat de muis erover glijden om de `class` ervan te zien.

3.3.12 Een lijst aanmaken

Het aanmaken van elementen met *chaining* heeft subtiliteiten die we beter uittesten. Geen betere leerschool dan *bewust* enkele fouten maken… Veronderstel dat we een *bulleted list* willen aanmaken aan de hand van het *array*.

☞ Plaats het volgende array in *about.js*:

```
  …
var lijst = ['roger', 'evelyn','hilde','jan'];
  …
```

De vaste html van deze pagina is te vinden in *content/inhoud.php*.

☞ Open dit bestand en zoek de functie *getAbout()*

Daar vind je een titel *'Ons team'* met een `id`, verder geen tekst. Je hoeft verder in dit php bestand niets te doen, je kan hem eventueel opnieuw sluiten.

We willen de teamlijst net onder die titel als *unordered list* aanmaken.

☞ We starten door één `ul` element toe te voegen na de titel:

```
…
$('<ul>').insertAfter('#team');
…
```

Bespreking:

- Let op de schrijfwijze: de *jQuery function* `$('<ul>')` levert hier een **nieuw aangemaakt element**, terwijl `$('ul')` een **selectie**(set) van bestaande `ul` elementen zou geven
- `insertAfter` plaatst de `ul` onmiddellijk na de `h3.`
  `insertAfter` voegt dus toe **na** een element, niet **in**!
- voorlopig levert dit een leeg `ul` element op: ongeldige HTML want hij moet minstens één `li` hebben. Controleer je html met *FireBug*: het `ul` element is inderdaad aanwezig

☞ de volgende code levert hetzelfde resultaat op en is interessanter want dit returnt opnieuw het element `#team`:

```
…
$('#team').after('<ul>');
…
```

☞ dit sterkt ons in ons voornemen en we proberen te *chainen*:

```
…
$('#team')
        .after('<ul>')
                .append('<li>eerste item</li>')
                .append('<li>tweede item</li>');
…
```

Op het eerste gezicht lijkt het gelukt: er is een bolletjeslijst te zien, maar controle van de code toont dat de `li` **in** de `h3` ingevoegd werd en de `ul` staat ergens eenzaam er onder…

Waarom werkt deze redenering niet?

Omdat de meeste JQ methods steeds de set returnen waarmee de chain **begon**: in dit geval het `#team` element.De bedoeling van chaining is dus dat je meerdere methods kan uitvoeren op de **beginset**.
Chaining betekent dus niet *'voortbouwen op het vorige*'.
De twee `append`'s gebeuren dus elke keer op het `#team` element.

Ook niet goed is

```
…
$('<li>eerste item</li>').appendTo('<ul>').appendTo('#team');
…
```

Hier wordt de `ul` overschreven…

Hoe gaan we dan wel tewerk? Er zijn minstens twee mogelijkheden:
Een eerste mogelijkheid is een `insertAfter` en een `append` combineren:

```
…
$('<ul>')
  .insertAfter('#team')
        .append('<li>eerste item</li>')
        .append('<li>tweede item</li>');
…
```

Bespreking:

- bij `$('<ul>').insertAfter('#team')` is het nieuw `ul` element de beginset
- daarom kunnen de `append`'s van de `li` items zonder problemen daarop gebeuren

Een andere mogelijkheid is het gebruik van haakjes om voorrang te geven aan de bewerkingen:

```
…
$('#team')
  .after($('<ul>')
        .append('<li>eerste item</li>')
        .append('<li>tweede item</li>')
  );
…
```

Hoe?

- Eerst wordt het eerste `li` item toegevoegd aan een **nieuw** `ul` element, daarna het tweede item
- pas daarna wordt het `ul` element aan de `#team` toegevoegd

☞ Dit kunnen we beter opsplitsen in twee statements:

```
…
  var $uul = $('<ul>').append('<li>eerste item</li>').append('<li>tweede item</li>');
  $('#team').after($uul);
```

```
…
```

Opmerking:

- je ziet dat we hier de variabele `$uul` beginnen met een `$`-teken (net zoals in PHP waar dit verplicht is).
  In Javascript is dat helemaal niet nodig, maar het mag.
  Waarom doen we dit?
  Om aan te geven in de benaming van de variabele dat het een *JQuery wrapped set* betreft. `$uul` is een jQ set en dat geven we zo aan.

Je bent vrij om deze manier van werken te volgen of niet, maar hoe duidelijker hoe beter uiteraard!

We kunnen dus besluiten dat *chaining* bedoeld is om meerdere acties op **hetzelfde** element los te laten en niet om (wat de naam toch wel een beetje impliceert) bouwstenen op elkaar te plaatsen.

☞ Met deze nieuwe kennis kunnen we nu een efficiënte manier bedenken om een lijst te bevolken met items uit het array:

```
…
var $uul = $('<ul>');
$.each(lijst, function(n, value){
        $('<li>').text(value).appendTo($uul);
})
$('#team').after($uul);
…
```

We zien een ongeordende lijst van namen verschijnen: wat we wilden.

Bespreking:

- de variabele *$uul* bevat een nieuw `ul` element
- de JQ functie `$.each` kan gebruikt worden om te itereren doorheen objecten of arrays en vervangt een `forin` lus. In dit geval lussen we doorheen het array *lijst*
- we passen een **anonieme functie** toe op de lus waarin we een nieuw `li` element toevoegen aan de var *$uul* en er eerst de `text` value van instellen op de waarde van elk array-item. De index *n* gebruiken we niet
- het nieuwe `ul` element wordt ingevoegd na de titel

Als je **grote stukken HTML moet invoegen** (bv.een lijst van tientallen items of een tabel) dan is het volgende alternatief beduidend sneller:

```
…
  var $uul = $('<ul>');
  var strDeLijst = '';
```

```
$.each(lijst, function(n, value){
                strDeLijst += '<li>' + value + '</li>';
})
$uul.html(strDeLijst);
$('#team').after($uul);
…
```

Bespreking:

- de JQ method `html()`is vergelijkbaar met `text()`; beide werken identiek:
  - ze **lezen** de inhoud van een element indien er *geen argument* is, of
  - ze **voegen** een inhoud **in** als je die inhoud meegeeft als *argument*.
- Terwijl `text()`de inhoud niet interpreteert en dus als platte tekst beschouwt, *parsed* `html()`die inhoud als HTML.
  `html()` is dus de JQ versie van de Javascript method `innerHTML`
- de lijst wordt hier opgebouwd als één string en dan in één keer ingevoegd. Nog beter ware ook het `ul` element erbij te nemen, maar dan moet je een container hebben om de html op toe te passen

Beperk het aantal wijzigingen in de DOM tot een minimum: probeer zoveel mogelijk voor te bereiden en grijp dan één keer in

3.3.13 Een keuzelijst aanmaken

Nu je deze eenvoudige lijst kan maken en de verschillende JQ methods begrijpt, zullen we de bolletjeslijst vervangen door een keuzelijst. Het wordt ook iets complexer: er komt een `div#teamboks` als container die zelf een andere `div#teamgegevens` en een `select` element bevat.

Er zijn reeds CSS rules aangemaakt voor de styling van deze elementen. De bedoeling is later met een AJAX call de gegevens van elk teamlid te tonen in de `div#teamgegevens`.

Je mag de eerste versie commentariëren (het array blijft actief) en vervangen door:

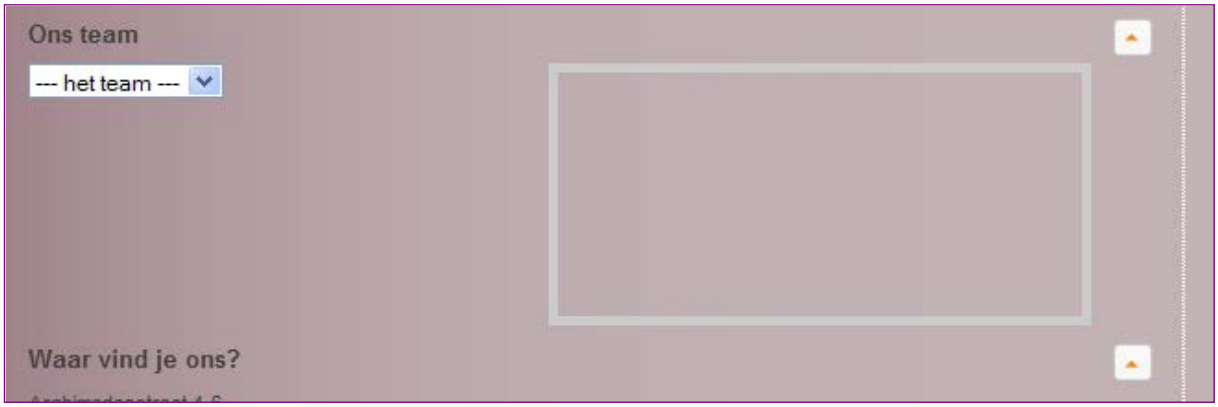
```
…
var lijst = ['roger', 'evelyn','hilde','jan'];


  //oorspronkelijke versie: een UL
  …
  // versie vr JSONgegevens
  var $container        = $('<div id="teamboks">');
  var $diefrechts       = $('<div id="teamgegevens">');
  var $keuzelijst       = $('<select id="teamkeuzelijst">');
  var strDeOptions = '<option value="">--- het team ---</option>';
  $.each(lijst, function(n, value){
                strDeOptions += '<option>' + value + '</option>';
```

```
    })
    $keuzelijst.html(strDeOptions);
    $container.append($keuzelijst).prepend($diefrechts);
    $('#team').after($container);
…
```

Bespreking:

- er worden 3 elementen aangemaakt: *$container*, *$diefrechts* en *$keuzelijst*
- een *string* variabele, *strDeOptions,* bevat alle namen uit het array als `option` elementen, inclusief een eerste, standaard `option` element
- het `select` element *$keuzelijst* krijgt de `option` elementen toegevoegd
- het nieuwe *$container* element krijgt zowel *$diefrechts* als *$keuzelijst* ingevoegd. De `div#teamgegevens` wordt eerst ingevoegd, `prepend`, omdat hij rechts zal *ge-float* worden en daarom best eerst in de DOM tree staat
- de *$container* wordt na de titel *team* geplaatst

Het resultaat is een `select` element en een zichtbaar kader aan de rechterkant.



Als kennismaking met jQuery is dit voldoende.
We gaan nu wat dieper in op enkele aspecten van de library en werken dit 'team' straks verder af.

3.4 een jQuery inhoudstafel

3.4.1 Doel

We maken een inhoudsopgave die zich baseert op de hiërarchie van de aanwezige HTML5 *sectioning* elementen: `article, section, nav, aside` en de koppen die het er in vindt.

In het bestand *about.js* vond je reeds de functie *walkTree().* Dit is een zuivere Javascript functie die doorheen alle nodes van een DOM tree "wandelt" van boven tot beneden. Het laat je dus toe de volledige DOMtree te doorlezen en er op het gepaste moment iets mee te doen. We hebben de oorspronkelijke functie lichtjes aangepast om samen te werken met jQuery.

De functie heeft vier argumenten:

- *root*: de beginnode waar de "wandeling" start
- *$list*: een jQ set met een `ol` of `ul` element waarin we de eigenlijke inhoudsopgave zullen zetten en opbouwen (onder de vorm van `li` elementen).
- *enter*: een functie die uitgevoerd wordt bij het binnentreden van een node
- *exit*: een functie die uitgevoerd wordt bij het verlaten van een node

We gebruiken hiervoor opnieuw de "*Over ons*" pagina van de website en dus komt het script ook in *about.js.*

In de pagina vind je een `section` met de `class`*"no-toc overbodig "* en daarin een `div` met de `id` "toc".

De afkorting "*toc*" staat voor "*Table of contents*".

```
…
<section class='no-toc overbodig'>
  <div id='toc'>Hier komt de automatische inhoudstafel. Het feit dat u dit leest duidt
  op een probleem: activeer dan Javascript en ververs de pagina</div>
</section>
…
```

De inhoudsopgave zal in de `div#toc` gezet worden en de aanwezige inhoud moet verdwijnen.

3.4.2 In de document.ready function

In de `document.ready` function, onderaan, typen we volgende code:

```
…
//Maak de inhoudsopgave
  var root       = $('article')[0];
  var $list      = $('<ol>');
  $('#toc').empty().append(walkTree(root,$list,enterNode,exitNode));

  }); //einde doc ready
```

Bespreking:

- De var `root` is het eerste (en waarschijnlijk enige) item in de set van `article` elementen. Merk op dat deze var, door de `[0]`, een DOM node is en geen *wrapped set*
- De var `$list` is een set met een nieuw `ol` element. Het is verder leeg
- Eerst legen we de `div#toc` met de jQ `empty()` method: alle inhoud wordt verwijderd
- Onmiddellijk erna (door chaining), voegen we de return waarde van de functie `walkTree` opnieuw in in dit element
    - de `article` node wordt doorgegeven als *root* argument
    - de *$list* variabele als tweede argument
    - de functies *enterNode* en *exitNode* moeten we nog maken

In de functie *walkTree()*, merk je dat deze de variabele `$list` opnieuw returnt, deze zal echter volledig opgebouwd zijn. Deze returnwaarde wordt dus ingevoegd in `div#toc.`

3.4.3 Nodes controleren

De rest van onze code mag buiten de `document.ready` functie staan, net als *walkTree().*

Eerst initialiseren we enkele globale variabelen:

```
…
});//einde doc ready

var arrKoppen    = ["h1","h2","h3","h4","h5","h6"];
var arrSections = ["article","section","aside","nav"];
var getal        = 1;

var walkTree = function (root, $list, enter, exit){
…
```

Bespreking:

- De var *arrKoppen* is een array met alle mogelijke *hX elementen*.
- De var *arrSections*is een array met alle *sectioning* elementen: zij die in aanmerking komen om in de inhoudstafel opgenomen te worden.
- De var *getal* is een integer

De functie *walkTree()* overloopt alle DOM nodes: zowel textnodes, elementnodes, commentaarnodes, etc..

Omdat wij enkele geïnteresseerd zijn in *sectioning* elementen, definiëren we een functie *checkNode* die een selectie zal maken:

```
var checkNode = function(node){
  // controleert of deze node in aanmerking komt voor de inhoudsopgave
```

```
// enkel als elementNode, in de lijst sectionElms en geen no-toc
var strNotoc = "no-toc";
return (node.nodeType==1 &&arrSections.indexOf(node.tagName.toLowerCase())>=0
                         &&node.className.indexOf(strNotoc)==-1)
}
```

Bespreking:

- De *node* wordt doorgegeven als argument
- Een string var *strNotoc* bevat de tekst die we gebruiken als `class` om uitzonderingen aan te duiden die niet moeten opgenomen worden
- De waarde van de evaluatie van de expressie wordt gereturnt:
  - enkel elementen: `nodeType`==1
  - enkel indien de `tagName` te vinden is in *arrSections*
  - enkel indien "no-toc" NIET voorkomt in het `class` attribuut

De Javascript functie `indexOf` kan zowel voor een array als voor een string gebruikt worden. Het zoekt een in het array of in de ander string. Indien ze die vindt, geeft ze de positie ervan terug, indien niet gevonden geef ze -1.

Je vraagt je misschien af waar jQuery hierbij blijft? Omdat we hier met nodes werken is het veel efficienter met DOM en Javascript functies tewerk te gaan. Tenslotte is jQuery een Javascript library die JS verbetert waar het nodig is. Waar het niet nodig is, hoeft het ook niet.

3.4.4 De lijst opbouwen

Voor elke node die *walkTree* doorloopt kijken we of die node in aanmerking komt voor de opbouw van de inhoudsopgave. De functie *enterNode()* neemt het grootste deel van die opbouw voor zijn rekening:

```
function enterNode(node,$list) {
  //bouwt $list op bij het binnengaan van een node
        if(checkNode(node))
        {
                var $nieuw = $('<li>').attr("tabindex",getal.toString());
                var $a  = $('<a>').attr({
                        "href"  :"#" + getal.toString(),
                        "id"    :"o" + getal.toString()
                        });

                node.setAttribute("id",getal.toString());
                getal++;

                $a.text(zoekKoppen(node));
                $nieuw.append($a);

                if($list[0].tagName=="LI"){
                        var $nieuweLijst = $('<ol>').append($nieuw);
                        $list.append($nieuweLijst);
                        $list   = $nieuw;
```

```
                }
                else{
                        $list.append($nieuw);
                        $list   = $nieuw;
                }
  }
  return $list;
}
```

Bespreking:

- De huidige *node* en *$list* worden doorgegeven als argument
- na controle door *checkNode*
- bouwen we een *$nieuw* item op dat een `li` element bevat.
- er wordt onmiddellijk een `tabindex` attribuut aan toegevoegd met de jQ functie `attr(attribuut, waarde)`
- er wordt een `a` element aangemaakt en ook hier worden attributen aan toegevoegd. Merk op dat dit deze keer op een andere manier gebeurt: de attributen worden als een *map* doorgegeven: een object waarin de attributen als properties met waarden aanwezig zijn.
    - Het `href` attribuut verwijst naar een bookmark opgebouwd met de huidige waarde van getal, vb. "#3"
    - Het `id` attribuut wordt op dezelfde manier opgebouwd, vb "o3"
- de *node* zelf krijgt de huidige waarde van *getal* toegewezen als id, dus vb "3"
- *getal* wordt verhoogd

- de tekst van de hyperlink wordt geleverd door de – nog te maken – functie *zoekKoppen()*
- de *$nieuw* node (een `li` element) krijgt de hyperlink toegevoegd
- als de huidige versie van *$list* een `li` element is (als er een sublijst gestart moet worden)
    - wordt een `ol` element vastgemaakt aan de *$lijst* en wordt zijn eerste `li` de huidige *$list*
    - zoniet, wordt *$nieuw* toegevoegd aan $list (een `ol`)

De functie *exitNode()* sluit de opbouw van nodes af:

```
var exitNode = function(node,$list){
  //bij het verlaten van de node
  if(checkNode(node)){
        if($list[0].tagName=="OL") {$list = $list.parent()}
        $list = $list.parent();
  }
  return $list;
}
```

De functie *zoekKoppen()* zoekt en leest de inhoud van de belangrijkste kop per sectie:

```
var zoekKoppen = function(node)
{
  var $node = $(node);
  var koptekst = "";
  //zoek de hoogste kop, return zijn tekst
  $.each(arrKoppen,function(i,v){
          var $kop = $(v,$node);
          if($kop.length > 0) {
                        koptekst = $kop.first().text();
                        return false;
                        }
          })
  return koptekst;
}
```

Bespreking:

- de DOM *node* wordt doorgegeven als argument
- die DOM *node* wordt omgezet in een jQuery set met $()
- de var *koptekst* wordt een lege string
- de core functie `each` lust doorheen het array *arrKoppen*
    - er wordt een set samengesteld voor elk item in *arrKoppen*
    - met de *$node* als context (dus wordt enkel gezocht vanuit *$node*)
    - als er minstens één dergelijke kop gevonden wordt:
        - wordt de var *koptekst* gevuld met de `text()` van het eerste item van die set
        - de `each` lus wordt verbroken door de `return false`
- de *koptekst* wordt door de functie gereturnt, een lege string indien er geen kop zou zijn

Het resultaat:


Wie zijn we en wat doen we?
1. Wie zijn we en wat doen we?
1. Ons bedrijf
1. Wie zij we?
2. Ons team
3. Contacteer ons
4. Waar vind je ons?
2. Nieuws
1. Opendeurdagen 2009
2. Opendeurdagen 2008
3. Materialen
1. Terracota bloempotten
2. Potgrond
4. Links

3.5 Storage

Op de *'Plantenshop'* pagina van de website kan de gebruiker een selectie maken uit onze plantencollectie. Initieel kan men enkel een keuze maken uit de 'Soorten' planten: *heesters, vaste planten, klimplanten*,…

We willen echter ook *'geavanceerd zoeken'* voorzien: daarvoor dient de hyperlink die reeds klaar staat.

- ☞ Open de pagina *content/inhoud.php* en bekijk de functie *getPlanten()*. Die bevat de code voor het formulier en zijn resultaat.

Je bemerkt een `div#zoeken` met daarin genest een `div#adv_zoeken`.
Deze laatste is echter niet te zien op de pagina dank zij het inline `style` attribuut met `display:none`.

```
…
//zoeken controls
  $str .= "<section>
                <div id='zoeken'>
        <p><a id='adv_zoeken_link' href='#'>geavanceerd zoeken</a></p>
                <form name='frm1' id='frm1' class='cmxform' action='index.php'
  method='get'>
                <input type='hidden' name='page' value='shop' />
                <p><label for='soort_id'>Soorten: </label>". $soorten_dd . "</p>";


  //adv zoeken
  $str .= "<!--start geavanceerd zoeken -->
        <div id='adv_zoeken' style='display:none'>";
  $str .= "<p><label for='kleur'>kleur: </label>". $kleuren_dd . "</p>";
  $str .= "<p><label for='hoogte'>hoogte tussen: </label>
        <input type='text' id='hoogte_min' name='hoogte_min' size='4' value='0'  />
  en    <input type='text' id='hoogte_max' name='hoogte_max' size='4' value='5000'
  /></p>
        <!--slider hier-->
        <div id='slider-range-hoogte' class='slider'></div>";
        $str .= "</div>
  <!--  einde geavanceerd zoeken -->";


  $str .= "<input type='submit' value='zoeken' /><input type='reset' />
  </form>
  </div>
        </section>";
…
```

Dus momenteel is enkel het zoekvak SOORTEN te zien.

Het keuzevak KLEUR en de vakken HOOGTE in "geavanceerd zoeken" zijn niet te zien.

3.5.1 toggle

De bedoeling is dat de hyperlink toegang geeft tot de geavanceerde zoekmogelijkheden en dat deze situatie kan omgekeerd worden.

- ☞ eerst en vooral verwijder je het `style="display:none"` attribuut uit `div#adv_zoeken`

nu worden de geavanceerde opties getoond.
We zullen `div#adv_zoeken` dynamisch verbergen via een *scriptstatement*.

- ☞ maak een nieuw Javascript bestand *shop.js* aan in de map *js* van de applicatie.
- ☞ pas*index.php* aan voor de `case` "shop" pagina en koppel het nieuwe script:

```
…
case "shop":
  /*** Planten pagina, PHP, non-ajax ***/
  //init zoekvariabelen
  $soort_id      = (isset($_GET['soort_id']))?$_GET['soort_id']:'%';
  $kleur         = (isset($_GET['kleur']))?$_GET['kleur']:'%';
  $hoogte_min    = (isset($_GET['hoogte_min']))?intval($_GET['hoogte_min']):0;
  $hoogte_max    = (isset($_GET['hoogte_max']))?intval($_GET['hoogte_max']):5000;
  $tpl['title']  = "de Plantenshop: ons aanbod";
  $tpl['body_id']= "shop";
  //content
  $tpl['rechts']        = getPlanten($soort_id, $kleur, $hoogte_min, $hoogte_max);
  $tpl['paginaScripts'] = getScriptElements("js/shop.js");
  break;
…
```

- ☞ in de pagina *shop.js* zetten we een `document.ready` handler:

```
$(function(){


});
```

- ☞ om gemakkelijk te werken maken we twee objectvariabelen aan:

```
var $advZoeken          = $('#adv_zoeken');
var $advZoekenLink      = $('#adv_zoeken_link');
```

wat is wat:

- de variabele `$advZoeken` is de *wrapped set* met de `div` die de geavanceerde zoekopties bevat
- de var `$advZoekenLink` is de hyperlink

- ☞ eerst maken we een eenvoudig script die de beginsituatie herstelt:

```
$(function(){
  var $advZoeken        = $('#adv_zoeken');
  var $advZoekenLink    = $('#adv_zoeken_link');

  $advZoeken.hide();
  $advZoekenLink.click(function(e){
        e.preventDefault();
        toggleZoeken($(this),$advZoeken)

  })

});//einde doc ready
```

Bespreking:

- we beginnen met de `div#adv_zoeken` opnieuw te verbergen d.m.v. de JQ method `hide()`. We geven `hide()`ook geen argument mee, dus gaat het snel. Dat gebeurt onmiddellijk na het inladen.
- daarna registreren we een *event handler* voor het `click` event van de hyperlink:
  - bemerk de *e* in de anonieme functie
  - de `e.preventDefault()`zorgt ervoor dat de hyperlink's *default action* niet uitgevoerd wordt. Het argument `e` is het event zelf en hebben we meegekregen in de event handler . Dit statement verhindert dat de pagina naar boven springt (op een scherm met kleine resolutie)
  - we delegeren de uitvoering naar een functie *toggleZoeken()*. Waarom?
    omdat we straks wijzigingen aanbrengen en dat gaat makkelijker bij een modulaire aanpak
  - we geven twee argumenten door:
    - *$(this)* een wrapped set met de hyperlink
    - *$advZoeken*: de set met de `div`

☞ De functie *toggleZoeken()* mag buiten de `document.ready` handler geschreven worden :

```
function toggleZoeken($lienk, $el){
   /*
   @$lienk       de hyperlink
   @$el het element dat getoggled moet worden
   */

  $el.toggle('slow', function(){
      tekst = ($el.css('display')=="none")?"geavanceerd zoeken":"eenvoudig zoeken";
      $lienk.text(tekst);
   })
```

```
}
```

Bespreking:

- de functie heeft 2 argumenten *$lienk* en *$el*, respectievelijk de hyperlink waarop geklikt wordt en het element dat verborgen/getoond moet worden
- de jQ method `toggle()`toont/verbergt een element.
  Het optionele argument *speed* kan *'slow', 'normal', 'fast'* zijn of een aantal milliseconden zijn
  Er kan een *callback* functie aan gekoppeld worden: een andere functie die uitgevoerd wordt bij de `toggle`.
  `hide()` en `show()`werken op dezelfde manier.
- wij plaatsen er een anonieme functie als *callback* in:
  - deze test de `display` property van `$el` (= *$advZoeken)* en naargelang kiest hij een andere tekst voor de variabele *tekst*
  - *tekst* wordt dan met `text()`in `$lienk`(= *$advZoekenLink*) geplaatst: zo wijzigt ook de tekst van de link navenant

☞ Probeer dit even uit: je moet in staat zijn de geavanceeerde zoekopties te tonen/verbergen en de hyperlinktekst wijzigt mee.

3.5.2 localStorage

Maar we willen ook dat deze *setting –eenvoudig/geavanceerd-* bewaard wordt om toegepast te worden bij een volgende bezoek aan de pagina. Daarvoor hadden we vroeger *cookies* nodig, met moderne browsers gebruiken we ***localStorage***.

`localStorage` en `sessionStorage` laten de programmeur toe op een eenvoudige manier gegevens op te slaan op de PC van de gebruiker, respectievelijk permanent of tijdelijk.

Voor meer details verwijzen we naar de Javascript cursus. We gaan er van uit dat we in een recente browser werken.

☞ we wijzigen het vorige script:

```
$(function(){

  var $advZoeken        = $('#adv_zoeken');
  var $advZoekenLink    = $('#adv_zoeken_link');
  //$advZoeken.hide();

  //lees localStorage
  var zoek              = localStorage.getItem("advZoeken");
  var setting           = (zoek!=0 && zoek!=1)?0:zoek;
  //onmiddellijk toepassen
  toggleZoeken(setting,$advZoekenLink,$advZoeken);

  $advZoekenLink.click(function(e){
        e.preventDefault();
        setting = 1 - setting; //bitwise Xor
```

```
        toggleZoeken(setting,$(this),$advZoeken);
        localStorage.setItem("advZoeken",setting);


  })
});//einde doc ready
```

Bespreking:

- wis of commentarieer het statement `advZoeken.hide()`
- we lezen de waarde van het `localStorage` item *"advZoeken"* in in de variabele *zoek*
- de var *setting* neemt de waarde van *zoek* over, tenzij zoek niet bestaat (null waarde = geen localStorage item gevonden), dan krijgt *setting* de waarde 0
- eenmaal de *setting* gekend, kunnen we het wel/niet tonen bepalen:
  dat gebeurt met de functie *toggleZoeken()* - die aangepast moet worden met een extra (eerste) argument
- de functie *toggleZoeken()* wordt onmiddellijk na het lezen van de setting uitgevoerd om de toestand van een vorig bezoek terug in te stellen
- in de `click` event handler
  - bepalen we de *omgekeerdesetting*: als *setting* `0` is (advzoeken niet getoond) dan wordt *waarde* `1` en omgekeerd.
  - nu roepen we toggleZoeken op met de omgekeerde setting zodat de toestand omkeert
- eenmaal dat gebeurd is, schrijven we het `localStorage` item opnieuw weg met de nieuwe setting

☞ als je dit allemaal volgt, passen we de functie *toggleZoeken()* aan:

```
…
  function toggleZoeken(toon, $lienk, $el){
   /*
  @toon          1|0 setting tonen of verbergen
   @$lienk       de hyperlink
   @$el          het element dat getoggled moet worden
   */

  //eerste versie
  //$el.toggle('slow', function(){
  //tekst = ($el.css('display')=="none")?"geavanceerd zoeken":"eenvoudig zoeken";
  //$lienk.text(tekst);
  //})

  var txt_een    = "eenvoudig zoeken";
  var txt_adv    = "geavanceerd zoeken";
```

```
if(toon==1){
                $el.show('slow');
                $lienk.text(txt_een);
}
else if(toon==0){
                $el.hide('fast');
                $lienk.text(txt_adv);
        }
else{throw new Error("arg toon verkeerd")}
}
…
```

verdere bespreking:

- de functie heeft nu een derde argument *toon* waarmee de setting doorgegeven wordt
- twee interne tekstvariabelen bepalen de tekst getoond in de hyperlink
- als toon `1` is dan
  - wordt *adv_zoeken* getoond
  - wijzigt de tekst van de hyperlink
- als toon `0` is dan
  - wordt *adv_zoeken* verborgen
  - wijzigt de tekst van de hyperlink
- als toon nog iets anders is, dan is er een fout

☞ probeer dit even uit: wissel de toestand, sluit de browser en open de pagina opnieuw: de toestand moet dezelfde zijn als waarin je afsloot.

3.6 jQuery UI

De **jQuery UI library** is bedoeld om de opmaak van

3.6.1 Widgets

een aantal handige *'widgets', effects* en *utilities* die met weinig programmacode kunnen toegepast worden en zo de bezoekers van je website veel gebruiksgemak geven. Veel programmeurs gebruiken jQuery uitsluitend voor de widgets.

Waarom het wiel uitvinden als het toegepast kan worden als widget?

We hebben al de **Button** toegepast, nu proberen we ook deze UI widgets uit:

- de **Slider** widget
- de **Accordion** *widget*
- de **Tabs** *widget*
- de **DatePicker** widget
- de **Dialog** widget

Terwijl je deze projecten uitvoert, kan je nalezen op

- de **documentatie** van de widgets op http://jqueryui.com/demos/
- de **API** van de widgets op http://api.jqueryui.com/

De verschillende *themes* hebben geen enkele invloed op de functionaliteit van de widgets, ze bepalen enkel de 'look' ervan.

3.6.2 UI library koppeling

De UI library is al gekoppeld, dat heb je daarnet gedaan:

- de koppeling naar het JS bestand
  js/vendor/jquery/js/jquery-ui-1.10.3.custom.min.js
- de koppeling naar het UI stylesheet
  js/vendor/jquery/css/ui-lightness/jquery-ui-1.10.3.custom.min.css

Zie je straks echter een foutmelding in de aard van

"$("#slider-range-hoogte").slider() is not a function"

dan duidt dat op een fout in de koppeling: de function `slider` werd niet gevonden. 9/10 kans dat de koppeling van de UI library niet goed is.

3.6.3 Slider

- ☞ open het bestand *content*/inhoud.*php,* zoek de functie *getPlanten()*.
- ☞ zoek in de HTML code naar een commentaar `<!--start slider-->.`

Door de **Slider** *widget* hier te gebruiken kunnen we de gebruiker wat meer gemak bieden voor het instellen van de twee waarden `hoogte_min` en `hoogte_max`, zonder daarom de tekstvakken te verwijderen. Het zullen nog altijd de `input` elementen *hoogte_min* en *hoogte_max* zijn die hun waarden doorgeven aan de server.

☞ vervang de commentaar door een geïdentificeerd `div` element:

```
…
$str .= "<div>
         <label></label>
         <div class='controlbox vert'>
         <!--start slider -->
         <div id='slider-range-hoogte' class='slider'></div>
         <!--einde slider -->
         </div>
         </div>";
  …
```

☞ draag er zorg voor dat de variabele *$str* nog steeds goed afgesloten is met een aanhalingsteken.

☞ de pagina-specifieke javascript bevindt zich in *js/shop.js*: open dat bestand.

In de `document.ready` handler (waar reeds heel wat staat met betrekking tot advanced zoeken), roepen we nu de *widget* **slider** op voor deze div:

```
…
$("#slider-range-hoogte").slider();
…
```

Een slider verschijnt op het formuliertje. In het stylesheet *plantenshop.css* bevindt zich reeds een *style rule* die o.a. de breedte van de slider beperkt.

We willen echter een *slider* met twee knoppen, eentje voor de minimum- en eentje voor de maximumwaarde. Uiteraard moet dan de waarde van elke knop gereflecteerd worden in het juiste tekstvak.

☞ daarvoor specificeren we enkele opties door een *option object* te plaatsen als argument van de *slider()* functie:

```
…
$("#slider-range-hoogte").slider({
         range: true,
         values: [100, 500],
         min: 0,
         max: 5000,
         step: 10,
         slide: function(event, ui) {
                $("#hoogte_min").val($(this).slider("values", 0));
                $("#hoogte_max").val($(this).slider("values", 1));
         }
  });
```

```
…
```

Bespreking:

- met de optie `range` zorgen we voor twee sliderknoppen
- de `values` zijn de waarden die we toekennen bij het opstarten, het is een array. Omdat `range:true` is geven we er twee mee, maar er kunnen er ook meer zijn: als `range:false` (of afwezig) is kan je meerdere sliderknoppen met hun eigen value hebben
- het `min` wordt voor de hoogte ingesteld op 0
- het `max` op de hoogste plant (boom) die we hebben
- met `step` stellen we de sprong in van de slider
- de property `slide` is een eventhandler die de waarde van de tekstvakken *hoogte_min* en *hoogte_max* updatet. De functie `val()` wordt gebruikt om de waarde van een formuliercontrol te lezen/zetten.
  `this` is hier de slider zelf.

Oorsponkelijk staan de standaardwaarden van de twee vakken resp. op 0 en 5000. Je bemerkt dat ze verspringen naar 100 en 500 als je de slider beweegt. Om de startwaarden van de slider ook in onze teksvakken te krijgen,

☞ verwijderen we in de HTML het `value` attribuut van de `input` elementen in *inhoud.php*:

```
…
$str .= "<div>
        <label for='hoogte'>hoogte tussen: </label>
        <div class='controlbox vert'>
        <input type='text' id='hoogte_min' name='hoogte_min' size='4' class='kort'
 value='0'  /> en
        <input type='text' id='hoogte_max' name='hoogte_max' size='4' class='kort'
 value='5000' />
        </div>
        </div>";
 …
```

☞ en voegen nog wat code toe aan het script:

```
…
$("#slider-range-hoogte").slider({
 range: true,
 min: 0,
 max: 5000,
 step: 10,
 values: [100, 500],
 slide: function(event, ui) {
```

```
        $("#hoogte_min").val($(this).slider("values", 0));
        $("#hoogte_max").val($(this).slider("values", 1));
  }
});

//initialiseren van de startwaarden
$("#hoogte_min").val($("#slider-range-hoogte").slider("values", 0));
$("#hoogte_max").val($("#slider-range-hoogte").slider("values", 1));
…
```

Hier doen we hetzelfde als in het `slide` event, maar bij het opstarten. We kunnen ook `this` niet gebruiken, we moeten de slider met naam refereren.

Als je de slider nu probeert werkt alles goed met uitzondering van één probleempje: "*de slider kan moeilijk teruggeplaatst worden op de min en max waarden*".

Dit wordt veroorzaakt door de niet-correcte waarde van het `slide` event. We lossen dat op door het `stop` event te gebruiken.

☞ pas aan:

```
…
$("#slider-range-hoogte").slider({
  range: true,
  min: 0,
  max: 5000,
  step: 10,
  values: [100, 500],
  slide: function(event, ui) {
        $("#hoogte_min").val($(this).slider("values", 0));
        $("#hoogte_max").val($(this).slider("values", 1));
        },
  stop: function(event, ui) {
        $("#hoogte_min").val($(this).slider("values", 0));
        $("#hoogte_max").val($(this).slider("values", 1));
        }
});
//initialiseren van de startwaarden
$("#hoogte_min").val($("#slider-range-hoogte").slider("values", 0));
$("#hoogte_max").val($("#slider-range-hoogte").slider("values", 1));
…
```

Tenslotte een extraatje:

de UI Slider biedt geen mogelijkheid om een tekst te plaatsen op de knoppen.

Als we de dynamische HTML bekijken merken we dat de knoppen een `a` element zijn met de `class` *ui-slider-handle*.

☞ We maken daarvan gebruik om een `title` attribuut toe te voegen:

```
…
});

  //initialiseren van de startwaarden
  $("#hoogte_min").val($("#slider-range-hoogte").slider("values", 0));
  $("#hoogte_max").val($("#slider-range-hoogte").slider("values", 1));

  //toevoegen van een title text aan de slideknoppen
  $(".ui-slider-handle","#slider-range-hoogte")
          .first().attr({'title':'Minimum hoogte'})
          .end()
          .last().attr({'title':'Maximum hoogte'})
…
```

Bespreking:

- de selectie wordt gemaakt op de `class` *ui-slider-handle* binnen de context van *#slider-range-hoogte*. Het resultaat zal de twee `a` elementen zijn.
- met `first()` vernauwen we de selectie tot de eerste knop en plaatsen er het title attribuut op
- `end()` doet de selectie terugkeren naar de vorige toestand
- en `last()` haalt er de tweede knop uit waar we hetzelfde doen

De slider knoppen hebben nu een begeleidende tekst.

3.6.4 Accordion

De Accordion widget maakt panelen die je kunt open- en dichtklappen. Ideaal om een grotere inhoud weinig ruimte te doen innemen.

- ☞ bekijk de homepagina van het website
- ☞ open opnieuw *content/inhoud.php* en zoek de functie *getHome()*
- ☞ zoek het HTML gedeelte tussen de commentaar tags `<!--start keuzes-->` en `<!-- einde keuzes -->`

Dit deel bevat de gekleurde blokjes *"Eén- en twee-jarigen"* tot *"water en vijverplanten".*

Elk blokje in de `section` bestaat uit een `h3` gevolgd door een `div` met één of meerdere `p` elementen met tekst:

```
<section id='keuzes'>
<h3>Een- en tweejarigen</h3>
<div>
<p>Een- en tweejarigen kunt u zelf zaaien. Deze planten bloeien vaak meer en langer
  dan vaste planten en brengen kleur en variatie. Daarnaast kunnen zij goed leemtes
  vullen in de border. Een- en tweejarigen zijn te krijgen in vele verschillende
  vormen, van hele kleine tere plantjes tot klimmers die in onafzienbare tijd uw muur
  of schutting bedekken met een kleurige bloemenpracht. <a
  href='shop.php?soort_id=3'>Bekijk onze selectie Een- en tweejarigen</a></p>
</div>
<h3>Kruiden, vaste planten en heide</h3>
<div>
<p>De groep…
```

Omdat die nogal wat plaats innemen, willen we ze in een ***Accordion*** *widget* plaatsen.

☞ de scripts voor deze pagina plaatsen we in een nieuw javascript bestand *home.js* dat je ook in de map *js* zet.

```
// JavaScript Document
//script voor homepagina
```

```
$(function(){


});//einde doc ready
```

☞ ook dit bestand moeten we koppelen in *index.php* bij de "homepagina" :

```
…
else {
        /*** homepagina ***/
        $tpl['body_id']                 = "home";
        //content
        $tpl['rechts']                  = getHome();
        $tpl['paginaScripts']           = getScriptElements("js/home.js");
}
…
```

Hoe werkt de *Accordion widget*?

De ***ideale*** HTML structuur om een *Accordionwidget* op toe te passen is deze:

```
<div id="container">
      <h3><a href= "">titel 1</a></h3>
      <div>
            <p>inhoud</p>
            <p>meer inhoud</p>
      </div
      <h3><a href= "">titel 2</a></h3>
      <div>
            <p>inhoud</p>
            <p>meer inhoud</p>
      </div
…
</div>
```

- een "**container**" bevat meerdere *sub-blokken* die telkens bestaan uit
  - een **titel**, hier een `h3`
    - een hyperlink is niet verplicht, maar geeft extra werk aan CSS indien niet gebruikt
    - een hyperlink alleen, zonder de `h3`, is ook voldoende
  - een **inhoudsblok** `div` die de *next sibling* is van de titel
- je mag andere elementen gebruiken voor titel en inhoud: dat kunnen `li` elementen zijn of een `h3` en een `div` of een `h2` met een `p`, etc…
- het JQ script wordt enkel toegepast op de container
- als er een probleem is, dan wordt deze structuur zonder problemen getoond als gewone HTML

☞ we passen de Accordion toe op onze "container" `section#keuzes` in *home.js*:

```
$(function(){

$('#keuzes').accordion();

});
```

Het resultaat is onmiddellijk: onze blokjes zijn vervat in een Accordion!
Maar we zijn onze kleuren kwijt…

- Probeer de *widget* uit: klik ze open en dicht.

Welk paneel is open?

Het eerste inhoudsblok is standaard geopend, maar we willen dat bij het starten het tweede blok open is.

- Dan moeten we een option object meegeven in de method:

```
…
$('#keuzes').accordion({
  active:1
  });
…
```

Bespreking:

- `active` bepaalt het actieve (=geopende) element.
  De index telt vanaf 0 (=eerste)
- wil je geen enkel blok geopend, stel dan `active:false` in

Ikoontjes

Momenteel krijgen we de standaard ikoontjes,

☞ we willen een ander soort ikoontje gebruiken:

```
…
var ikoontjes = {
         header: "ui-icon-circle-arrow-e",
         headerSelected: "ui-icon-circle-arrow-s"
  }

$('#keuzes').accordion({
  active:1,
  icons: ikoontjes
  });
…
```

Bespreking:

- we maken een var *ikoontjes* aan die een object maakt met twee mogelijke properties:
  - `header` voor de gesloten blok en
  - `headerSelected` voor het **actieve** blok
  - beide maken hier gebruik van de ingebouwde ikoontjes van het UI CSS Framework.
- we gebruiken deze *var* in de property *icons*
- je kan ook de ikoontjes afzetten door `icons: false` te gebruiken
- let op de komma na de eerste optie

Hoogte

Als je even het blokje "Bomen en struiken" opent bemerk je dat er een hoeveelheid witruimte te zien onder de tekst, terwijl "Water- en vijverplanten", soms een scrollbar krijgt.

Dat kan gecontroleerd worden met de optie `heightStyle`, die kan 3 mogelijke waarden hebben:

- `"auto"` waarbij alle panelen de hoogte hebben van de grootste (standaardwaarde)
- `"fill"` panelen worden zo hoog als de ruimte van de accordion's parent toelaat.
- `"content"` waarbij een paneel zo hoog is als zijn inhoud

☞ we wijzigen deze eigenschap:

```
…
$('#keuzes').accordion({
  active:1,
  icons: ikoontjes,
  heightStyle:"content"
  });
…
```

Het blokje "Bomen en struiken" opent nu met enkel de ruimte nodig voor de tekst.

object properties zijn hoofdlettergevoelig!

heightStyle≠ heightstyle

Alles sluiten

Momenteel is er steeds één blokje open, het is onmogelijk ze allemaal te sluiten.

☞ Met `collapsible` kunnen we dat veranderen:

```
…
$('#keuzes').accordion({
  active:1,
  icons: ikoontjes,
  heightStyle:"content",
  collapsible:true
  });
…
```

Animation

De beweging waarmee de panelen openen of sluiten kan ook beïnvloed worden met de property `animate`. De effecten gebruiken de easing fucnties die ingebouwd zijn. Het standaard effect is "*swing*".

- ☞ eerst zetten we even alle *animation* af:

```
…
$('#keuzes').accordion({
  active:1,
  icons: ikoontjes,
  heightStyle:"content",
  collapsible:true,
  animate:false
  });
…
```

Nu is het openen/sluiten zeer abrupt.

- ☞ daarom wijzigen we naar:

```
  animate:"easeOutBounce"
```

In een accordion zijn niet alle effecten even goed zichtbaar.

- ☞ Bekijk in de jQuery API de andere *easing* functies: http://api.jqueryui.com/easings/

Kleuren

Daarnet zijn we onze oorspronkelijke kleuren kwijt geraakt. De reden is natuurlijk dat de Accordion widget zijn eigen kleuren CSS erop toepast, ná de eigen CSS.

Het is niet mogelijk de widget zo in te stellen dat hij de oorspronkelijke kleuren gebruikt, we zijn verplicht in te grijpen in onze eigen CSS.

Het probleem met het wijzigen van de UI styles is dat je goed moet nagaan welke UI classes invloed hebben op de elementen en wat ze precies doen.

- ☞ bekijk in je *developer tool* één van de `h3` koppen van de Accordion

dan bemerk je een massa classes die door jQ toegevoegd werden:

```
<h3 class="ui-accordion-header ui-helper-reset ui-state-default ui-accordion-icons
  ui-accordion-header-active ui-state-active ui-corner-top" role="tab" id="ui-
  accordion-keuzes-header-1" aria-controls="ui-accordion-keuzes-panel-1" aria-
  selected="true" tabindex="0"><span class="ui-accordion-header-icon ui-icon ui-icon-
  triangle-1-s"></span>Kruiden, vaste planten en heide</h3>
```

Al deze classes veroorzaken laag na laag CSS die bovenop je eigen CSS komt!

De belangrijkste `class` hier is `ui-accordion-header`, net zoals er een `ui-accordion-content` bestaat voor de sibling `div`'s.

We proberen daarop in te spelen:

- ☞ open *plantenshop.css* en zoek het gedeelte over `#keuzes`
- ☞ voeg volgende CSS toe:

```
#keuzes .ui-accordion-header, #keuzes .ui-accordion-content{
        background-color:#FF7A0F;
  }
```

Dit is een test: we kijken of we alles oranje kunnen maken, maar dit mislukt???
Het kan een tijdje duren voor je doorhebt *waarom* dit niet lukt: je ziet in je dev tool dat deze style rule toegepast wordt, waarom zie je dan geen oranje?

De reden is dat er door de UI classes ook een `background-image` gebruikt wordt die bovenop de kleur gaat liggen.

- ☞ daarom doen we dit:

```
#keuzes .ui-accordion-header, #keuzes .ui-accordion-content{
        background-image:none;
        border:none;
        border-radius:0;
  }
```

Bespreking:

- de test achtergrondkleur is niet meer nodig: de oorsponkelijke kleuren komen er weer door
- we zetten expliciet de styles gebruikt door de UI af: geen `background-image` meer, geen `border` en ook geen `border-radius`

Onze accordion is klaar.

3.6.5 Tabs

Een andere ruimte-besparende *widget* is de UI Tabs: die maakt panelen met tabbladen.

- ☞ ga naar de pagina "Verzorging".
- ☞ open het bestand *content*/inhoud.*php* en ga op zoek naar de functie *getVerzorging().*
- ☞ zoek de `section` met daarin de `div#verzorging` (commentaar).

Deze `div` bevat zelf een aantal andere `div` elementen met een `id`: *"bodem", "vermeerderen", "licht*",…Momenteel volgen die blokken elkaar op als gewone tekst, ze nemen echter zeer veel plaats in, scrollen is nodig om alles te lezen.

Met de *widget* **Tabs** kunnen we ze heel snel anders voorstellen.

De vereiste HTML structuur, een *container* met daarin de blokken met `id`, hebben we al. Nu is er enkel nog een navigatiestructuur nodig onder de vorm van een *unordered list* met *hyperlinks* die verwijzen naar de `id's`.

- ☞ voeg de volgende `ul` toe in bovenaan het blok "verzorging":

```
…
<!--start div verzorging-->
<div id='verzorging'>
        <!—start UL toegevoegd voor tabs-->
        <ul>
              <li><a href='#bodem'>Bodem en voedsel</a></li>
              <li><a href='#vermeerderen'>Vermeerderen</a></li>
              <li><a href='#licht'>zon of schaduw</a></li>
              <li><a href='#waterplanten'>waterplanten</a></li>
        </ul>
        <!--einde UL toegevoegd voor tabs-->
  <div id='bodem'>
…
```

Nu hebben we een aantal **hyperlinks** naar *bookmarks* in de pagina. Test ze uit, ze moeten werken.

In een volgende stap activeren we de **Tabs** voor onze container.

- ☞ maak een nieuw JS bestand *zorg.js* en plaats dat ook in de map *js*
- ☞ voeg de koppeling ook toe aan de `case` "zorg" van *index.php*:

```
…
case "zorg":
  /*** Verzorging pagina ***/
  $tpl['title']         = "de Plantenshop: welke zorg moet je je planten geven?";
  $tpl['body_id']       = "zorg";
  //content
  $tpl['rechts']        = getVerzorging();
  $tpl['paginaScripts'] = getScriptElements("js/zorg.js");
break;
…
```

- ☞ daarna vullen we *zorg.js* aan:

```
// JavaScript Document
//script voor verzorging pagina
```

```
$(function(){
   $('#verzorging').tabs();
});
```

Het resultaat van die ene lijn code is verbazingwekkend: de **Tabs** zijn geboren.

☞ probeer ze even

Opties

Standaard is de eerste tab geselecteerd, wij willen de tweede natuurlijk. Ook hier worden de opties als *properties* van een **object** in de *method* geplaatst:

```
…
   $('#verzorging').tabs({
         active: 1
  });
…
```

Bespreking:

- `active` bepaalt de geselecteerde tab via een index getal.
  De eerste tab heeft index 0.

☞ De laatste tab "Waterplanten" willen we initieel inactief hebben:

```
…
   $('#verzorging').tabs({
         active: 1,
         disabled: [3]
  });
…
```

Bespreking:

- `disabled` is een array met de indexen van alle tabs die *zichtbaar*, maar *niet aanklikbaar* zijn

Nu zijn we uiteraard verplicht een mogelijkheid te voorzien om de tab te activeren.

☞ voeg de volgende HTML toe net boven de `div#verzorging` in *inhoud.php*:

```
…
$str .= "<section>
        <p><label><input type='checkbox' id='toonWaterplanten' />
                        info inclusief waterplanten</label></p>
        <!--start verzorging-->
        <div id='verzorging'>
                …
```

We hebben nu een *checkbox* die we kunnen aan en afzetten. We moeten er ook nog JQ code voor schrijven.

☞ plaats deze binnen de `document.ready` handler maar onder de `tabs()` method:

```
…
$('#toonWaterplanten').change(function(){
        if(this.checked){
        $('#verzorging').tabs('enable', 3).tabs( "option", "active", 3 );
        }
        else {
        $('#verzorging').tabs("option", "active", 0).tabs('disable', 3);
        }
  });
…
```

Bespreking:

- de `change` event handler werkt op het *checkbox* element, een `click` event kan ook gebruikt worden
- als deze *checkbox* dus wijzigt, testen we zijn DOM `checked` property.
- we gebruiken de *Tabs* `enable` en `disable` methods om de tab met index 3 te activeren volgens de *Boolean* waarde van die `checked` property
- bij `enable` is het logisch dat we het tabblad ook selecteren:
  dat doen we door in het `options` object de eigenschap `active` de waarde `3` te geven.
  Op die manier kan je een optie *na* de initialisatie instellen: een *setter*
- we *chainen* de twee methods
- bij `disable` moeten we eerst een ander tabblad selecteren vooraleer het te desactiveren

Een *disabled* tab is **zichtbaar** voor de gebruiker, die weet dan dat hij geactiveerd kan worden door één of andere keuze.

Het is echter helemaal niet zeker dat de tab 'waterplanten' steeds een index 3 zal hebben. We passen twee dingen aan:

- ☞ de *wrapped set* plaatsen we in een variabele bovenin de *document.ready* functie om er makkelijker mee te kunnen werken

```
$(function(){
  var $tabs = $('#verzorging');
  …
```

- ☞ we zoeken dynamisch de index van het tab-element 'waterplanten' op

```
…
$('#toonWaterplanten').change(function(){
        var wpI= $('.ui-tabs-nav a').index($('a[href=#waterplanten]'));
        if(this.checked){
                $tabs.tabs('enable', wpI).tabs( "option", "active", wpI );
        }
        else {
                $tabs.tabs("option", "active", 0).tabs('disable', wpI);
        }
  });
…
```

Bespreking:

- Om de index van de betreffende tab te weten te komen, zoeken we zijn plaats in de lijst van hyperlinks: de *wrapped set* `$('.ui-tabs-nav a')` bevat alle hyperlinks van de tab navigatie, op dit moment een viertal.
  Uit deze collectie halen we de `index()` (JQ method) van het item`('a[href=#waterplanten]')`: met name het element dat een `href` attribuut met de inhoud *'#waterplanten'*.
- Deze index stoppen we in de variabele *wpI*

Je kan ook dynamisch tabs gaan bijmaken. We proberen ook dit eens uit.
Voeg opnieuw een stukje HTML toe aan de functie *getVerzorging()* van *inhoud.php:*

```
…
<p><label><input type='checkbox' id='toonWaterplanten' /> info inclusief
  waterplanten</label></p>
<p>Soms is goede verzorging onvoldoende: een aantal ziektes belagen onze
  tuinplanten.<a href='#' id='toonZiektes'> Meer weten over ziektes?</a></p>
<!--start verzorging-->
<div id='verzorging'>
```

```
…
```

daar hoort natuurlijk ook wat JQ code bij. Voeg ook deze event handler toe in de `document.ready` handler:

```
…
$('#toonZiektes').click(function(e){
  e.preventDefault();
  var aantalTabs        = $('.ui-tabs-nav a').length;
  var tekst             = "ziektes"
  var eInh              = "<div id='" + tekst + "'>";
  var eLink             = "<li><a href='#" + tekst + "'>" + tekst + "</a></li>";
  var $nieuweTabInhoud  = $(eInh).load("inc/ziektes.html");
  $tabs.append($nieuweTabInhoud);  //inhoud toevoegen
  $tabs.find("ul").append(eLink);  //navigatie item toevoegen
  $tabs.tabs("refresh");
  $tabs.tabs("option", "active", aantalTabs);
});
…
```

Bespreking:

- de `click` event handler werkt deze keer op de hyperlink
- De `e.preventDefault()`annuleert het *default event* van de hyperlink
- de var *$tabs* is reeds beschikbaar
- we berekenen het aantal tabs in het Tabs element door het aantal hyperlinks te tellen in de navigatielijst: dat moet natuurlijk overeenkomen met het aantal inhoud containers. De property `length` vertelt het ons
- we stellen twee nieuwe elementen samen, die allebei gebruik maken van de tekst "ziektes": de een voor zijn `href` attribuut de ander voor zijn `id`
- voor de inhoud gebruiken we een extern bestand *ziektes.html*: dat laden we in het nieuwe element *eInh* met de algemene jQ functie `load()`
- dit element voegen we toe achteraan *$tabs* als laatste element
- we maken een `li` element met `a` element er in en voegen dat ook toe als laatste element van het `ul` element in *$tabs*
- we hernieuwen het Tabs element met de method `tabs("refresh")`
- we selecteren het nieuw tabblad net als voorheen met een *setter* van options

Probeer even uit.

"Houston, we have a problem!":
klikken op de hyperlink blijft nieuwe "ziektes" tabbladen produceren !!

Tabs is blind voor het tweemaal invoegen van dezelfde inhoud. Dit probleem moeten we zelf oplossen.

De event handler `click` is een korte versie van de algemene eventhandler `on`.

We kunnen

```
$('#toonZiektes').click(function(){ …
```

evengoed schrijven als

```
$('#toonZiektes').on('click',function(){ …
```

maar er bestaat ook een speciale versie van `on`, nl. `one`. Met deze eventhandler kan je een actie slechts één keer uitvoeren, daarna wordt het event "ontbonden".

We wijzigen de code:

```
…
$('#toonZiektes').one('click', function(e){
  …
})
…
```

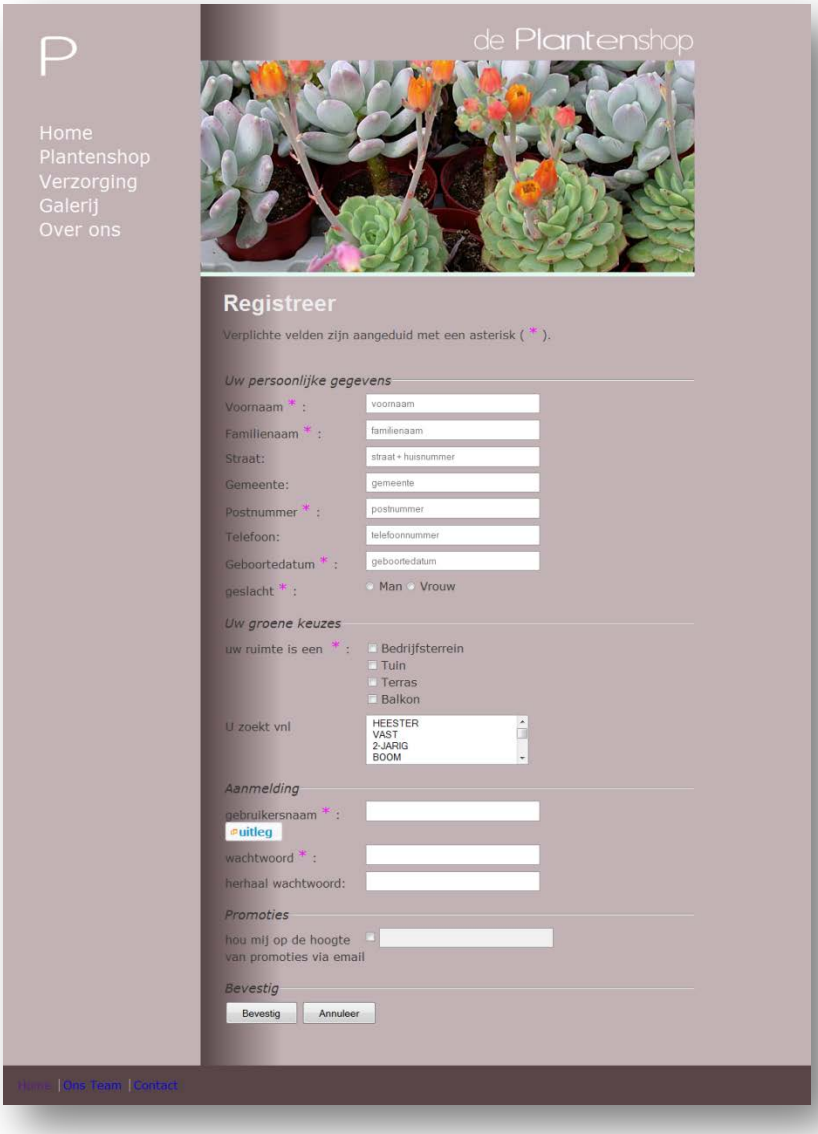
Nu bemerk je dat de hyperlink slecht éénmaal werkt, maar een hyperlink die niet werkt kunnen we beter verwijderen. Wijzig opnieuw:

```
…
$('#toonZiektes').click(function(e){
  e.preventDefault();
  var aantalTabs        = $('.ui-tabs-nav a').length;
  var tekst             = "ziektes";
  var eInh              = "<div id='" + tekst + "'>";
  var eLink             = "<li><a href='#" + tekst + "'>" + tekst + "</a></li>";
  var $nieuweTabInhoud  = $(eInh).load("inc/ziektes.html");
  $tabs.append($nieuweTabInhoud);  //inhoud toevoegen
  $tabs.find("ul").append(eLink);  //navigatie item toevoegen
  $tabs.tabs("refresh");
  $tabs.tabs("option", "active", aantalTabs);
$(this).remove();
});
```

De method `remove()` verwijdert `this`, dus de hyperlink waarop geklikt werd.

3.7 Plugin: Formulier validatie

Op de pagina "Plantenshop" bemerk je een hyperlink "*registreer hier*": klik er op.
Je komt terecht op een formulier:



Open het bestand *content/inhoud.php* en zoek de functie *getRegistreer().*
Enkele opmerkingen over dit formulier:

- het registreer formulier is specifiek gemaakt om de validator uit te testen, zoek niet teveel achter de betekenis van sommige velden en de gevraagde validatieregeltjes …
- Merk op dat de `action` van het formulier verwijst naar een bestand *reflect_data.php*. Dat is een tijdelijke oplossing om onze validatie te kunnen uittesten. Dit scriptje toont ons gewoon de naam en de inhoud van alle velden die we doorstuurden, meer niet

Gebruiksvriendelijkheid:
als je van de gebruiker verwacht dat hij velden op een correcte manier invult, zorg dan dat je ***zelf duidelijk*** *bent*

Dat kan gaan van zeer eenvoudig tot complex:

- **duid verplichte velden** aan binnen de tekst van het `label` element, doe dat tekstueel of met ikoontje, voorbeeld:

  voornaam (verplicht) :

  → een **vereist ijkpunt** voor *Toegankelijkheid*

- gebruik het `title` attribuut van het `input` element voor een *tooltip* met wat meer tekst:

```
<input title="uw straat met huisnummer" id="straat" ↵
     name="straat"  />
```

  → een **aangeraden** ijkpunt voor *Toegankelijkheid*

- gebruik het `placeholder` attribuut voor tekstvakken om een standaardtekst in het veld te zetten. De tekst verdwijnt als de focus erop komt
- Voor **meer uitleg** kunnen we een **dialoogvenster** gebruiken (met JQ). Andere mogelijkheden zijn custom tooltipvensters etc…

3.7.1 De validator downloaden en koppelen

Eén van de meest tijdrovende taken van de client-side programmeur is het maken van een validatiesysteem voor de formulieren op de site. Alhoewel er nu een zekere interne browservalidatie met HTML5 is, kiezen we toch voor de ***Validation*** jQuery plugin (door Jörn Zaefferer).

- ☞ download de laatste versie van http://bassistance.de/jquery-plugins/jquery-plugin-validation of
- ☞ download van de jQuery site op http://plugins.jquery.com/ bij *forms*.
- ☞ lees ook de documentatie hand op http://docs.jquery.com/Plugins/Validation

- ☞ Extraheer de *zip file* naar een map *jquery-validate-x.x.x* en plaats die binnen de map *js/vendor/jquery*.

In de map *jquery-validate-x.x.x* vind je een aantal submappen waarvan je *dist* en *lib* nodig hebt. *Localisation* komt misschien nog van pas maar demo moet je verwijderen op een productie site.

- ☞ In *index.php* zoek de case "registreer" en voeg twee koppelingen toe:
  - naar *registreer.js*
  - naar de validator library

```
…
case "registreer":
  /*** Registreer formulier pagina ***/
  $tpl['title']         = "de Plantenshop: registreer u als klant";
  $tpl['body_id']       = "registreer";
  //content
```

```
  $tpl['rechts']        = getRegistreer();
  $tpl['paginaScripts'] = getScriptElements(array("js/vendor/jquery/
        jquery-validation-1.11.1/dist/jquery.validate.min.js",
                              "js/registreer.js"));
  break;
…
```

Hier koppelen we twee pagina-specifieke libraries aan de pagina d.m.v. een PHP `array`. Het heeft weinig zin de validator te laden voor andere pagina's.
Je kan uiteraard in je developer tool nu checken of de plugin geladen is.

- ☞ nu maken we een bestand *registreer.js* aan en plaatsen dat in de map *js*.

Je kan ook programmatorisch controleren of de plugin zal werken door aan *object detection* te doen.

- ☞ voeg een `document.ready` handler in en schrijf het volgende:

```
$(function(){
  if(jQuery().validate) { console.log("validate geladen");  }
  else { console.log("validate NIET geladen");}
}); // einde doc.ready
```

Als de plugin library geladen is zal er een `validate` object als property van het `jQuery` object te vinden zijn. Het object is er, je moet er nu iets mee doen.

- ☞ je kan het vorig scriptje nu verwijderen of commentariëren (of laten staan)
- ☞ we starten nu de plugin **validator** op het `form` element in de pagina:

```
$(function(){
  $("#regForm").validate();
}); // einde doc.ready
```

Op zichzelf is deze **method** onvoldoende: het laadt gewoon de plugin op de `form` maar we moeten nog specificeren wat we exact verlangen van de velden en hoe en welke foutboodschappen getoond worden.
Dat doen we met *options,* die gezet worden als properties zijn van een `options` object in de `validate` method: `rules,messages` en `submitHandler`.

- ☞ voeg toe aan `validate()`:

```
$(function(){
  $("#regForm").validate({
        rules:{},
        messages:{},
        submitHandler:function(form){
              form.submit();
        }
  }); //einde validator
```

```
}); //einde doc.ready
```

- let op de `{ }` binnen de `validate()` method: alle *options* zitten vervat in dit object
- In `rules` zullen we - opnieuw onder de vorm van een properties object - voor elke formcontrol die we willen beveiligen de validatieregel vermelden.
  Let op de komma erna!
- in `messages` zetten we dan, ook onder de vorm van een object, de bericht(en) die we willen zien verschijnen
- de `submitHandler` *value* is een functie waar we de normale `submit` van het formulier oproepen, je kan hier ook andere dingen doen die je wil uitvoeren bij een succesvolle submit
- er zijn nog heel wat andere *options*: `debug` , `invalidHandler`…

Let op de komma's!
- een komma tekort tussen twee eigenschappen = probleem
- een trailing komma achteraan = probleem (met IE)

3.7.2 Een verplicht veld

We beginnen met een eerste vereiste, om te controleren of de validatie werkt, vooraleer we verder gaan.
De *voornaam* is een verplicht veld:

```
…
$("#regForm").validate({
        rules:{
                vnaam: "required"
        },
        messages:{
                vnaam: "voornaam is verplicht"
        },
        submitHandler: function(form) {
           form.submit();
        }
  });
…
```

Bespreking:

- de eenvoudigste `rule` value is "*required*" die ervoor zorgt dat enige inhoud verplicht is
- `rules` bestaat dus uit *key:value* paren, *properties* dus, waarvan we er nu eentje hebben.

Een rule kan als **enkele regel** geschreven worden of als een **samenstelling** van meerdere:

- voor een enkele regel gebruik je de `id` van het veld als *key* en de validatieregel als `string` voor de *value* :

```
vnaam: "required"
```

of

- voor meer dan één regel voor hetzelfde veld: een *object* dat bestaat uit de *key/value* paren , bijvoorbeeld

```
username: {
          required : true,
          minlength : 5
          }
```

- voor `messages` hetzelfde systeem, maar hier is de *value* het bericht dat moet verschijnen
- let op dat je **geen punt-komma** na het *key:value* paar zet! Enkel een komma kan, als er een tweede paar volgt

☞ we testen deze regel even uit: klik onderaan op de knop "Bevestig".

Bij afwezigheid van de voornaam verschijnt een **bericht** in het rood naast het veld en het `input` element wordt rood **omrand.**

☞ als je nadien de voornaam invult verdwijnt het foutbericht

Er is sprake van ***actievevalidatie*** *(eager validation)*. Het formulier zal nu wel ge-*submit* worden.

☞ ook het *familienaam* veld is verplicht: stel ook hier de rule `required` in, maar niet de `message`.

Het is interessant om te merken dat de validator **toch** een foutbericht toont. Hij neemt gewoon het `title` attribuut van de control over!

3.7.3 HTML5 attributen

Ook HTML5 attributen worden ondersteund:

☞ open *inhoud.php*, zoek de functie *getRegistreer()* en voeg een `required` attribuut toe voor de *straat*:

```
…
<div>
  <label for='straat'>Straat:</label>
  <input type='text' title='uw straat  met huisnummer'  placeholder='straat +
  huisnummer'  id='straat' name='straat' required />
</div>
…
```

☞ test uit: ook hier verschijnt een foutbericht als je niets invult

3.7.4 Fouten debuggen

Omdat de `options` nogal complex kunnen worden, kan er ook gemakkelijk iets mis gaan. Het is dan ook niet altijd duidelijk waar het misging: je weet nog wel uit de JS cursus dat bij de minste fout een formulier gewoon `submit()`, doorgaat naar de *action* pagina en je dus geen kans krijgt om eventuele foutboodschappen in de *JS console* te zien.

☞ om die reden voegen we de option `debug` toe:

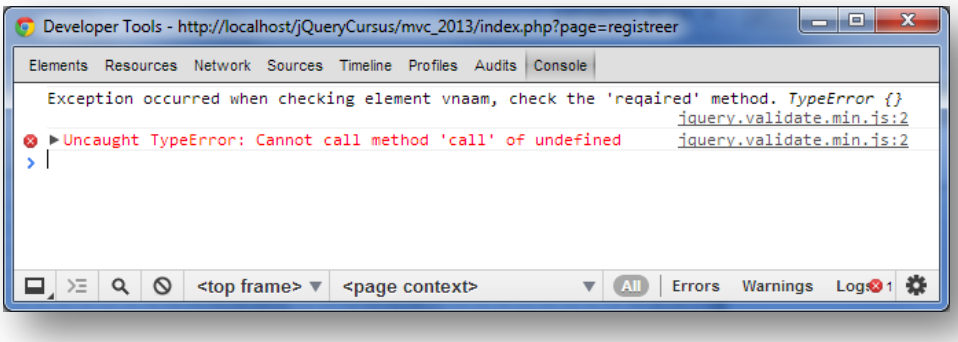
```
…
$("#regForm").validate({
          debug:true,
          rules:{vnaam: "required" },
          messages:{vnaam: "voornaam is verplicht"},
          submitHandler: function(form) {
             form.submit();
          }
  });
…
```

`debug` heeft een `Boolean` waarde, dus straks kan je gewoon `debug:false` zetten.

☞ nu moet je je *developer's tool* erbij halen op de JS console.

Als er iets fout gaat, zal de Validator het form NIET submitten en een foutboodschap tonen in de console.

☞ probeer even: wijzig "*required*" in "*reqaired*"



De fout in *Chrome dev tool*.

Syntax-fouten echter – zoals een comma vergeten – worden niet opgevangen want dan laadt de Validator niet. Dus als je onmiddellijk doorgestuurd wordt naar *reflect_data.php* terwijl je dat niet verwacht, heb je een Javascript fout gemaakt.

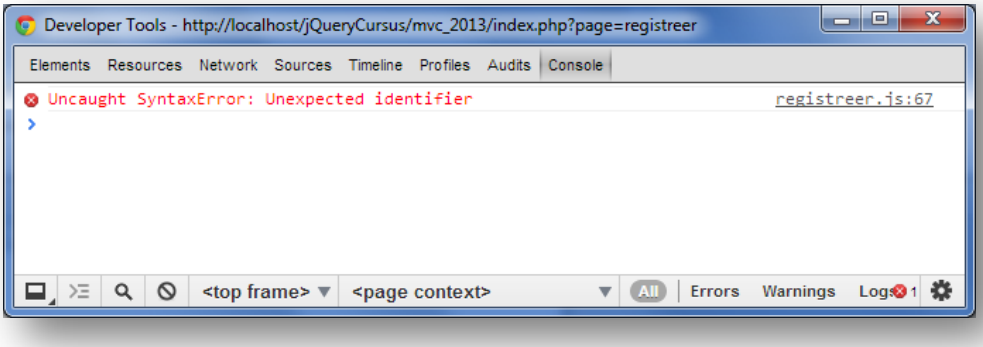
Om die te vinden moet je verhinderen dat het formulier gesubmit wordt: zo kan je misschien de foutboodschap zien in de JS console.

☞ voeg in zo'n geval deze lijn in vóór de aanroep naar `validate()`:

```
$("#regForm").submit(function(e) {e.preventDefault() })
$("#regForm").validate({
```

```
        debug:true,
…
```

Dan wordt de fout (en de lijn waarop) getoond in de console:



Eénmaal de fout gevonden, vergeet niet dit lijntje te commentariëren!

Een andere mogelijkheid is de console van je *dev tool* zo in te stellen dat de foutboodschappen "*persistent*" zijn, dus nooit gewist worden: kijk in de settings van je tool.

3.7.5 Een postnummer

Het postnummerveld moet ook aan vereisten voldoen: het moet een getal zijn van niet meer en niet minder dan 4 cijfers.

☞ Voeg toe:

```
…
rules:{
  …
  postnr: {
          required: true,
          digits: true,
          minlength: 4,
          maxlength: 4
  },
…
},
…
```

☞ en de messages:

```
…
messages:{
  …
  postnr: {
          required: "de postcode is verplicht",
          digits: "een postcode bestaat enkel uit getallen",
          minlength: "een postcodenummer bestaat uit exact 4 getallen",
```

```
        maxlength: "een postcodenummer bestaat uit exact 4 getallen"
  },
  …
},
…
```

Bespreking:

- `digits` accepteert enkel getallen
- `minlength` verwacht een minimum aantal karakters
- `maxlength` verwacht maximaal een aantal karakters
- omdat de laatste twee berichten dezelfde zijn, zouden we ze kunnen vervangen door een variabele

Straks gebruiken we nog meer ingebouwde validatieregels, niet allemaal echter, daarom kan je misschien even de volledige lijst bekijken op

http://docs.jquery.com/Plugins/Validation
en klik daar op "*List of built-in Validation methods*"

3.7.6 Een datum

Ook het geboortedatum veld moet gecontroleerd worden.

☞ dat kan met de rules `date` of `dateISO`:

```
…
rules:{
  …
  geboren:{
        required: true,
        dateISO: true
  }
  …
},
…
```

☞ en de messages:

```
…
messages:{
  …
  geboren:{
        required: "Geef uw geboortedatum in, aub",
        dateISO: "de datum moet het formaat YYYY-MM-DD hebben"
  }
  …
```

```
},
…
```

Bespreking:

- een `dateISO` opmaak gebruikt een eenvoudige *regular expression* om het patroon van een datum te checken en aanvaardt 1954/12/14 en 1954-12-14 .

  In deze oefening willen we het formaat YYY-MM-DD gebruiken om dat via PHP makkelijk te kunnen doorgeven aan de MySQL database die dit formaat gebruikt.
- er zijn andere opmaken mogelijk:
  - de `date` opmaak gebruikt infeite de Javascript `Date()` functie om te zien of het een `Date` object kan maken van de input. Een datum zoals 14/12/1954 zal aanvaard worden.
  - `DateDE` is een Duits formaat: 14.12.1954
  - de meegeleverde library *additional-methods.js* bevat nog meer formaten, o.a. een `dateNL` opmaak 14-12-1954.
    We gebruiken deze library straks
  - Geen van deze mogelijkheden garanderen 100% de correctheid van een datum (ze doen geen *sanitation check*),
    zo zullen de meeste 2011-02-31 niet herkennen als fout

Om die reden combineer je best een datumveld met een *Datepicker*, een kleine kalender *widget*, waaruit de gebruiker een datum kan kiezen

☞ test eerst even deze validatieregel uit.

3.7.7 Datepicker widget

We zullen dus gebruik maken van de jQuery UI *Datepicker* op dit veld.Dit is een UI widget die in de UI library vervat zit, we hoeven dus geen extra library te koppelen.

☞ voeg de volgende lijn toe aan de `document.ready` functie in *registreer.js*:

```
$("#geboren").datepicker();
```

☞ plaats nu de focus op het datumveld (muis of klavier): het *Datepicker* widget is actief.

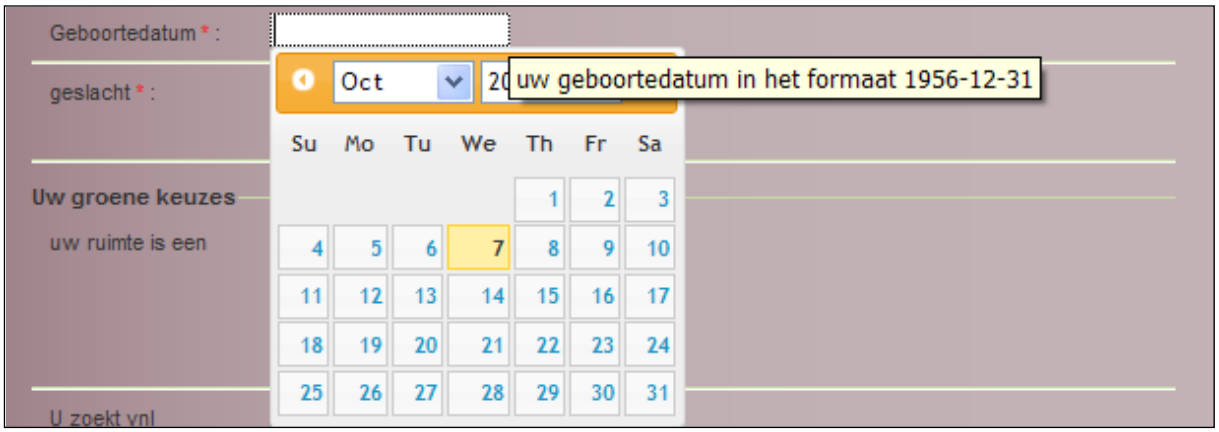
We maken enkele aanpassingen aan de functionaliteit van de *Datepicker*:

```
…
$("#geboren").datepicker({
        dateFormat: "yy-mm-dd",
        yearRange: '-80:+00',
        changeMonth: true,
        changeYear: true
});
```

```
…
```

Bespreking:

- de option `dateFormat` bepaalt de opmaak van de geproduceerde datum. Standaard is dat "mm/dd/yy"
- omdat we een geboortedatum willen, kunnen we het de gebruiker niet aandoen 60 jaar terug te klikken met de pijltjes, dus maken we gebruik van de `yearRange` optie die we op "-80:00" zetten zodat we samen met `changeYear` gemakkelijk 80 jaar terug kunnen
- `changeMonth` en `changeYear` tonen keuzelijsten respectievelijk voor maand en jaar



De *Datepicker* heeft een fenomenaal aantal opties die je toelaten hem aan te passen. Lees ze na op de jQuery UI website.

Momenteel hebben we een Engelstalige *Datepicker*. In België zijn drie taalgroepen aanwezig, we willen de mogelijkheid voorzien de widget te "localiseren".
jQuery voorziet voor enkele UI widgets "*regionalisation features*".
In vroegere versies zaten de lokale *datepicker* bestanden bij de download, in deze laatste blijkbaar niet meer.

- ☞ je kan ze vinden op http://jquery-ui.googlecode.com/svn/tags/latest/ui/i18n/
- ☞ download de bestanden *jquery.datepicker-nl-BE.js, jquery.datepicker-fr.js* en *jquery.datepicker-de.js* en sla ze op in de map *jquery/jquery-validation-1.11.1/localization*.

- ☞ open het bestand *pres/head.tpl* en voeg een script tag toe voor deze drie bestanden:

```
…
<!-- algemene JS scripts voor alle pagina's via vaste SCRIPT elementen-->
<script src="js/vendor/modernizr-2.6.2.min.js"></script>
<script src="js/vendor/jquery/js/jquery-1.10.1.min.js"></script>
<script src="js/vendor/jquery/js/jquery-ui-1.10.3.custom.min.js"></script>
<!--localisation files datepicker-->
```

```
<script type='text/javascript' src='js/vendor/jquery/↵
  jquery-validation-1.11.1/localization/jquery.ui.datepicker-nl-BE.js'></script>
<script type='text/javascript' src='js/vendor/jquery/↵
  jquery-validation-1.11.1/localization/jquery.ui.datepicker-fr.js'></script>
<script type='text/javascript' src='js/vendor/jquery/↵
  jquery-validation-1.11.1/localization/jquery.ui.datepicker-de.js'></script>
<!-- pagina-gebonden JS scripts-->
{$paginaScripts}
…
```

☞ in het bestand *registreer.js* kunnen we nu de taalinstelling voor de *Datepicker* kiezen:

```
$(function(){
        $.datepicker.setDefaults($.datepicker.regional['nl-BE']);
        $("#geboren").datepicker({
                dateFormat: "yy-mm-dd",
                yearRange: '-80:00',
                changeMonth: true,
                changeYear: true,
        });
…
```

Bespreking:

- de standaardinstellingen van de *Datepicker* kan je zetten met `$.datepicker.setDefaults(setting)`
- hier stellen we de 'regionalisatie' in op `$.datepicker.regional['nl-BE']`

De **localisation** bevat een aantal opties zoals de dag- en maandnamen, de lokale woorden voor de knoppen en het standaard datumformaat.

☞ open het bestand *ui.datepicker-nl-BE.js* om ze na te kijken.

3.7.8 Radiobuttons

Het volgende veld is een paar *radiobuttons* "man/vrouw". Initieel is geen van beide aangevinkt en dat moeten we controleren:

```
…
rules:{
        …
        sexe:"required"
        …
        },
messages:{
        …
        sexe:"kies uw geslacht"
        …
        },
```

```
…
```

Bespreking:

- een set radiobuttons controleren is in principe identiek als een tekstveld: *required*

3.7.9 Checkboxes

Het volgende veld bevat een viertal checkboxes waarvan er minstens ééntje aangevinkt moet zijn.

Het betreft hier vier `input type=checkbox` elementen die allemaal de `name` *"ruimte[ ]"* hebben. Deze `name` duidt een *array* aan dat we aan serverzijde zullen analyseren. Om ervoor te zorgen dat minstens één optie aangevinkt wordt, gebruiken we opnieuw de *required* regel:

```
…
rules:{
          …
          "ruimte[]":"required"
          …
          },
messages:{
          …
          "ruimte[]":"kies minstens &eacute;&eacute;n optie"
          …
          },
…
```

Bespreking:

- omdat de `name` van het element een paar haakjes bevat, omsluiten we die met aanhalingstekens
- gebruik indien nodig gecodeerde karakters want de tekst verschijnt in de HTML van de pagina

Als je dit uittest, merk je dat het foutbericht vlak na het eerste `input` element verschijnt (in sommige omstandigheden kan dit ook gebeuren bij de radio buttons hierboven).

We kunnen de plaatsing van ons foutbericht bepalen door middel van de `errorPlacement` property, die zetten we opnieuw als optie in de `validate()` method:

```
…,
errorPlacement: function(error,element){
    var $ctrlbx = element.parents("div.controlbox");
  if($ctrlbx.length!=0){
            error.insertAfter($ctrlbx);
```

```
    }
    else{
            error.insertAfter(element);
    }
},
submitHandler:function(form){
…
```

Bespreking:

- de `errorPlacement` optie is een **method**, een functie dus, met twee argumenten:
  - `error`: de fout zelf
  - `element`: het element die de fout opriep
- in de HTML zal je opmerken dat we sommige groepen controls in een "controlbox" (`div.controlbox`) gezet hebben. Meestal zijn dat groepen radiobuttons en checkboxes. Dit deden we om gemakkelijker het form te kunnen lay-outen.
  Daar maken we nu gebruik van om de foutboodschappen beter te plaatsen: ipv onmiddellijk na het `input` element, plaatsen we het na de "controlbox".
  - we zoeken met de jQ method `parents()` de *parentNodes* van het element af om te zien of `div.controlbox` één van de parents is. `parents()` zoekt alle parents af tot op het root element en kan een filter gebruiken.
  - als we die vinden dan plaatsten we het fout label erna, anders
  - doen we normaal en wordt de fout onmiddellijk na het formcontrol gezet

☞ controleer even de HTML-broncode met je dev tool om dit goed te begrijpen.

De plaatsing van de foutberichten wordt uiteraard ook via CSS beïnvloed.

3.7.10 Een multi-select keuzelijst

Het volgende element is een open keuzelijst waaruit meerdere opties gekozen kunnen worden.

Zoals je weet wordt het uitzicht van een `select` element bepaald door de aanwezigheid van het `multiple='multiple'` attribuut. Ook deze control wordt als een array doorgegeven: zijn `id` is "*soort_id[ ]*".

Het heeft in deze context weinig zin, maar we stellen als voorwaarde dat er minstens één soort gekozen wordt, maar ook niet meer dan vier.

```
…
rules:{
        …
        "soort_id[]":{
                required:true,
                rangelength:[1,4]
                }
```

```
        …
        },
messages:{
        …
        "soort_id[]":"kies minstens &eacute;&eacute;n soort maar niet meer dan 4",
                …
        },
…
```

3.7.11 Een gebruikersnaam en een wachtwoord

Het volgende veld, *username*, moet uiteraard ingevuld worden en bestaan uit een minimum van 8 karakters. Dit hebben we al gedaan.

Het wachtwoord moet echter *STRONG* zijn: het moet minstens 8 karakters lang zijn en er moet minstens één kleine letter, 1 Hoofdletter, 1 getal en 1 speciaal karakter (`@#$%^&+=`) in voorkomen.

Een dergelijke complexe voorwaarde stel je best samen met een *regular expression* en voorbeelden hiervan kan je vinden op het net.
Een kant-en-klare *rule* hiervoor bevat de validator niet. We moeten een eigen rule toevoegen aan de validator. Dit doen we *buiten* de `validate` method maar *binnen* de `document.ready` functie:

```
$(function(){
  …
$.validator.addMethod("wwCheck", function(value, element) {
  return value.match(/^.*(?=.{8,})(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[@#$%^&+=]).*$/);
  });
…
$("#regForm").validate({
rules:{
        …
        username:{
                required:true,
                minlength:8
                },
        ww1:    {
                wwCheck:true
                }
        …
        },
messages:{
        …
        username:"uw gebruikersnaam is verplicht en moet minimum 8 karakters hebben",
        ww1:"het wachtwoord moet min 8 karakters lang zijn en moet minstens ↵
                &eacute;&eacute;nkleine letter, 1 Hoofdletter, 1 getal en ↵
```

```
                    1 speciaal karakter (@#$%^&+=) bevatten"
        …
        },
```

Bespreking:

- `$` staat voor het `jQuery`object, dus `jQuery.validator.addMethod` kan evengoed
- met de `addMethod` method voegen we een 'custom' methode toe aan de validator: dit is een functie met de argumenten *value* en *element* die een `boolean` waarde returnt
- we maken gebruiken van de JS functie `match` om de geldigheid te toetsen aan de regular expressie

Een goede manier om er zeker van te zijn dat een gebruiker zijn eigen wachtwoord zal onthouden is het hem nog een keer te laten typen: het veld *ww2* moet identiek zijn aan *ww1*.

```
rules:{
        …
        ww1:    {
                wwCheck:true
                },
        ww2:    {
                equalTo: "#ww1"
                }
        …
  },
messages:{
        …
        ww1:"het wachtwoord moet min 8 karakters lang zijn en moet minstens één ↵
                kleine letter, 1 Hoofdletter, 1 getal en ↵
                1 speciaal karakter (@#$%^&+=) bevatten",
        ww2: "wachtwoord niet identiek"
        …
}
```

Bespreking:

- de method `equalTo(other)` verwacht dat een veld identiek is aan een ander veld. Deze method is enkel bestemd voor tekstvelden.
  Het argument *other* is een geldige JQ selector

3.7.12 Promoties via email

De checkbox *Promoties* vraagt of de klant via email op de hoogte wil gehouden worden. Het veld *email* is nu `disabled` via een vast attribuut in de HTML.

In eerste instantie moeten we ervoor zorgen dat bij het aanvinken van *promos, email* opnieuw actief wordt en vice versa.

Dat doen we via een kort scriptje **buiten** de validator.

☞ Schrijf het volgende boven de *validate()* method:

```
$('#promos').click(function(){
    if ($(this).is(':checked')) {
           $('#email').removeAttr('disabled')[0].focus();
    } else {
           $('#email').attr('disabled', true).val("");
  }
  })
```

Bespreking:

- via de eventhandler `click()` controleren we of `this` (= de `#promos`checkbox), aangevinkt is of niet met de `is()` method.
  De selector `:checked` controleert dit
- indien aangevinkt: verwijderen we dit attribuut en plaatsen de focus op het *email* veld. Omdat `focus()` een javascript method is, halen we het eerste (en enige) item uit de *wrapped set* met `[0].` Dat kan onmiddellijk na de `removeAttr()` method omdat de oorsponkelijke set steeds gereturned wordt
- indien niet, zorgen we ervoor dat het `disabled` attribuut opnieuw in het `input`veld staat
- we legen ook het veld met `val()` en een lege String.
  Dit is niet strikt noodzakelijk want `disabled` velden geven hun waarde nooit door aan de server

☞ Probeer dit even uit.

Nu moeten we nog zorgen voor validatie regels voor het email veld
Het veld *email* is enkel verplicht als promos aangevinkt is, daarom de volgende rules:

```
…
$("#regForm").validate({
  rules:{
        …
        email:{
               required: "#promos:checked",
               email:true
               }
        },
  messages:{
        …
        email:{
               required: "Een emailadres is nodig om u te kunnen
                      contacteren",
               email: "het emailadres is ongeldig"
               }
        },
  submitHandler: function(form) {
```

```
            form.submit();
        }
  });
…
```

Bespreking:

- de `required` rule voor *email* gebruikt een ***dependency expression***: afhankelijk van de `boolean` waarde van dit statement is `required` actief of niet. Dus als `#promos` aangevinkt is, is *email* required
- de `email` rule controleert vanzelf de geldigheid van het ingevulde emailadres
- de `messages` volgen deze twee regels

3.7.13 Foutboodschappen in een aparte container

Een andere manier om foutboodschappen te tonen, vooral interessant als er veel kunnen zijn, is ze allemaal verzamelen in één container en deze onderaan of bovenaan het formulier tonen.

We kunnen onze validatie met enkele eenvoudige stappen op die manier omvormen.

Eerst en vooral voegen we twee container elementen toe aan de HTML van de pagina.

☞ vul aan in de functie *getRegistreer()* van *inhoud.php.* De eerste container komt net boven het formulier:

```
…
$str .= "<section>
         <!--start foutBox -->
         <div class='foutBox' id='fouten'><h2>Fouten</h2><ul></ul></div>
         <!--einde foutBox -->
         <form id='regForm' name='regForm' method='get' action='reflect_data.php' >
                <fieldset>
                <legend>Uw persoonlijke gegevens</legend>
…
```

☞ de tweede net onder het `form` element:

```
…
</form>
<!--start foutBox-->
  <div class='foutBox'><a href='#fouten'>sommige ingevulde gegevens zijn
  foutief!</a></div>
  <!--einde foutBox-->
";
return $str;
}
…
```

Het stylesheet *plantenshop.css* bevat reeds een aantal *rules* om de `class.foutbox` en zijn inhoud op te maken.

☞ in *registreer.js* commentariëren we ook het gedeelte betreffende de plaatsing van de *error labels*:

```
…
/*
            errorPlacement: function(error,element){
                    var $ctrlbx = element.parents("div.controlbox");
                    console.log($ctrlbx.length);
                    if($ctrlbx.length!=0){
                            error.insertAfter($ctrlbx);
                    }
                    else{
                            error.insertAfter(element);
                    }
            },
  */
…
```

☞ en vervangen het door enkele rules in de `validate()`:

```
…
  errorContainer: $foutBoksen,
  errorLabelContainer: $("ul", $foutBoksen),
  wrapper: "li",
  submitHandler:function(form){
                form.submit();
        }
  }); //einde validator
…
```

De variabele *$foutBoksen* moeten we echter nog declareren.

☞ dat doen we buiten, net vóór de `validate()` method:

```
…
var $foutBoksen = $('div.foutBox');
$('#regForm').validate({
…
```

Bespreking:

- de `errorContainer` property is de set container(s) waar de fouten getoond zullen worden. Er kunnen dus meer dan één containers zijn.

- de `errorLabelContainer` is het `element`binnen één van de `errorContainer(s)` dat de foutboodschappen zal ontvangen, hier een `ul` element
- elk label wordt nog eens in een `wrapper` – een `li` element - geplaatst

Als resultaat krijg je nu zowel onderaan als bovenaan een waarschuwing en in de bovenste container zitten de boodschappen:

Ook hier wordt "*eager*" gevalideerd: zodra een fout opgelost is, verdwijnt het betreffende bericht uit de *errorContainer*.

3.7.14 Dialog en Button widgets

Het perfecte formulier is zo duidelijk dat de gebruiker geen fouten kán maken...(Confusius)

Phww… die Confusius wist alles… we doen ons best!

Een *voornaam* en een *familienaam* zijn velden die voor zich spreken maar de vereisten voor gebruikersnamen en wachtwoorden zijn minder vanzelfsprekend.

Daarom is ons formulier ideaal om de UI **Dialog** widget eens uit te proberen. De library is reeds gekoppeld dus we kunnen onmiddellijk aan de slag.

De **Dialog** werkt volgens het volgende principe:

- een `div` element met `id` bevat de tekst en titel van het bericht.
  Deze `div` is reeds aanwezig in de pagina (maar zou ook dynamisch kunnen aangemaakt worden)
- op deze `div` wordt de method `dialog()` uitgevoerd:
  een eerste gevolg hiervan is dat de `div` bij het starten niet zichtbaar is
- een *event handler* gekoppeld aan een hyperlink, knop of nog iets anders, opent het dialoogvenster met de inhoud van de `div`

De tekst voor deze dialog hebben we al voorzien in een functie *getUsernameDialog()* die je kan vinden in *inhoud.php*. Die hoef je dus niet meer te typen.

☞ zoek deze functie:

```
…
function getUsernameDialog(){


  $str = "<!-- ui-dialog -->
  <div id='dialog_username' class='dialoogvenster' title='Uw gebruikersnaam'>
  <p>Uw gebruikersnaam is een unieke naam waarmee u wil aanmelden op het systeem. <br
  />andere gebruikers kennen je onder die naam, niet je echte naam. Die enkel bekend
  bij de beheerder van de website</p>
  <p>Kies bij voorkeur een gemakkelijke naam waarmee je je kunt identificeren,
  bijvoorbeeld <i>'jean smits'</i>, <i>'lieselot'</i> of <i>'Superwoman'</i></p>
  <p>De gebruikersnaam moet uniek zijn, als iemand anders ze al heeft zal je gevraagd
  worden een andere te kiezen</p>
          </div>
  <!-- ui-dialog -->";
return $str;
}…
```

Let op het volgende:

- de `div` heeft een `id` om te kunnen identificeren
- hij heeft ook een `class` om makkelijk te kunnen scripten en stylen.
  Het algemene stylesheet *plantenshop.css* bevat ook hier reeds een aantal rules om het uitzicht van het dialoogvenster en zijn link te bepalen
- hij heeft een `title` attribuut dat gebruikt wordt in de dialog

Het enige dat we moeten doen is deze HTML in de pagina bijvoegen. De plaats waar je dat doet is infeite onbelangrijk vermits de dialog initieel onzichtbaar zal zijn en slechts tevoorschijn komt op verzoek.

☞ voeg in de functie *getRegistreer()* het volgende lijntje toe helemaal achteraan (voor het gemak):

```
…
<div class='foutBox'><a href='#fouten'>sommige ingevulde gegevens zijn
  foutief!</a></div>";
$str .= getUsernameDialog();


  return $str;
}
```

☞ we plaatsen nu ook een hyperlink bij het overeenkomstige tekstvak (*gebruikersnaam*):

```
…
$str .= "<fieldset>
          <legend>Aanmelding</legend>
```

```
        <div>
        <label for='username'>gebruikersnaam<abbr ⮵
                    class='verplicht' title='verplicht'>*</abbr>:
        <!--start dialogbutton-->
        <br><a href='#' id='dialog_link_username'>Meer uitleg</a>
        <!--einde dialogbutton -->
        </label>
        <input type='text' title='de gebruikersnaam waarmee u wil aanmelden'⮵
                        id='username' name='username'  />
        </div>
…
```

Deze hyperlink heeft een `id` waarmee we een event handler zullen koppelen en waarmee we ook de **Button** widget erop zullen toepassen.

Nu hebben we nog een JQ scriptje nodig om het te doen werken.

☞ plaats deze code in de `document.ready` handler van *registreer.js*:

```
…
  //alle dialoogvensters: instellingen
  $(".dialoogvenster").dialog({
        autoOpen: false ,
        buttons: {
              "Ok": function() { $(this).dialog("close"); }
        },
        modal:true,
        width: 600
  });

  // de dialoog Button
  $('#dialog_link_username')
        .button({icons: {secondary: "ui-icon-help"}})
        .click(function(e){
              e.preventDefault();
              $('#dialog_username').dialog('open');

  });
…
```

We bespreken eerst de werking van de hyperlink:

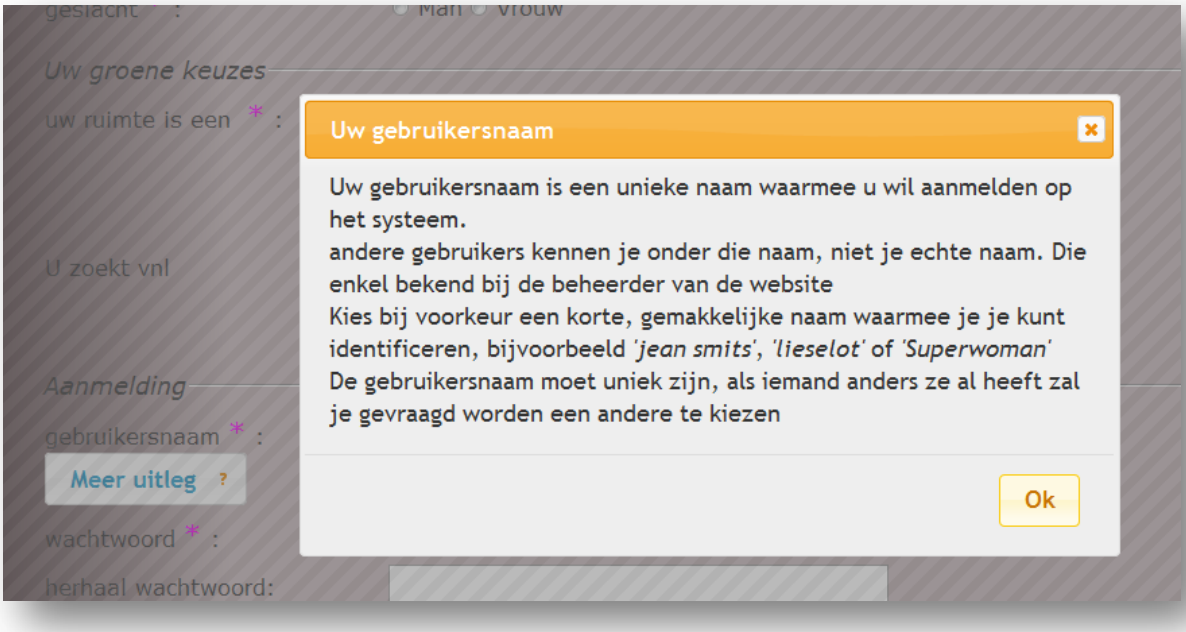
- we vinden de hyperlink via zijn `id#dialog_link_username`
- eerst passen we er de **Button** widget op toe:
  - we gebruiken eenvoudigweg de method `button()` op de link: die verandert hem in een knop

  - we passen de optie `icons` toe, die zelf een object bevat met de property `secondary`: dat is een ikoontje ná de tekst.
- nu chainen we met een *event handler* `click` die het dialoogvenster *#dialog_username*opent door middel van de method `dialog("open")`
- `e.preventDefault()`neutraliseert de *default action* van de hyperlink, zodat hij verder niet doorklikt

De method `dialog():`

- de method `dialog()` wordt uitgevoerd op de wrapped set *$(".dialoogvenster")*
- de gebruikte opties voor deze method zijn:
  - `autoOpen: false` zorgt ervoor dat hij niet vanzelf opent
  - `buttons`: is een **object** met de knoppen die getoond moeten worden.: hier enkel een "OK" knop.
    - De property *key* wordt ook de tekst op de knop ("OK").
    - De *callback function* van de *key* is de handeling die opgeroepen wordt, hier wordt het dialoogvenster gesloten
    - Het `$(this)` object in deze functie is het dialoogvenster zelf
  - `modal:true` zorgt voor het **modaal** openen van het venster: andere items op de pagina worden tijdelijk onbereikbaar tot op de OK knop geklikt wordt
  - `width`: bepaalt de breedte van dit venster. Kan ook met CSS bepaald worden

En dit alles geeft dit resultaat:



3.8 Plugin: Lightbox 2

Eén van de meest populairste plugins om foto's te tonen, werd oorspronkelijk ontwikkeld voor *scriptaculous* en *prototype*, maar is in de laatste versies ook beschikbaar voor *jQuery*: *Lightbox2*

Er bestaan veel *Slideshow* plugins, welke kiezen? Waar moet je op letten?

1. ***doet de plugin wat je ervan verlangt?***
   Is de belangrijkste vraag: de plugin **moet doen wat jij wil**. Toeters en bellen zijn mooi maar als je ze niet kunt gebruiken, neem een andere.
   Bekijk de demo's, lees de feedback

2. ***is de plugin licht?***
   zijn de bestanden niet te groot? de optelsom van alle plugins die je gebruikt kan serieus beginnen wegen!

3. ***is ze gemakkelijk configureerbaar? is er documentatie?***
   sommige plugins hebben weinig of geen uitleg. *Lightbox* is op het randje. Voor veel programmeurs is documentatie teveel gevraagd …

4. ***is de plugin volwassen en wordt ze geüpdate voor recente jQ versies?***
   bestaat hij al een tijdje? zijn er al verschillende versies van geweest? en wordt hij nog steeds geüpdate?

Eén van de meest voorkomende problemen met plugins is dat ze gebouwd zijn op een ***bepaalde jQuery versie*** *en niet geüpdate worden voor de nieuwere versies. Dit probleem zullen we hier ook ervaren én proberen op te lossen.*

3.8.1 Downloaden en koppelen

Op de homepage

http://lokeshdhakar.com/projects/lightbox2/

bemerk je dat deze versie (*Lightbox 2.51*) gebouwd is op jQuery 1.7.2 (bijgeleverd). Wij werken echter met jQuery 1.10 en dat willen we blijven doen, maar helaas werkt deze versie niet met jQuery 1.10.x (we besparen je hier enkele uren hoofdbrekens…)

Wat zoeken op het net verduidelijkt dat we de eersten niet zijn die dit probleem ervaren wat ons leidt naar **Github** waar iemand een *fork* maakte van het oorspronkelijke project om het probleem op te lossen:

https://github.com/careilly/lightbox2

- ☞ surf ernaartoe,klik naast *branch:master* het keuzelijstje open en kies de *branch:update_for_jquery_1.9.0*
- ☞ klik bovenaan op ZIP om deze *branch* te downloaden
- ☞ extraheer de bestanden:je krijgt een map *lightbox2-update_for_jquery_1.9*
- ☞ wijzig de map naam naar *lightbox*
- ☞ wis de submappen *coffee, sass, releases*

- plaats/kopieer de map *lightbox* in *js/vendor/jquery.*
- De map *lightbox/js* bevat ook nog eens een copy van *jquery* en *jquery-ui*: die mag je wissen, we zullen koppelen naar de eigen library.

Als je meer wil weten over Git en Github, lees dan één van de vele tutorials op het net

Dee oplossing kan overbodig worden als een nieuwe versie van Lightbox wél gaat werken met jQ 1.10 of 2: in dat geval meld dit aan je coach.

3.8.2 Toepassen op bestaande HTML

We zullen de plugin toepassen op onze *galerij* pagina, die nu een aantal *thumbnails* toont met onderschriften. Bekijk de HTML code van de thumbnails: ze bestaan telkens uit een `figure` element, met een `a` element, met daarin een `img`. Elke `figure` heeft ook een `figcaption`. De hyperlinks koppelen rechtsreeks aan de grote versie van een plantenfoto.

- zoek deze inhoud ook op in de pagina *inhoud.php* bij de functie *getGalerij*

Alle foto's van de planten bevinden zich in de map *images/planten*. Van elke plant is een grote foto en een thumbnail aanwezig.

Omdat de plugin enkel op de *galerij* pagina gebruikt zal worden zullen we hem ook enkel laden op die pagina.

- open *index.php* en pas de case "*galerij*" aan als volgt:

```
…
$tpl['paginaScripts']          = ↵
  getScriptElements(array("js/vendor/jquery/lightbox/js/lightbox.js"));
$tpl['paginaStylesheets']      =↵
  getLinkElements("js/vendor/jquery/lightbox/css/lightbox.css");
break;
```

Bespreking:

- we laden het lightbox script en het bijhorende stylesheet
- vermits jQuery geladen wordt via het template zal die eerst gekoppeld worden vóór de plugin, een voorwaarde

De plugin kan nu eenvoudig toegepast worden op een bestaande hyperlink door er een `rel="lightbox"` attribuut in te plaatsen.

- in de functie getGalerij in inhoud.php wijzig de eerste hyperlink als volgt:

```
<figure id='fig1'>
  <a href='images/planten/lupine.jpg' rel='lightbox' title='Kenai Peninsula Lupine,
  prachtige lentebloeier'>
<img src='images/planten/th_lupine.jpg' /></a>
<figcaption><b>Kenai Peninsula Lupine</b><br> prachtige lentebloeier</figcaption>
</figure>
```

En test uit door eens op de eerste thumbnail te klikken: de foto opent in Lightbox.
Bemerk dat het `title` attribuut van de hyperlink getoond wordt als onderschrift van de foto.

Zo kunnen we losse beelden tonen, maar we willen een soort slideshow hebben waarbij we doorheen alle beelden kunnen laveren als een set. Om dit te bereiken gebruiken we een arraynaam in het `rel` attribuut. Die naam is zelf te kiezen.

- ☞ wijzig nu alle hyperlinks als volgt:

```
<figure id='fig1'>
  <a href='images/planten/lupine.jpg' rel='lightbox[planten]' title='Kenai Peninsula
  Lupine, prachtige lentebloeier'>
<img src='images/planten/th_lupine.jpg' /></a>
<figcaption><b>Kenai Peninsula Lupine </b><br>prachtige lentebloeier</figcaption>
</figure>
…
```

En probeer opnieuw. De slideshow is klaar.
Zo kan je meerdere fotosets definiëren op dezelfde pagina.

3.8.3 Opties

In tegenstelling tot de jQuery UI widgets kan je deze plugin niet configureren via een startfunctie. We zijn verplicht de plugin code zelf te wijzigen.

- ☞ open *lightbox.js*
- ☞ zoek de `LightboxOptions` (lijn 52)
- ☞ wijzig de overgangstijden:

```
this.resizeDuration = 300; //700
      this.fadeDuration = 200; //500
```

- ☞ wijzig de tekst:

```
this.labelImage = "Foto"; //"Image"
this.labelOf = "van";  //"of"
```

Test uit.

3.9 Plugin: DataTables

3.9.1 Serverside vs clientside programmeren

In de klassieke (*we bedoelen 'oude'*) manier van webapplicaties opzetten werd de volledige inhoud van een pagina geleverd door een serverside programma (vb. PHP), terwijl de clientside (Javascript) een minieme rol werd toebedeeld.

De huidige werkelijkheid ligt veel meer in het midden en afhankelijk van de manier waarop de applicatie opgezet is, gaat dat van 2/3 serverside + 1/3 clientside zijn, of ook 1/3 serverside + 2/3 clientside bij Ajax applicaties.

In dit hoofdstuk zullen we demonstreren hoe de serverside (PHP) enkel de ruwe gegevens levert terwijl de client-side de verwerking in de HTML voor zijn rekening neemt. Dit doen we op onze *shop* pagina.

Die pagina bestaat grotendeels uit een tabel van gegevens met erboven wat controls om een selectie te maken. De data wordt geleverd door een PHP script dat ook het `table` element aanmaakt. Dit is de klassieke manier van werken: de serverside controller maakt de HTML aan, toch moeten we ons afvragen of het aanmaken van de `table` niet aan een client-side script overgelaten moet worden?

3.9.2 Download en installatie

We maken gebruik van de krachtige jQuery plugin "***dataTables***" die alles in huis heeft om tabellen client-side te bewerken.

Deze plugin is zeer uitgebreid en ook zeer goed gedocumenteerd. We raden je daarom aan om de documentatie op de website bij de hand te houden en wat we doen even na te checken. Ga echter niet alles lezen want dan ben je zoet voor enkele weken…

De plug-in is te vinden op *http://datatables.net*

- download de laatste versie
- pak de zip file uit en verplaats de map *DataTables-1.9.4* onder *js/vendor/jquery/*
- enkel de submap *media* is belangrijk

Nu koppelen we de nodige library bestanden in *index.php*.

- zoek de `case` "shop"
- in de variabele `$tpl['paginaScripts']` werd reeds *shop.js* geladen. Voeg daar nu dataTables aan toe
- voeg een variabele `$tpl['paginaStylesheets']` toe om het dataTable stylesheet te laden

```
…
case "shop":
  /*** Planten pagina, PHP, non-ajax ***/
  //init zoekvariabelen
  $soort_id      =(isset($_GET['soort_id']))?$_GET['soort_id']:'%';
  $kleur         = (isset($_GET['kleur']))?$_GET['kleur']:'%';
  $hoogte_min    = (isset($_GET['hoogte_min']))?intval($_GET['hoogte_min']):0;
  $hoogte_max    = (isset($_GET['hoogte_max']))?intval($_GET['hoogte_max']):5000;
  $tpl['title']  = "de Plantenshop: ons aanbod";
```

```
  $tpl['body_id'] = "shop";
  //content
  $tpl['rechts'] = getPlanten($soort_id, $kleur, $hoogte_min, $hoogte_max);
  $tpl['paginaScripts'] = getScriptElements(array(
         "js/vendor/jquery/Datatables-1.9.4/media/js/jquery.dataTables.min.js",
         "js/shop.js"));
  $tpl['paginaStylesheets'] = getLinkElements("js/vendor/jquery/
                        Datatables-1.9.4/media/css/jquery.dataTables.css");

  break;
…
```

Opmerking:

- bemerk dat we de PHP functie `array()` gebruikt hebben om op die manier meerdere scripts te kunnen laden. Let op de comma!

3.9.3 Activering

☞ open nu het bestand *shop.js* en maak de volgende toevoeging onderaan de `document.ready` functie, dus ná alle code over *adv zoeken/verbergen* en de *slider*:

```
//code adv zoeken/verbergen, slider
…
//datatables
  $("#plantenlijst").dataTable();

});//einde doc ready
```

Je bemerkt onmiddellijk dat er iets gebeurt met de tabel:

Bespreking:

- *dataTable* werkt op een geldig gestructureerde HTML `table` met verplichte `thead` en `tbody` tags. Een `tfoot` sectie is optioneel.
- het `table` element wordt geïdentificeerd via een JQ selector, zo kan je één of meerdere tabellen sorteerbaar maken

3.9.4 Opties

De plugin *dataTable* heeft een groot aantal opties waarvan de meeste standaard actief zijn.

- ***Paginering***:
  er worden slechts 10 records getoond, er is een keuzelijstje om er meer te tonen en onderaan de tabel hebben we controls om de rest van de records te zien.
- ***Sorteerbaar***:
  klik op een kolomtitel en je bemerkt dat er gesorteerd wordt
- ***Filter***:
  er is een "*Search"* vak: typ "gras" in het zoekvak en je ziet enkel de planten met gras in hun naam
- ***Opmaak*** van de rijen van de tabel. Een kijkje via een *webdeveloper tool* toont ons dat de rijen alternerend een class "*odd*" of "*even*" hebben.

In de volgende stappen zullen we de meest courante opties wat nader bekijken. Bedenk dat deze plugin véél meer kan dan we in deze cursus laten zien: wil je meer, dan is het aan jou om de documentatie op de website door te nemen.

Een overzicht van de opties vind je in de doc op USAGE - OPTIONS

De instellingen kunnen bepaald worden via een *opties* object in de `dataTable()` method zoals we reeds zagen in verschillende widgets.

Paginering:

☞ veronderstel dat je geen *Paginering* wil

```
…
$("#plantenlijst").dataTable({
  "bPaginate": false
});
…
```

Alle records worden getoond.

- de `boolean` waarde van de property `bPaginate` controleert het pagineren. Laat je de property weg dan wordt standaard gepagineerd
- merk ook op dat we alle properties in "aanhalingstekens" gezet hebben. We volgen hierbij de documentatie van de plugin, maar aanhalingstekens zijn infeite enkel nodig als er speciale karakters gebruikt worden

☞ zet de property *bPaginate* op `true` of verwijder

We krijgen opnieuw paginering en nu stellen we een aantal parameters in die ermee te maken hebben:

```
…
$("#plantenlijst").dataTable({
  "bPaginate": true,
  "iDisplayLength": 20,
  "iDisplayStart": 20,
  "sPaginationType": "full_numbers",
  "aLengthMenu": [[10, 25, 50, -1], [10, 25, 50, "Alle records"]]
});
…
```

Bespreking:

- vergeet de komma's tussen de properties niet!
- merk op dat alle properties in *dataTable* via hun naamgeving (eerste letters) aangeven welk **type** variabele verwacht wordt: zo is `bPaginate` een `boolean`, terwijl `iDisplayLength` een `integer` getal bevat
- `iDisplayLength` bepaalt het aantal records dat op één "pagina" te zien is. Dat kan uiteraard gewijzigd worden via het keuzelijstje bovenaan
- `iDisplayStart` zet de startpositie één record verder dan het opgegeven getal: hier initialiseert de tabel op de tweede pagina van 20 records
- `sPaginationType` heeft slechts twee mogelijke waarden: "*full_numbers*" en "*two_button*", de standaard instelling."*full_numbers*" zorgt voor een reeks paginanummers samen met pijltjesknoppen.

- `aLengthMenu` bepaalt de keuzes in het "*Show entries*" lijstje via een array met twee subarrays. Het eerste subarray bevat de values van de `option` elementen in de keuzelijst, het tweede de teksten die te zien zullen zijn.
  De value `-1` zorgt voor een volledige lijst, alle records dus

Sorteren:

Sorteren is standaard actief, je bemerkt het als je op één van de kolomkoppen klikt.

☞ om de optie te desactiveren gebruiken we `bSort`:

```
…
$("#plantenlijst").dataTable({
  "bPaginate": true,
  "bSort":false,
  "iDisplayLength": 20,
  "iDisplayStart": 20,
  "sPaginationType": "full_numbers",
  "aLengthMenu": [[10, 25, 50, -1], [10, 25, 50, "Alle records"]]
});
…
```

Sorteren kan nu niet meer.

☞ zet `bSort`op `true` of verwijder.
☞ commentarieer ook de `iDisplayStart` property zodat je de records vanaf de eerste ziet.

De records zijn nu in de tabel aanwezig zoals ze geleverd worden door het server script, in dit geval is dat gesorteerd op *soort*. Je ziet ook dat deze (eerste) kolom een andere kleur heeft dan de andere kolommen.

We stellen nu een aantal opties in die het initieel sorteren beïnvloed:

```
$("#plantenlijst").dataTable({
        "bPaginate": true,
        "bSort": true,
        "iDisplayLength": 20,
        /*"iDisplayStart": 20,*/
        "sPaginationType": "full_numbers",
        "aLengthMenu": [[10, 25, 50, -1], [10, 25, 50, " Alle records "]],
        "bProcessing": true,
        "aaSorting": [[6,'asc'], [2,'desc']]
})
```

Bespreking:

- `bProcessing` bepaalt of er een "Processing…" bericht te zien zal zijn tijdens het sorteren. Dit kan de gebruiker gerust stellen als er een groot aantal records gesorteerd moeten worden. Hier zal je dat waarschijnlijk niet eens opmerken

- `aaSorting` is een array van subarrays. Elk subarray stelt een kolom voor: het bevat zijn indexnummer (0 = eerste) en de manier waarop gesorteerd moet worden

Nu wordt de tabel eerst gesorteerd op de laatste kolom (*rubriek*) en daarbinnen op *hoogte*. Deze sortering gebeurt enkel bij de start, je kan hem op elk moment wijzigen door zelf op een kolom te sorteren.

☞ gebruik nu je *developer tool* om eens te kijken naar een `td` element van één van de gesorteerde rijen: je bemerkt dat deze elementen een `class` hebben zoals "*sorting_1*","*sorting_2*",…

Deze classes geven – in combinatie met de `tr` classes "odd" en "even" - de iets donkerder kleur aan de kolom. Je kan ze bekijken in *jquery.dataTables.css*.

Kolommen

Individuele kolommen kunnen beïnvloed worden op twee manieren: met de properties `aoColumns` of `aoColumnDefs`**.**

- `aoColumns` is een array van objecten met gegevens voor elke kolom.
  Je bent **verplicht alle kolommen** in dit array te plaatsen, in de juiste volgorde! desnoods met een `null` waarde
- als je `aoColumnDefs` gebruikt kan je er enkel die kolommen in zetten die je wil veranderen. Ook deze property is een array van objecten. Om aan te duiden over welke kolom het gaat, gebruik je de `aTargets` property: een array van indexgetallen of andere waarden

In de volgende aanpassingen gebruiken we `aoColumnDefs` omdat deze manier iets flexibeler is.

```
$("#plantenlijst").dataTable({
        "bPaginate": true,
        "bSort": true,
        "iDisplayLength": 20,
        /*"iDisplayStart": 20,*/
        "sPaginationType": "full_numbers",
        "aLengthMenu": [[10, 25, 50, -1], [10, 25, 50, " Alle records "]],
        "bProcessing": true,
        "aaSorting": [[6,'asc'], [2,'desc']],
        "aoColumnDefs": [
              { "bVisible": false, "aTargets": [ 5 ] },
              { "bSortable": false, "aTargets": [ 2, 6 ] },
              { "asSorting": [ "desc" ], "aTargets": [ 3 ] },
              { "bSearchable": false, "sTitle": "Rubriek", "aTargets": [ 6 ] },
              { "sTitle": "Lengte", "sWidth": "5%", "aTargets": [ 2 ] },
              { "sClass": "dt_fluo", "aTargets": [ 0 ] }
  ]
```

```
})
```

Bespreking:

- **in alle objecten** bemerk je de eigenschap `aTargets`: in dit array staat op welke kolom(men) de wijzigingen betrekking hebben
- `bVisible`: met `false` kan je een kolom verbergen, hier de 6de kolom (*prijs*)
- `bSortable`: met `false` zorg je ervoor dat een kolom niet langer sorteerbaar is, ondanks de oorspronkelijke instelling, hier de 3de en de 7de kolom (hou er rekening mee dat er nu reeds één kolom verborgen is)
- `aSorting['asc','desc']`: om de sorteervolgorde te controleren in een kolom, voor de 4de kolom (*hoogte*) zorgen we dat die enkel dalend gesorteerd kan worden
- `bSearchable`: met `false` zorg je ervoor dat de zoekfunctie niet in deze kolom gaat zoeken. Hier verhinderen we het zoeken in "Rubriek", zodat bij het zoeken naar de tekst "*boom*" niet de hele rubriek "*boom*" meegenomen wordt
- `sTitle` laat je toe een kolomtitel alsnog te veranderen, hier krijgt de kolom "hoogte" de titel “Lengte”
- `sWidth` (CSS waarde): bepaalt de breedte van een kolom, hier dezelfde kolom
- `sClass`: geeft een `class` attribuut waarde aan elke cel van de kolom, hier de `class` "*dt_fluo*" aan de eerste kolom. Wij hadden deze `class` reeds staan in ons stylesheet

Taal

Momenteel zijn alle opties en teksten in het Engels. Alle teksten kunnen gewijzigd worden – en dus in een andere taal gezet d.m.v. het `oLanguage` object.

De kortste manier is echter een apart taalbestandje te koppelen. Je vindt het op de website van *dataTables* bij *PLUG-INS -INTERNATIONALISATION,* dan *DUTCH*.

Kopieer de tekst en sla op in een bestandje *datatables.nederlands.txt* dat je in de *js* map zet:

```
{
    "sProcessing": "Bezig...",
    "sLengthMenu": "_MENU_ resultaten weergeven",
    "sZeroRecords": "Geen resultaten gevonden",
    "sInfo": "_START_ tot _END_ van _TOTAL_ resultaten",
    "sInfoEmpty": "Geen resultaten om weer te geven",
    "sInfoFiltered": " (gefilterd uit _MAX_ resultaten)",
    "sInfoPostFix": "",
    "sSearch": "Zoeken:",
    "sEmptyTable": "Geen resultaten aanwezig in de tabel",
    "sInfoThousands": ".",
    "sLoadingRecords": "Een moment geduld aub - bezig met laden...",
    "oPaginate": {
```

```
        "sFirst": "Eerste",
        "sLast": "Laatste",
        "sNext": "Volgende",
        "sPrevious": "Vorige"
    }
}
```

Je kan uiteraard je eigen teksten gebruiken.
Voeg daarna de `oLanguage.sUrl` property toe:

```
$("#plantenlijst").dataTable({
        /*
        "bPaginate": false,
        bSort: false,
        */
        "iDisplayLength": 20,
        /*"iDisplayStart": 20,*/
        "sPaginationType": "full_numbers",
        "aLengthMenu": [[10, 25, 50, -1], [10, 25, 50, " Alle records "]],
        "bProcessing": true,
        "aaSorting": [[7,'asc'], [3,'desc']]
        "aoColumnDefs": [
              { "bVisible": false, "aTargets": [ 0 ] },
  { "bSortable": false, "aTargets": [ 2,6 ] },
              { "asSorting": [ "desc" ], "aTargets": [ 3 ] },
              { "bSearchable": false,"sTitle": "Rubriek", "aTargets": [ 7 ] },
              { "sTitle": "Soort", "sWidth": "35%", "aTargets": [ 1 ] },
              { "sClass": "dt_fluo", "aTargets": [ 1 ] },
              { "fnRender": function ( oObj ) {
                    return oObj.aData[4] +'-'+ oObj.aData[5];},
                    "bUseRendered": false,
                    "aTargets": [4]
                    }
  ],
        "oLanguage": {"sUrl": "js/vendor/jquery/
                    Datatables-1.9.4/media/js/datatables.nederlands.txt"}
  })
```

Vanaf nu zijn de controls en de berichten in het Nederlands.

Tot zover de meest courante instellingen van de dataTable.

Tenslotte willen we nog opmerken dat deze versie van de *plantenshop* op de **klassieke server-side manier** werkt: de server produceert alles: de pagina, het table element en de gegevens er in.

Kies je andere voorwaarden in het formulier dan stuur je opnieuw een *http* verzoek naar de server en je krijgt opnieuw een volledige pagina met alles erop en er aan. *jQuery* en *dataTables* komen pas in werking **nadat** alles ingeladen is.

We gebruiken de plugin hier dus eerder voor wat "cosmetische ingrepen" achteraf.

Straks vormen we de pagina echter om tot een **Ajax versie** waarbij *dataTables* ook het ophalen van gegevens en het aanmaken van de table voor zijn rekening neemt.

4 Ajax

Ajax is een techniek om via Javascript externe gegevens, **asynchroon** op te halen, zonder de volledige pagina te moeten herladen. Javascript maakt gebruik van het `XMLHttpRequest`object (XHR), waarvan het gebruik complex is.

Met jQuery echter wordt het opzetten van een Ajax-applicatie toch wat gemakkelijker.

In tegenstelling tot wat de naam doet vermoeden kan je met XHR méér dan enkel XML gegevens ophalen: ook JSON, HTML en pure tekst kunnen naast XML doorgegeven worden. Het is daarom aan te raden dat je de volgende hoofdstukken in de *Javascript theorie* cursus eens opnieuw naleest als je er niet mee vertrouwd bent:

- JSON
- XML primer
- Ajax

Als voorbeeld van een Ajax applicatie verwijzen we je naar een typische webshop: www.vandenborre.be. Ga daar naar een rubriek, bv. de *Tablets* pagina. Links heb je allerlei checkboxes om je keuze te versmallen, vb. met *Android* als besturingssyteem. Telkens je een optie aan/afvinkt wordt een Ajaxcall gemaakt naar de server en komt er een JSON object als response, zonder dat de volledige pagina vernieuwd wordt.

4.1 De Ajax functies

jQuery heeft één algemene functie die **alles kan** wat er maar mogelijk is op gebied van Ajax calls: `$.ajax()`.

Alle andere Ajax functies zijn van deze afgeleid: `.load()`,`$.get()`, `$.getJSON()`,`$.post()`, `$.getScript()`, …

Deze afgeleide functies zijn dan specifiek gericht op één bepaald aspect/doel, zo is de functie `$.getJSON` specifiek bedoeld voor het ophalen van JSON data via een Ajax call. Je hebt de functie `$.ajax()`dus meestal niet nodig omdat er een meer gespecialiseerde afgeleide functie bestaat.

4.2 Het team met JSON data

In het hoofdstukje 3.3.13 "*een keuzelijst aanmaken"* maakte je een *dropdown* aan voor de teamleden van onze shop. Daarbij werd ook dynamisch een lege container `div#teamgegevens` aangemaakt.

- ☞ ga naar de pagina "*Over ons*"
- ☞ open opnieuw het bestand *about.js*

Als er een teamlid uit de keuzelijst geselecteerd wordt, willen we dat zijn gegevens rechts in de `div#teamgegevens` verschijnt.

Die gegevens komen van de server, in JSON vorm, geleverd door het PHP script *ajax_json_team.php*. Hoe dit script deze gegevens aanmaakt, is niet ons probleem: de back-end programmeur weet dat hij die gegevens moet leveren voor een Ajax call, hoe hij dat doet valt buiten de scope van deze cursus.

Ajax applicaties kunnen enkel getest worden als je de pagina vanaf een ***webserver*** *bekijkt, dus bv. via http://localhost/plantenshop/...., NIET via file://*

☞ we proberen even: ga via je webserver eerst naar de 'Over ons' pagina en vervang dan in de adresbalk, de *index.php?page=about* door *services/ajax_json_team.php*, dus

http://localhost/plantenshop/services/ajax_json_team.php

Je krijgt een JSON object te zien met alle teamleden.

Op die manier kunnen we het script echter niet gebruiken, het verwacht een parameter *'teamlid'* met de *voornaam* van het teamlid.

☞ we proberen opnieuw: geef in http://localhost/plantenshop/services/ajax_json_team.php?teamlid=roger

```
{
  naam: "Roger Mary",
  leeftijd: 59,
  functie: "bedrijfsleider",
  foto: "roger.jpg"
}
```

Bedenk dat het de browser is die deze gegevens nu zichtbaar maakt. JSON data is eenvoudigweg een string variabele die teruggestuurd wordt en het is de JQ method `$.getJSON`die ze zal opvangen voor ons.

☞ in *about.js*, in de `document.ready` handler onderaan, plaatsen we een *event handler* die een wijziging in de keuzelijst opvangt:

```
…
//************** AJAX call nr JSON gegevens team ***********************//


  $('#teamkeuzelijst')
          .change(function(){
                  var waarde = $(this).val();
                  console.log(waarde + ' gekozen');
          })
}) //einde document.ready
```

Bespreking:

- dit script moet **na** het script komen dat de keuzelijst `#teamkeuzelijst` dynamisch aangemaakt heeft
- het `change` event doet een anonieme functie starten
- de waarde van je keuze krijgen met `$(this).val()`
- het `console.log` statement is enkel om te debuggen

Probeer dit even uit: je moet de tekst in de console te zien krijgen.

We vullen verder aan:

```
…
$('#teamkeuzelijst')
        .change(function(){
                var waarde = $(this).val();
                //console.log(waarde + ' gekozen');
                $.getJSON(url, data, success)
        })
…
```

`$.getJSON` is de JSON versie van de uitgebreide functie `$.ajax`. Hierboven hebben we nog geen echte argumenten ingevuld, we hebben enkel de syntax geschreven:

- `url` is een string naar het URL dat de call verwerkt
- `data` (optioneel) is een ***querystring*** die samen met het *request* verstuurd wordt. Een querystring is een lijst *naam=waarde* paren, aan elkaar bevestigd met &-tekens.

  `data` kan ook onder de vorm van een ***map*** worden doorgegeven. Een *map* is een **object** met eigenschappen en waarden. Een map wordt intern omgezet naar een querystring. Een map is handiger als je zelf vaste waarden wil doorgeven.
- `success` is een *callback* functie die de teruggestuurde JSONgegevens verwerkt

We vervangen nu de syntax door echte gegevens:

```
…
$('#teamkeuzelijst')
        .change(function(){
                var waarde = $(this).val();
                //console.log(waarde + ' gekozen');
                $.getJSON(
                        'services/ajax_json_team.php',
                        {teamlid:waarde},
                        function(jeeson){
                          var strHTML = "";
                          if(jeeson.naam){
                           strHTML += "<img src='images/" + jeeson.foto  + "' />";
                           strHTML += "<h3>" + jeeson.naam  + "</h3>";
                           strHTML += "<p>leeftijd: " + jeeson.leeftijd  + "</p>";
                           strHTML += "<p>functie: " + jeeson.functie  + "</p>";
                          }
                          $('#teamgegevens').html(strHTML);
                        }
```

```
                )//einde getJSON
        })
…
```

Bespreking:

- het `url` wordt *services/ajax_json_team.php*. Let op de komma!
- de `data` zijn een *map* (=object) met de parameter *'teamlid'* die de waarde van het geselecteerde `option` uit de lijst meekrijgt, dus bv. "jan"
- de `success`*callback* functie neemt als parameter de teruggestuurde gegevens, die ik hier *jeeson* noem, maar die je eender welke *identifier*naam kan geven (in de boekjes dikwijls *data*, maar dat hebben we al, daarom)
- deze functie bouwt een string op aan de hand van de properties van het JSON object (we veronderstellen slechts 1 teamlid) en
- voegt die met de method `html` in in #teamgegegevens

Nu krijg je de gegevens van een teamlid te zien als je die selecteert:

In je webdeveloper tool kan je de Ajax call observeren in de *Console* flap: na het aanklikken van een teamlid bemerk je het `GET` request. Je kunt er de *parameters*, *headers* en de *response* van observeren. Elke keer je een ander teamlid kiest gebeurt een nieuwe call zonder dat de volledige pagina moet vernieuwd worden.

4.3 Ajax connectie met dataTables

We willen ook de dataTables via ajax doen werken. We bezinnen ons even over wat we gaan doen:

Klassiek serverside	Ajax
▪ **PHP script maakt alle HTML: menu, kop, tabel met gegevens… alles** ▪ **gebruiker kiest andere optie →** ▪ `form` **doet HTTP request voor**	▪ PHP script maakt volledige pagina bij eerste bezoek ▪ gebruiker kiest andere optie → ▪ *jQuery DataTables* doet XHR request voor nieuwe data

volledig nieuwe pagina ▪ **jQuery DataTables maakt tabel op**	▪ *jQuery DataTables* vult geraamte HTML `table` op en maakt die op

Technisch gezien betekent dit ook aan beide zijden – serverside, clientside - een heel andere werkwijze. In het geval van onze tabel is dat

	Klassiek serverside	Ajax
serverside (PHP controller)	▪ controller levert gegevens verpakt in HTML table en de rest van de pagina ▪ gegevens komen via normale businesslaag	▪ controller levert enkel pagina structuur ▪ Presentatielaag moet geraamte van `table` bevatten ▪ **Speciale PHP Ajax service** nodig die gegevens levert in JSON of XML vorm. Dit gebeurt onafhankelijk van de PHP controller en enkel op verzoek van het clientside script
clientside (dataTables)	▪ biedt enkel functionaliteit als gegevens reeds aanwezig zijn (opmaken, sorteren, filteren)	▪ doet XHR verzoek bij elke wijziging zoekfunctie ▪ verwerkt die gegevens in de pagina (hier table) ▪ opmaken, sorteren, filteren

Om het werk dat we tot nu toe gedaan hebben niet te verliezen, maken we een andere pagina aan:

- ☞ open *inhoud.php* en zoek de functie *getMenu()*
- ☞ voeg een extra menuItem "AjaxShop" toe:

```
function getMenu(){
  //menu
  $str = "<ul id='menu'>
                        <li><a href='index.php'>Home</a></li>
                        <li><a href='index.php?page=shop'>Plantenshop</a></li>
                        <li><a href='index.php?page=ajaxshop'>AjaxShop</a></li>
                        <li><a href='index.php?page=zorg'>Verzorging</a></li>
                        <li><a href='index.php?page=galerij'>Galerij</a></li>
                        <li><a href='index.php?page=about'>Over ons</a></li>
```

```
                </ul>";
  return $str;
}
```

- zoek ook de functie *getPlanten()*, kopieer die en hernoem naar *getAjaxPlanten()*
- verwijder alle argumenten uit de functieaanroep, dus

```
…
//***********************************************
function getAjaxPlanten(~~$soort_id, $kleur, $hoogte_min, $hoogte_max~~){
  //Plantenlijst via Ajax
…
```

- wijzig de variabele `$tbl_Planten` naar:

```
$tbl_Planten = "
<table  id='plantenlijst'class='omlijnd'>
  <thead>
        <tr>
                <th>Soort</th>
                <th>kleur</th>
                <th>hoogte</th>
                <th>beginbloei</th>
                <th>eindbloei</th>
                <th>prijs</th>
                <th>rubriek</th>
        </tr>
  </thead>
  <tbody></tbody>
  </table>
  ";
```

waarmee we de oproep naar de plantenservice vervangen door statische HTML.

Werken we met een serverside ajaxscript dan moet een basisstructuur van het table element verplicht aanwezig zijn:

- Een `thead` element met alle kolomtitels (je kan ze via *dataTables* nog wijzigen)
- Een leeg `tbody` element

- open *index.php* en zoek de `case "shop"`
- kopieer die volledig en wijzig naar `case "ajaxshop"`
- vervang daarin de oproep naar *getPlanten()* naar *getAjaxPlanten()*:

```
…
//content
$tpl['rechts']          = getAjaxPlanten();
…
```

☞ vervang ook de oproep naar *shop.js* naar *ajaxshop.js*:

```
…
$tpl['paginaScripts']   = getScriptElements(array("js/vendor/jquery/Datatables-
  1.9.4/media/js/jquery.dataTables.min.js","js/ajaxshop.js"));
…
```

☞ kopieer het javascript *shop.js* en hernoem de kopie *ajaxshop.js*

4.3.1 Serverside ajax script

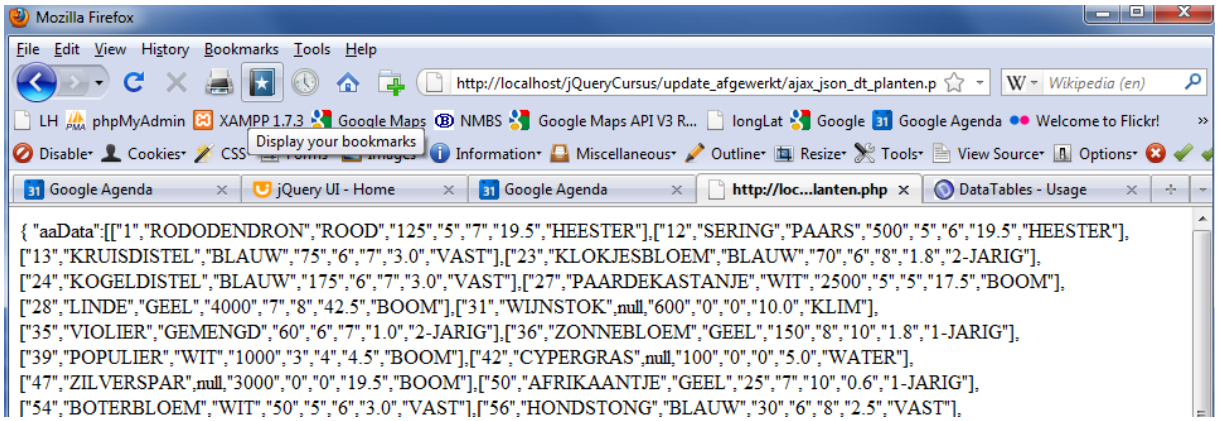
dataTables is in staat een JSON bron te gebruiken om een tabel te vullen. De vorm van deze JSON bron is specifiek voor dataTables: je kan niet eender welk JSON object gebruiken.

De gevraagde vorm bestaat uit een **object** met één property `aaData` die zelf een array als waarde heeft. Dit array bevat dan subarrays met je gegevens:

```
{"aaData":[
        ["1","RODODENDRON","ROOD","125","5","7","19.5","HEESTER"],
        ["12","SERING","PAARS","500","5","6","19.5","HEESTER"],
        ["13","KRUISDISTEL","BLAUW","75","6","7","3.0","VAST"],
        …
  ]
}
```

In onze plantenshop website levert het script *ajax_json_dt_planten.php* de plantengegevens in deze vorm. We zullen het even testen.

☞ typ in de adresbalk van je browser het url van het script in en druk Enter, je krijgt de volledige plantenset in JSON vorm:



Hierboven zie je de output in Firefox. Andere browsers kunnen deze output ook wat gaan opmaken.
Het script reageert ook op een aantal querystring parameters. Bijvoorbeeld:

```
ajax_json_dt_planten.php?soort_id=1
```

geeft enkel de planten van de soort "Heester".
Deze queryparameters zijn nodig om het script te laten reageren op de keuzemogelijkheden van het formulier.

4.3.2 dataTables maakt Ajax connectie

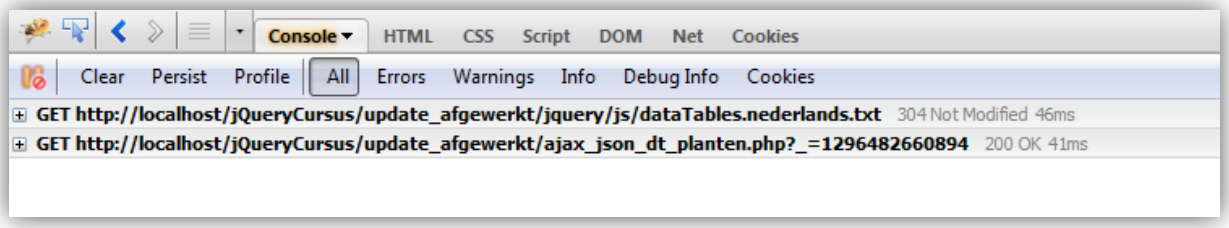
Als je de "Ajaxshop" pagina nu bekijkt zie je enkel de kolomtitels.
We moeten plugin nog aangeven dat hijzelf de data moet ophalen.

- ☞ open het bestand *ajaxshop.js*. het is momenteel slechts een copy van *shop.js*
- ☞ in de `dataTable` method van vorig hoofdstuk voegen we de `sAjaxsource` property toe:

```
//datatables
 $("#plantenlijst").dataTable({
                "sAjaxSource": "services/ajax_json_dt_planten.php",
                "bPaginate":true,
                "bSort":true,
                "iDisplayLength": 20,
                /*"iDisplayStart": 20,*/
                "sPaginationType": "full_numbers",
                "aLengthMenu": [[10, 25, 50, -1], [10, 25, 50, "Alle records"]],
                "bProcessing": true,
                "aaSorting": [[6,'asc'], [2,'desc']],

                "aoColumnDefs": [
                        { "bVisible": false, "aTargets": [ 5 ] },
                        { "bSortable": false, "aTargets": [ 2, 6 ] },
                        { "asSorting": [ "desc" ], "aTargets": [ 3 ] },
                        { "bSearchable": false, "sTitle": "Rubriek", "aTargets": [ 6 ]
 },
                        { "sTitle": "Lengte", "sWidth": "5%", "aTargets": [ 2 ] },
                        { "sClass": "dt_fluo", "aTargets": [ 0 ] }

 ],
                "oLanguage": {"sUrl": "js/vendor/jquery/Datatables-
1.9.4/media/js/datatables.nederlands.txt"}
        });
```

Zo eenvoudig is het: de gegevens worden opnieuw ingeladen in de table maar deze keer via een Ajax call uit de clientside. Veel verschil merk je niet op, toch zie je de

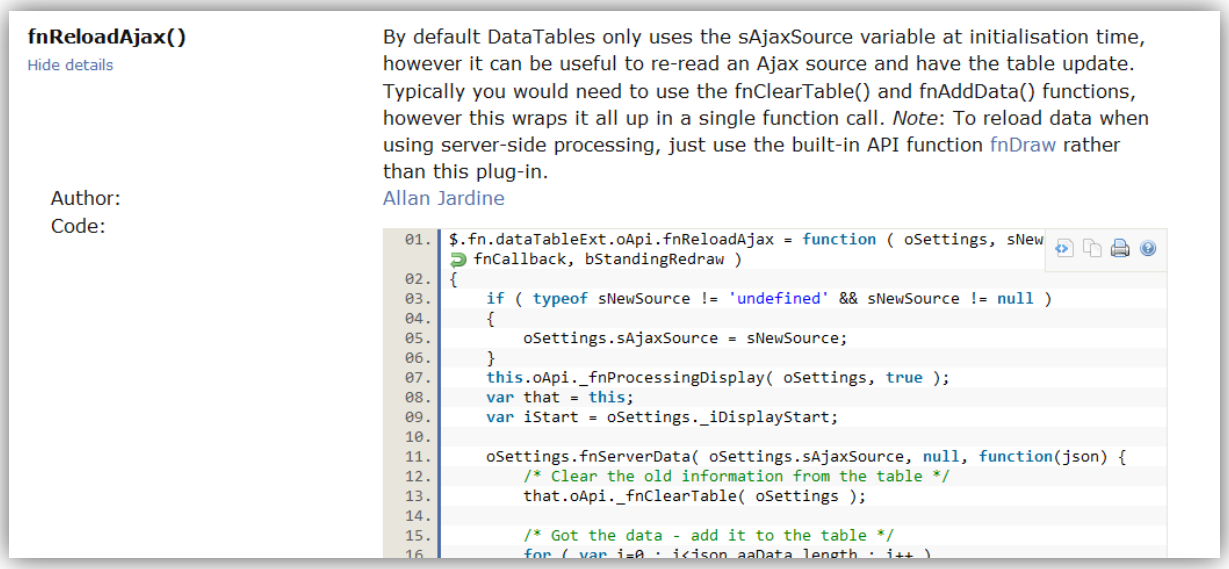
Ajax call passeren in het console venster van Firebug:



4.3.3 Dynamisch bijwerken

In een volgende stap willen we dat de gegevenstabel aanpast als de gebruiker een wijziging aanbrengt in de zoek controls *soort*, *kleur* en *hoogte*. Dat gebeurt momenteel niet.

Standaard laadt `sAjaxsource` de gegevens op **bij het initialiseren**.
Om **nadien** een "opfrissing" te doen hebben we nood aan de plug-in functie `fnReloadAjax()`.
Die functie vinden we op de http://www.datatables.net/plug-ins/api pagina.



- ☞ kopieer de code in een nieuw javascript bestand en noem het *dataTables.fnReloadAjax.js*.
- ☞ plaats het bestand in de *js/vendor/jquery/datatables-1.9.4/media/js* map.
- ☞ in *index.php*, in de `case` "ajaxshop", pas nu ook de variabele `$tpl['paginaScripts']` aan:

```
…
$tpl['paginaScripts']   = getScriptElements(array(
        "js/vendor/jquery/Datatables-1.9.4/media/js/jquery.dataTables.min.js",
        "js/vendor/jquery/Datatables-1.9.4/media/js/dataTables.fnReloadAjax.js",
        "js/ajaxshop.js"));
…
```

Het formulier bevat drie controls voor het opzoeken van planten: *soorten*, *kleur* en *min* en *maxhoogte*. Bij een wijziging van hun waarden moet de tabel aangepast worden. Dat betekent dat we voor elk een eventhandler moet schrijven. Dat mag natuurlijk dezelfde eventhandler zijn.
Om de tabel te kunnen bereiken hebben we een variabele nodig;

- in *ajaxshop.js*, pas de method `$("#plantenlijst").dataTable({})` aan:

```
var oTable = $("#plantenlijst").dataTable({
…
})
```

Dit maakt geen enkel verschil uit voor de functionaliteit, maar onze plantenlijst zit nu wel vervat in de var *oTable*.

- we schrijven nu de functie `herlaadTabel`:

```
function herlaadTabel(){
  //ajaxcall vr nieuwe gegevens vanuit sAjaxSource
  oTable.fnReloadAjax();
  }
```

Nu hebben we een eventhandler nodig voor de drie controls.

- Soorten en kleur kunnen we in één keer doen:

```
/******************event handlers*********************************************/
$("#kleur, #soort_id").change(function(){
  herlaadTabel();
  })

function herlaadTabel(){
  //ajaxcall vr nieuwe gegevens vanuit sAjaxSource
  oTable.fnReloadAjax();
  }

/******************dataTable*************************************************/

 var oTable = $("#plantenlijst").dataTable({
…
```

- Voor de hoogte passen we de `slider()` method aan:

```
//min max hoogte slider
  $("#slider-range-hoogte").slider({
          range   : true,
          values  : [0, 5000],
          min     : 0,
```

```
        max     : 5000,
        step    : 10,
        slide   : function(event, ui) {
                        $("#hoogte_min").val($(this).slider("values", 0));
                        $("#hoogte_max").val($(this).slider("values", 1));
                        herlaadTabel();
                },
        stop: function(event, ui) {
                        $("#hoogte_min").val($(this).slider("values", 0));
                        $("#hoogte_max").val($(this).slider("values", 1));
                        herlaadTabel();
                }
});
```

Als je nu één van de controls wijzigt, bemerk je in de javascript console dat er telkens een Ajax call gebeurt: je ziet de `GET` method uitgevoerd worden.
Maar… de gegevens van de tabel wijzigen niet, we krijgen telkens de volledige tabel terug!?
Inderdaad, de aanpassingen die zover maakten zijn enkel relevant als de brongegevens wijzigen, zoals bv een RSS feed. Dat is hier niet het geval: de database wijzigt zelden.
Wat wij willen is andere parameters meegeven aan de query!

Om zelf controle te krijgen over de ajaxcall die toe nu toe automatisch gebeurd is, moeten we gebruik maken van de property `fnServerData`.
We maken een eerste aanpassing die de kleur zal doen werken.

☞ voeg volgende code toe aan de `dataTable()` method:

```
…
"sAjaxSource": "ajax_json_dt_planten.php",
"fnServerData": function (sSource, aoData, fnCallback ) {
        $.getJSON(     sSource, [ {
                                "name":"kleur",
                                "value":$("#kleur option:selected").val()
                                } ],
                        function (json){fnCallback(json) });
        },
  "iDisplayLength": 20,
…
```

Bespreking:

- `fnServerData` is een functie met de argumenten `sSource`, `aoData` en `fnCallback`. Deze drie argumenten worden doorgegeven aan een `$.getJSON` method die de oproep doet.
- Enkel `aoData` is voor ons belangrijk: dit argument is oorspronkelijk een leeg array waarin je de parameters voor de query als objecten kan stoppen. De documentatie adviseert `aoData.push` te gebruiken, maar dat geeft problemen (een bug?).

Wij vervangen het gewoon door een eigen array:

```
[ {"name":"kleur","value":$("#kleur option:selected").val()}]
```

Bemerk dat we verplicht zijn de properties `name` en `value` te gebruiken, we kunnen niet zeggen `"kleur":$("#kleur option:selected").val()`

Als je nu een bepaalde kleur kiest, dan krijg je de planten van die kleur.
Dus begrijp je dat we nu eenvoudigweg hetzelfde zouden kunnen doen voor *soorten* en ook *min hoogte* en *max hoogte*. Maar er bestaat een nóg betere oplossing …

- ☞ verander de `$.getJSON` method als volgt:

```
…
$.getJSON(      sSource,
                $('form').serializeArray(),
                function (json) {fnCallback(json)}
);
…
```

- ☞ om dit beter te begrijpen laten we via de functie *herlaadTabel()* wat debug informatie zien:

```
function herlaadTabel(){
  //ajaxcall vr nieuwe gegevens vanuit sAjaxSource
  var qs         = $('form').serialize()
  var qsa        = $('form').serializeArray()
  console.log(qs);
  console.log(qsa);
  oTable.fnReloadAjax();
  }
```

Probeer je formulier nu opnieuw met verschillende keuzes.

De debug info toont hoe `serialize` en `serializeArray` (jQ methods) werken: respectievelijk returnen ze een samengestelde string met alle formulierwaarden zoals:

`soort_id=8&kleur=rood&hoogte_min=0&hoogte_max=5000`

of een array van objecten met de properties `name` en `value`zoals:

```
[{"name":"soort_id","value":"8"},{"name":"kleur","value":"rood"},{"n
ame":"hoogte_min","value":"0"},{"name":"hoogte_max","value":"5000"}]
```

En dit laatste is precies wat we nodig hebben!

5 jQuery uitbreiden

Tot hier toe heb je de jQuery *core* library gebruikt, de *UI* widgets en enkele *plugins*. Hoe meer je jQuery gebruikt, hoe meer je de nood zal voelen om zelf eens een applicatie (*plugin*) of een custom control (*widget*) te maken…

In het volgende deel doen we exact dat: een **plugin** en een **widget** maken zodat we straks perfect weten *hoe* dat moet!

Eigen methods - die we integreren met de JQ library - kunnen we net als de JQ functies zelf in 2 categorieën onderbrengen:

- **utility functies**(*global functions*) die algemene functionaliteit bieden (zoals `$.each`)
- **methods** die werken op een *wrapped set* (zoals `.insertAfter`)

5.1 Naamgeving

Een probleem die we kunnen krijgen is "*name collision*": de naam van onze functie/method mag niet in conflict komen met een functie/method van JQ zelf. Niet enkel de functienamen maar ook de *bestandsnamen* waarin we dat opslaan kunnen een botsing geven!
En je plugin kan gebruikt worden in een situatie waarin ook andere libraries geladen worden, met dezelfde namen …

De regels die door het jQuery team opgelegd worden voor het benoemen van een bestand(en) zijn als volgt:

- prefix de naam met "*jquery*"
- vervolg dat met een *punt* en de *naam* van je plug-in
- eindig met "*.js"*

Plan je dus een plugin genaamd "*HTMLManager*", noem dan je bestand *jquery.htmlmanager.js.*

Je kan ook je eigen naam of die van je bedrijf verwerken:
j*query.vdab.htmlmanager.js* of *jquery.jean.htmlmanager.js*

De prefix "*jquery*"vermijdt enige problem met andere libraries. Controleer ook eens of iemand anders geen "*HTMLManager*" plugin gemaakt heeft, als je van plan bent hem te publiceren.

Eenmaal je plugin klaar, dan kan je hem registreren op de jQuery site
http://plugins.jquery.com/docs/publish/

5.2 Aan wie behoort de $?

In een jQuery script staat de `$` voor het `jQuery` object, het is dus een alias. Die verkorte schrijfwijze is erg handig maar niet uniek: er zijn heel wat libraries die `$` gebruiken, nota bene *Prototype, Mootools* en *YUI*.

In moderne applicaties werken dikwijls meerdere libraries samen, dus reserveren we het dollarteken binnenin een jQuery script voor `jQuery` door het door te geven als argument in de functie:

```
(function($) {

/* code gebruikt $ voor jQuery */
$("div").hide();

})(jQuery)
```

Deze manier van werken wordt niet alleen in jQuery plugins gebruikt maar ook in andere applicaties die jQuery gebruiken.

Een andere mogelijkheid is het gebruik van het '$' teken over te laten aan andere libraries. Dat doen we door`jQuery.noConflict()` als eerste jQ code te plaatsen vóór alle ander jQuery code en vanaf dan enkel `jQuery`, geen `$`, te gebruiken in ons script.

```
<script src="prototype.js"></script>
<script src="jquery.js"></script>
<script>
jQuery.noConflict();

     // Gebruik jQuery via jQuery(...)
jQuery(document).ready(function(){
       jQuery("div").hide();
});

     // Gebruik Prototype with $(...), etc.
$('someid').hide();
</script>
```

5.3 Een eigen utility functie

Utility functies werken op het jQuery object zelf en beginnen daarom steeds met een `$`, bijvoorbeeld`$.each()` of `$.noConflict().`

Het zijn functies die een algemene functionaliteit hebben, ze zijn NIET toepasbaar op een DOM element of een *wrapped set*. Misschien wel op een algemeen Javascript object zoals een array of een object of ze voeren een bepaalde procedure uit.

Zo kunnen we een algemene functie *zegDankUTegen (wie)* aanmaken.

Eerst hebben we een eigen plugin bestand nodig:

- ☞ maak een nieuw bestand *jquery.jouwvoornaam.utils.js* en zet die in de map *js/vendor/jquery/js*
- ☞ typ daarin de code voor de nieuwe functie:

```
// jquery.maurice.utils.js
```

```
// jQuery extensions door Maurice
(function($) {
        $.zegDankUTegen= function (wie){
                alert("DankUWel " + wie + " !");
        }
})(jQuery)
```

☞ koppel nu deze plugin door het bestand *pres/head.tpl* aan te passen:

```
…
<!-- algemene JS scripts voor alle pagina's via vaste SCRIPT elementen-->
<script src="js/vendor/modernizr-2.6.2.min.js"></script>
<script src="js/vendor/jquery/js/jquery-1.10.1.min.js"></script>
<script src="js/vendor/jquery/js/jquery-ui-1.10.3.custom.min.js"></script>
<script src="js/vendor/jquery/js/jquery.maurice.utils.js"></script>

<!--localisation files datepicker-->
…
```

☞ en tenslotte kunnen we het even uittesten op het einde van het script in *about.js*:

```
  …
$.zegDankUTegen('Gilbert');

})//einde document ready
```

☞ ververs even de *'Over Ons'* pagina om uit te testen.

Nog een voorbeeld:

```
// jQuery extensions door Maurice
(function($) {
$.zegDankUTegen = function (wie){
  alert("DankUwWel" + wie + "!");
  }
$.vandaag = function (){
  var vandaag = new Date();
  return vandaag.toLocaleDateString();
  }
})(jQuery)
```

☞ dat passen we dan ook toe in *about.js*:

```
  …
  $.zegDankUTegen('Gilbert');
```

```
  $('<li>').html($.vandaag()).prependTo('footer ul');
})//einde document ready
```

en zo verschijnt de huidige datum in de footer van alle pagina's.

5.4 Een eigen wrapper method

De kans is groot dat je een eigen method wil ontwikkelen voor een *wrapped set* die kan gebruikt worden in het *chaining* process.

- ☞ dergelijke methods voegen we toe aan het `jQuery.fn` object:

```
$.fn.wordtGroen = function(){
                return this.css('color','green');
        }
```

`jQuery.fn` is een alias voor `jQuery.prototype`.

Elke nieuwe method wordt meegegeven als een property van het type `function` in de `extend` method. Op deze manier kunnen we er meerdere tegelijk toevoegen.

- ☞ dat we kunnen toepassen als volgt in *about.js*:

```
  $('<li>').html($.vandaag()).prependTo('footer ul').wordtGroen();
```

In bovenstaand voorbeeld refereert `this` naar de *wrapped set* waarop we dus alle mogelijke JQ functies kunnen toepassen.

Belangrijk!

Een wrapper method moet altijd de oorspronkelijke set teruggeven, vandaar de `return this`. Op die manier kan je method in een chain gebruikt worden

5.5 Een select vullen als wrapper method

Een ietwat nuttiger method zou het vullen van een `select` element kunnen zijn: herinner je hoeveel maal we dat gedaan hebben in de Javascript cursus!

We zijn optimistisch en verkopen het vel van de beer voor hij geschoten is:

- ☞ pas de code in *about.js*, waar we de `select` 'teamkeuzelijst' opvullen, aan als volgt:

```
// teamlijst: versie vr JSONgegevens
  var $container        = $('<div id="teamboks">');
  var $diefrechts       = $('<div id="teamgegevens">');
  var $keuzelijst       = $('<select id="teamkeuzelijst">');
  var strDeOptions      = '<option value="">--- het team ---</option>';
  /*
  oorspronkelijke versie
```

```
$.each(lijst, function(n, value){
                strDeOptions  += '<option>' + value + '</option>';
})
$keuzelijst.html(strDeOptions);
*/

//met custom wrapper method
$keuzelijst.vulSelect(lijst, "-- kies een teamlid --");

$container.append($keuzelijst).prepend($diefrechts); //de float eerst in de flow
$('#team').after($container);
```

Bespreking:

- de oorspronkelijke lus `$.each` waarmee de lijst opgevuld werd, wordt gecommentarieerd
- we laten de lijst nu opvullen met onze eigen method:
  `keuzelijst.vulSelect(lijst, "-- kies een teamlid --");`
- merk op dat de keuzelijst nog steeds
  `$('<select id="teamkeuzelijst">')` is,
- en dat de default `option` vervangen werd door een string
  `"-- kies een teamlid --"`

☞ nu maken we deze method in onze eigen library *jquery.jouwvoornaam.utils.js:*

```
…
 $.fn.vulSelect = function(arrData, strFirstOption) {
  /*
   vult een SELECT met gegevens uit een array, een optioneel eerste item is mogelijk
  @arrData              1-dim array TEKST of 2-dim array VALUE|TEKST
  @strFirstOption       string, optioneel, de tekst voor een eerste, default option,
         de value is steeds ''
  */
  return this.each(function(){
   if (this.tagName=='SELECT') {
         var eSelect = $(this);
         if(strFirstOption != null) {
                eSelect.append("<option value='' selected='selected'>" ⏎
                                + strFirstOption + "</option>");
         }
         //is het array 1 of 2-dimensioneel?
         if(!$.isArray(arrData[0])) {
                $.each(arrData,function(index, data){
                       eSelect.append('<option value=' + arrData[index] + '>' ⏎
                                + arrData[index] + '</option>');
                });
```

```
        }
        else{
                $.each(arrData,function(index, data){
                        eSelect.append('<option value=' + arrData[index][0] + '>' +
                                + arrData[index][1] + '</option>');
                });
        }
  }  //einde if
})  //einde this.each
} //einde vulSelect
…
```

Bespreking:

- de wrapper method `vulSelect()` heeft twee argumenten:
  - *arrData*:
    - ofwel een 1-dimensioneel array zoals `['roger', 'evelyn','hilde','jan']`
      waarbij de `option` elementen van de keuzelijst voor `value` en tekst hetzelfde krijgen,
      dus `<option value='roger'>roger</option>`
    - ofwel een 2-dimensioneel (genest) array zoals `[[123,'roger'],[456,'evelyn'],[789,'hilde'],[012,'jan']]`
      waarbij de `option` elementen van de keuzelijst voor `value` en tekst respectievelijk het `[0]` en `[1]` item van het genest array krijgen,
      dus `<option value='123'>roger</option>`
  - *strFirstOption*:
    de tekst voor een eerste default `option`.
    De value van deze `option` is steeds '' (lege string).
- waarom begint de functie met `return this.each(function(){ …` ?
  omdat we uiteraard dezelfde lijst moeten returnen, en waarbij de `each` werkt op **alle** `select` **elementen** in de *wrapped set*!
  Denk er aan dat sets meerdere elementen kunnen bevatten.
  Zo kan je, indien nodig, meer dan één lijst terzelfdertijd opvullen
- we testen eerst of we wel met een `select` element te maken hebben met `if (this.tagName=='SELECT') {`
- we maken van `this` (= de `select`) een *wrapped set*`$(this)`, niet enkel om er JQ methods op toe te kunnen passen, maar ook om eventueel `this` te kunnen gebruiken in de `each` lussen die volgen
- we testen de aanwezigheid van *strFirstOption* en voegen eventueel een eerste `option` element toe
- we testen of het *arrData* een 1- of een 2 dimensioneel array is door het eerste item van het array (`arrData[0]`) te testen met de JQ method `$.isArray`.
- Afhankelijk van dit resultaat gaan we doorheen het array en bouwen de verschillende `option` elementen op.

- daarna wordt de opgebouwde `select` gereturned

Test nu even de werking van de teamlijst.

Het script in *about.js* is erg eenvoudig, stel je echter voor dat een *event* de lijst moet opvullen (denk aan gekoppelde keuzelijsten), dan zal het zeker voorvallen dat een gebruiker tweemaal na elkaar het *event* triggert, met als gevolg dat dezelfde items meermaals aan de `select` worden toegevoegd. Daarom is het veiliger de `select` eerst te legen alvorens hem opnieuw op te vullen.

Een *wrapper method* `leegSelect` gaat als volgt:

```
$.fn.leegSelect = function() {
  return this.each(function(){
   if (this.tagName=='SELECT') {
         $(this).empty();
  }
   });
 }
```

Bespreking:

- opnieuw returnen we alle items in de *wrapped set* met `this.each()`
- de method `empty()` verwijdert dan alle childNodes uit de `select`

Omdat infeite elke keer we een keuzelijst opvullen, de inhoud eerst geleegd kan worden, passen we ook onze `vulSelect` aan:

```
…
eSelect = $(this);
eSelect.leegSelect();
  if(strFirstOption != null) {
  …
```

5.6 Een widget maken met de Widget Factory

Als je zelf een widget wil maken vertrek je vanuit de UI *Widget Factory*, een constructor *Class* waarvan je een eigen widget afleidt.

Alle bestaande widgets (*tabs, accordion, dialog*, etc…) zijn afgeleid van de factory en bezitten dus diens basisfunctionaliteit.

De *Widget Factory* zelf is het bestand *jquery.ui.widget.js (*in de *development bundle)*, maar hij zit ook ingesloten in het jquery UI core library, je hebt het dus niet echt nodig en je hoeft er niet speciaal naar te koppelen.

De *Widget Factory* bevat een aantal private en publieke methods die je naar believen kan overschrijven en aanvullen. Daarnaast kan je je eigen methods schrijven. Je moet enkel volgende regeltjes volgen:

- een **private method** kan enkel van binnenin de widget gebruikt worden, door een andere method. Dit wordt gecontroleerd door de widget zelf.
  Zo'n private method begint steeds met een ***underscore***, zoals `_create`
- een **publieke method** kan gebruikt worden door een opdrachtgever om de widget te besturen. Een publieke functie begint nooit met een underscore.

Er zijn een aantal **publieke** properties, waarvan hier de belangrijkste:

method	Doel
`document`	het document waar het widget's element in zit
`options`	Een object met de instellingen voor de widget als properties. In de widget stel je hiermee de standaard opties in en die kunnen overschreven worden door de gebruiker. Bevat standaard een property `disabled`
`element`	jQ set. Het element waarop de widget opgeroepen is. Het DOM element in deze set is te bereiken met `this.element[0]`.
`namespace`	de plaats waar de widget opgeslagen is, bijvoorbeeld: als de namespace "`ui`" is dan wordt de widget opgeslagen in `$.ui`
`widgetName`	de naam van de widget

Enkele methods als voorbeeld:

method	priv/ pub	doel
`destroy()`	pub	verwijdert de instantie op het DOM element. Verder aan te vullen door de ontwikkelaar
`_destroy()`	priv	interne verdere afwerking van het verwijderen van de instantie
`option(key, value)`	pub	instellen van een optie **na** initialisatie
`enable()`	pub	zet de `disabled` optie op `false`. Verder in te vullen door de ontwikkelaar
`disable()`	pub	zet de `disabled` optie op `true`. Verder in te vullen door de ontwikkelaar
`_create()`	priv	code voor de aanmaak van de widget. Eenmalig.
`_doeIets()`	priv	eigen interne method
`doeIets`	pub	eigen publieke method

5.6.1 Een widget voor de fotogalerij

☞ open het bestand *content/inhoud.php* en zoek de functie *getGalerij()*.
Bekijk de HTML van de thumbnails.

De thumbnails op de pagina "Galerij" bestaan allen uit `figure` elementen met een `img` en een `figcaption` als children, HTML5 dus.

Het zou leuker zijn als de *captions* eerst verborgen waren en dat ze dynamisch tevoorschijn komen bij een `mouseover`. Omdat dit ons wel een leuk gadget lijkt die we ergens anders ook wel kunnen gebruiken – en misschien met anderen delen – maken we er een *widget* van.

We noemen onze widget "*knipoog*".

Koppelingen naar de *Widget Factory*zijn onnodig want, zoals gezegd, zit die vervat in de *UI core*.

Ons eigen widget bestand en een paginascript moeten we wel koppelen:

☞ open *index.php* bij de case "galerij" en voeg toe:

```
…
case "galerij":
  /*** Fotogalerij ***/
  $tpl['title']         = "de Plantenshop: fotogalerij";
  $tpl['body_id']       = "galerij";
  //content
  $tpl['rechts']        = getGalerij();
  $tpl['paginaScripts'] = ↵
                getScriptElements(array("js/vendor/jquery/lightbox/js/lightbox.js"↵
                ,"js/vendor/jquery/js/jquery.ui.knipoog.js","js/galerij.js"));
  break;
```

☞ maak een nieuw javascript bestand aan en noem het *jquery.ui.knipoog.js* en plaats het in de map *jquery/js*.Dit is het widget bestand zelf.

```
// JavaScript Document
// KNIPOOG, widget om het figcaption element van figure element dynamisch te tonen
// maakt gebruik van de widget factory

(function($) {



})(jQuery);
```

Bespreking:

- dit is een anonieme functie die zichzelf opstart: de twee haakjes achteraan (met het argument `jQuery`) zorgen daarvoor.
  Alle code erbinnen wordt onmiddellijk uitgevoerd.
- de variabelewaarde `jQuery` in die haakjes wordt doorgegeven aan het argument `$`-teken. Zoals eerder uitgelegd schermt dat het gebruik van de `$` af binnen de functie

☞ Nu definiëren we onze plugin met behulp van de *Widget Factory*:

```
// JavaScript Document
// KNIPOOG, widget om het figcaption element van figure element dynamisch te tonen
// maakt gebruik van de widget factory

(function($) {
  $.widget("ui.knipoog", { })

})(jQuery);
```

Bespreking:

- de method `widget` bevat twee argumenten:
  - de **naam** van de widget: "*knipoog*" die in de "ui" ***namespace*** valt
  - een *letterlijk* **object** dat alle properties en methods van de widget zal bevatten

We openen nu de accolades van het object en schrijven alle verdere code er in:

☞ een public property van de widget is `options`, een object:

```
(function($) {
  $.widget("ui.knipoog", {
        options: {
        location      :"top",
              color          : "black",
              bgColor        :"silver",
              speed          :"slow",
              padding: 4
              }

}); //einde widget
})(jQuery);
```

Bespreking:

- het `widget` object bevat verplicht een `options` object. Dat bevat een aantal instellingen,als properties, die de widget gebruikt en die we kunnen wijzigen bij het opstarten van de widget

5.6.2 Aanmaken van de instantie

De *private* method `_create` wordt door de widget slechts éénmaal uitgevoerd tijdens de initialisatie. Alles wat bij de start moet gebeuren, moet in deze method zitten.

Deze method móet dus ingevuld worden

☞ voorlopig beperken we ons tot het tonen van wat debug gegevens:

```
…
$.widget("ui.knipoog", {
        options: {
```

```
        location      :"top",
                color          : "black",
                bgColor        :"silver",
                speed          :"slow",
                padding: 4
                },
        _create: function(){
                //initialisatie van de widget
                //this.element bevat het figure element als JQset
                this.element.img      = $('img',this.element);
                this.element.cap      = $('figcaption',this.element);

                var o                 = this.options;
                console.log (this.element[0].nodeName);
                }

}); //einde widget
…
```

Bespreking:

- `this.element` is het **element** waarop de widget opgeroepen wordt, hier zal dat dus de `figure` zijn.
- `this.element.img` is de set `img` elementen binnenin de `figure`. Hier slechts één
- `this.element.cap` is de set `figcaption` elementen binnenin de `figure`. Hier ook slechts één
- de var *o* bevat de options
- het `console` statement toont de `nodeName` van het element waarop de widget werkt: dat *moet* hier "FIGURE" zijn. Merk op dat we de jQ set eerst moeten omzetten naar een DOM node met `[0]`

Om dit te kunnen uittesten moeten we de widget nog opstarten.

☞ open *galerij.js* en voeg toe

```
// JavaScript Document
// voor fotogalerij pagina

$(function(){

  $('figure').knipoog();

  })
```

Bespreking:

- in de `document.ready` handler selecteren we alle `figure` elementen in de pagina en voeren er onze "knipoog" widget op uit door zijn naam te gebruiken als method: `knipoog()`

Dit is voornamelijk bedoeld als controle of onze widget geladen wordt: je moet in de *Javascript Console* 15 keer "FIGURE" zien verschijnen voor elk element waarop de widget aangeroepen wordt. Gebeurt dat niet dan is de kans groot dat er iets mis gaat met het pad naar het widget bestand: controleer de bronnen die geladen worden via je developer tool.

Merk ook op dat we dus ***15 instances van onze widget hebben***, niet slechts 1!

We doen verder.

De method `_create` moet nu het grootste deel van het werk oplossen: de `figure` elementen en hun inhoud – `img` en `figcaption` - klaar maken met CSS en eventhandlers.

☞ zet het `console` statement in commentaar en voeg toe:

```
…
_create: function() {
         //initialisatie van de widget
         //this.element bevat het figure element als set
         this.element.img      = $('img',this.element);
         this.element.cap      = $('figcaption',this.element);
         var o                 = this.options;
         //console.log (this.element[0].nodeName);
         //vaste eigenschappen
         this.element.css({position:'relative', height: '100px'});
         this.element.cap
                .hide()
                .css({
                       position      : 'absolute',
                       left          : 0,
                       width         : this.element.img.width() - (o.padding * 2),
                       height        : '80px',
                       opacity       : '0.7',
                       padding       :o.padding
                       });

  },
…
```

Bespreking:

- met de method `css()` stellen we de position van de `figure` in op *relative*. Dat doen we om straks het `figcaption` *absolute* te kunnen positioneren
- we verkleinen de hoogte om een wat compactere lay-out te krijgen
- eerst verbergen we het `figcaption` element met `hide()`
- daarna stellen we een heleboel CSS eigenschappen ervan in:
  - we positioneren het element *absolute* in de linkerbovenhoek van zijn parent, `figure`. Bemerk dat we de eigenschap `top` achterwege laten (zie verder)
  - we stellen de `width` ervan in met een berekening: `width()` berekent de breedte van de `img` (die niet meegegeven is in de tag!) , min de dubbele `padding`
  - de hoogte wordt voorlopig op 80px gezet
  - de tekstkleur neemt de `option` *color* over
  - de achtergrondkleur neemt de `option` *bgColor* over
  - de `opacity` is 0.7
  - de `padding` wordt de `option` instelling ervan
- bedenk dat de `figcaption` nog steeds onzichtbaar is op dit moment

Je bemerkt waarschijnlijk dat de CSS die hier toegepast wordt niet volledig is: waar zijn `color`, `bgColor`, en `top`?

Om meer modulair te kunnen werken splitsen we de code op in twee delen:

- `_create()` bevat de code die slechts éénmalig uitgevoerd wordt
- terwijl de private functie `_CSStoepassen()` – zie hieronder– code zal bevatten die uitgevoerd wordt bij een update aan de opties

☞ eerst voegen we de oproep naar `_CSStoepassen()` onderaan toe aan `_create()`:

```
…
_create: function() {
                        …
                        padding        : o.padding
                        } );
        //aanpasbare eigenschappen
        this._CSStoepassen();
  },
…
```

☞ nu maken we de private functie `_CSStoepassen()`:

```
…
_CSStoepassen:function(){
        //alle aanpasbare eigenschappen hier
        this.element.cap.css({
                color                :this.options.color,
                backgroundColor      :this.options.bgColor,
                });
```

```
        //location speciaal
        switch(this.options.location){
                case "top":
                                this.element.cap.css({top:0});
                                break;
                case "bottom":
                                this.element.cap.css({bottom:0});
                                break;
                default:
                                this.element.cap.css({top:0});
                                break;
                }
        },
…
```

Bespreking:

- we passen de CSS eigenschappen `color` en `backgroundColor` toe met de waarden uit de options
- om de absolute positionering toe te passen moeten we weten of de option `location` de waarde "top" of "bottom" zal hebben.
  Het is niet mogelijk dynamisch een CSS property toe te kennen:
  `"top":0` of `"bottom":0,` zelf via een variabele.
- daarom gebruiken we een `switch` statement om ofwel `top` ofwel `bottom` toe te kennen aan de CSS van de `figcaption`

5.6.3 "Hover" event handler

Nu willen we dat een `mouseover/mouseout` van de prent de `figcaption` tevoorschijn tovert/doet verdwijnen. Daarvoor hebben we een "hover" event handler op de `figure` nodig. Ook die zetten we in een aparte private functie en roepen die op vanuit `_create`:

```
…
_create: function() {
                        …
                        padding        : o.padding
                        } );
        //aanpasbare eigenschappen
        this._CSStoepassen();
        //hover event handler voor het element
        this._setMouseHandler();
  },
…
```

En nu maken we de private functie `_setMouseHandler()`:

```
…
_setMouseHandler: function(){
  //hover event handler
  var self       = this;
  var o          = self.options;
  self.element.hover(
         function () {
         self.element.cap.show(o.speed);
         },
         function () {
         self.element.cap.hide(o.speed);
         });
  },
…
```

Bespreking:

- de var *self* wordt gelijkgesteld aan `this`:
  - `this` is **momenteel** de instantie van de widget
  - door `self` gelijk te stellen aan `this` wordt `self` ook de instantie van de widget.
    Nu bestaat er ook een `self.element,` een `self.element.img` en een `self.element.cap`
- de jQ method `hover()` bevat twee *event handlers* als anonieme functies: de eerste voor `mouseover`, de tweede voor `mouseout`
  - in een event handler is `this` niet langer de instantie van de widget: `this` wijst dan naar de *event target* van de event handler, dus het *element* waarop het event plaats heeft
  - daarom gebruiken we nu `self` om de widget aan te duiden.
  - in de eerste functie tonen we de `figcaption` met `show()`
    - `show()`- zoek even op in de jQ docs - accepteert een argument die de snelheid van het effect bepaalt: we gebruiken daarvoor de `options` instelling *speed*
  - in de tweede verbergen we hem weer met `hide()`

Probeer dit even uit: je ziet dat ons widget werkt, maar…

5.6.4 Window.load

Maar misschien werkt hij niet naar behoren?

Een probleem dat *kan* opduiken is dat de breedte van de caption niet correct is.

Dat wordt veroorzaakt door onze berekening: om de breedte van de `figcaption` exact te doen overeenkomen met die van de `img`, gebruiken we `this.element.img.width()`. Deze functie leest de reële waarde van de foto,

vermits die niet als attribuut staat in de `img` tag. Dat kan echter enkel als de foto **volledig** ingeladen is in de `img` tag!

We roepen de widget op in de `document.ready` handler: die vuurt af als de DOM tree volledig geladen is, maar dat is niet inclusief *alle* beelden op de website. Sommige beelden zullen al ingeladen zijn, andere komen pas toe na de `document.ready`.
Om die reden stappen we deze keer af van onze vertrouwde `document.ready` handler en vervangen die door `window.load`: dit *event* wacht tot **alles** ingeladen is.

☞ wijzig *galerij.js* naar

```
$(window).load(function(){


  $('figure').knipoog();


})// einde window.load
```

5.6.5 Effects

We willen de eenvoudige `show()` en `hide()` wat opsmukken met één van de UI *effects*: bekijk ze op de jQuery UI pagina.

☞ dus breiden we de `hover()` event handler uit:

```
…
self.element.hover(
        function () {
                self.element.cap.show("slide",{direction:"left"},o.speed,
                        function(){ });

                },
        function () {
                self.element.cap.hide('slide',{direction:"right"},o.speed,
                        function(){ });

                }
        );
…
```

Bespreking:

- de methods `show` en `hide` kunnen ook meer argumenten bevatten:
  - het *'slide'* `effect`
  - eventuele `options` (een object), hier met de prop *direction*
  - een `snelheid` als derde argument
  - en een `callback function` die uitgevoerd wordt als de method eindigt. Dit argument is optioneel en mag weggelaten worden als je het niet gebruikt, maar wij voorzien het hier met oog op wat komt. We laten de functies voorlopig leeg.

5.6.6 Options meegeven

☞ we kunnen de standaardinstellingen van de widget overschrijven door zelf enkele options mee te geven:

```
// JavaScript Document
// voor fotogalerij pagina
$(function(){
  $('figure').knipoog({bgColor:"cyan", color:"navy", location:"bottom"});
  })
```

Bespreking:

- de instellingen worden meegegeven als een object met properties
- de *widget factory* zorgt ervoor dat de standaard instellingen overschreven wordt met de meegegeven instellingen

De kleuren en de positie zijn aangepast.

5.6.7 Functieknoppen

Voor het vervolg van onze uitleg hebben we wat interactie nodig met de widget ná zijn opstart. We voorzien een aantal knopjes waarmee we publieke methods kunnen uitvoeren.

☞ open het bestand *content/inhoud.php* en zoek de functie *getGalerij()*.
☞ vul de `$str` variabele verder in met de volgende HTML op de plek waar je de commentaarlijn `//extra knoppen hier` ziet:

```
…
  <p>Enkele soorten:</p>";
  //extra knoppen hier
  $str .= "<p>
          <span id='stop' class='knop'>stop</span>
          <span id='boven' class='knop'>boven</span>
          <span id='aan' class='knop'>aan</span>
          <span id='af' class='knop'>af</span>
          </p>";
  $str .= "</section>";
…
```

Dit zorgt voor een aantal woorden boven de fotogalerij.
Om deze nu het uitzicht van een knop te geven gebruiken we de **Button** widget.

☞ we kennen die toe in *galerij.js*:

```
$(window).load(function(){
  //widget laden met options
  $('figure').knipoog({bgColor:"cyan", color:"navy",location:"bottom"});
  //knoppen stylen:
  $('.knop').button();
```

```
})// einde window.load
```

Bespreking:

- alle `span` elementen hebben de `class` 'knop' die we gebruiken als selector
- we passen de *Button* widget toe

Nu zien we duidelijke knoppen.

5.6.8 enable/disable

Elke widget heeft een publieke method `enable` en een method `disable`. De factory zelf voorziet deze methods via zijn constructor, maar meer dan een attribuut zetten in het element doen die niet. De rest moeten we zelf invullen.

☞ we voegen deze twee methods toe aan *knipoog*:

```
… ,
  enable: function() {


  },
  disable: function() {


  } ,
…
```

☞ om deze methods te kunnen uitvoeren koppelen we een event handler aan twee van de voorziene knoppen in *galerij.js*:

```
…
  $('.knop').button();

  //event handlers voor knoppen
  $('#af').on("click",function(){
         $('figure').knipoog("disable");
         })
  $('#aan').on("click",function(){
         $('figure').knipoog("enable");
         })
})// einde window.load
```

Bespreking:

- we gebruiken de algemene event handler `on()` om het `click` event te binden aan de method van onze widget:
- om een publieke method van een widget uit te voeren, geef je de naam van de method mee als `String` argument, dus hier

  ```
  $('figure').knipoog("disable");
  ```

- we doen hetzelfde voor de "aan" knop voor `enable`

Alle onderdelen zijn daarmee op hun plaats, maar als je klikt op "af" of "aan", gebeurt er niets. Nochtans hadden we gezegd dat de *widget factory* default functionaliteit heeft voor deze methods?

Ja maar, als je in Javascript een **override** doet, moet je de in de afgeleide instantie (onze 'knipoog' widget) de default method oproepen.

☞ vul daarom aan in *jquery.ui.knipoog.js*:

```
… ,
  enable: function() {
          $.Widget.prototype.enable.apply( this, arguments );
  },
  disable: function() {
          $.Widget.prototype.disable.apply( this, arguments );
  } ,
…
```

Hiermee passen we de method van de constructor toe.

☞ probeer opnieuw: nu merk je dat de beelden mat worden maar dat de *hover* nog steeds werkt, hoe komt dat?

Inderdaad, de `figure` elementen hebben verschillende `ui-disabled` classes gekregen, maar dat stopt onze functionaliteit niet, die moeten we zelf stilleggen.

☞ we vullen opnieuw aan:

```
… ,
  enable: function() {
          $.Widget.prototype.enable.apply( this, arguments );
          this._setMouseHandler();
  },
  disable: function() {
          $.Widget.prototype.disable.apply( this, arguments );
          this._removeMouseHandler();
  } ,
…
```

De private functie `_removeMouseHandler()` moeten we nog maken, dat doen we nu:

```
…
_removeMouseHandler: function(){
          this.element.unbind('mouseenter mouseleave');
  },
…
```

Bespreking:

- omdat de jQ `hover` event handler infeite twee handlers bevat, moeten we er ook twee verwijderen: `unbind` op het element, koppelt `mouseenter` en `mouseleave` los

Nu kan je de knoppen "af" en "aan" uittesten.

5.6.9 Locatie aanpassen

Hoe kunnen we een widget instelling *nadien* laten aanpassen door de gebruiker?

De knop "boven" moet de locatie van de bewegende `caption`– die nu via de `options` op "bottom" staat - opnieuw naar boven verhuizen.

In *galerij.js* voegen we een event handler voor de "boven" knop toe:

```
…
$('#boven').on("click",function(){
        $('figure').knipoog("option", "location","top"); //
  })
…
```

Bespreking:

- Om een publieke method van een widget te gebruiken is de syntax:

```
$('selector').widgetNaam('publiekeMethodNaam', argument1, argument2, …)
```

- dus hier is de method `option` die twee argumenten verwacht: de eigenschapsnaam en zijn waarde

Elke widget heeft – via zijn constructor - een publieke method `option` waarmee een instelling kan aangepast worden. Deze method gebruikt zelf een private method `_setOption` om individuele opties aan te passen.

Wij zullen enkel iets toevoegen aan de functionaliteit van deze *private method* om ervoor te zorgen dat onze aanpassingen onmiddellijk effect hebben.

☞ voeg een method `_setOption` toe aan onze widget in *jquery.ui.knipoog.js*:

```
… ,
_setOption: function(option, value) {
  $.Widget.prototype._setOption.apply( this, arguments );
        this._CSStoepassen();
  },
…
```

Bespreking:

- ook hier voeren we de default actie van `_setOption` eerst uit via het prototype van de widget
- en dan voeren we `_CSStoepassen()` opnieuw uit om de gewijzigde eigenschap toe te passen

De *captions* glijden nu langs de bovenzijde van de figuren.

5.6.10 destroy

Elke widgetinstantie beschikt ook over een **publieke** method `destroy` die de widget instantie wegneemt van het element. Maar ook hier doet het enkel dat, de beginsituatie terugzetten is onze taak.

We maken een method `destroy` aan die een *override* doet van de constructor method:

```
… ,
destroy: function() {
         this._vernietig();
  },
…
```

Maar ook hier delegeren we onmiddellijk naar een private functie. De reden wordt straks duidelijk.

De method `_vernietig()` roept de constructor method aan en zet de beginsituatie terug:

```
…
_vernietig :  function(){
         // call the base destroy function
         $.Widget.prototype.destroy.call( this, arguments );
         this._removeMouseHandler();
         this.element.css({height:'180px'});
         this.element.cap
                  .css({
                           position            :'static',
                           width               :'auto',
                           height              :'auto',
                           color               :'inherit',
                           backgroundColor     :'inherit',
                           opacity             :'1',
                           padding             :0
                         })
                  .show();
         },
…
```

Bespreking:

- eerst roepen we de base method aan: die verwijdert de instantie van de widget van ons element
- daarna verwijderen we de mouse handlers
- en zetten de `height` en `position` van het `figure` element terug neutraal
- daarna zetten we de CSS van het `figcaption` element terug wat het voordien was
- en tonen het `figcaption` element opnieuw

We moeten deze method ook nog koppelen aan de knop "stop" in *galerij.js*:

```
…
$('#stop').bind("click",function(){
        $('figure').knipoog("destroy"); //
  });
…
```

Nu kunnen we uittesten.

Er kan zich een probleem voordoen: als er tijdens het klikken op "stop" nog animaties bezig zijn, blijken niet alle instanties in hun oorspronkelijke vorm terug te keren, met als gevolg dat de layout slecht is.

Zolang een animatie aan de gang is wordt de `destroy` method niet volledig uitgevoerd.

Om dit op te lossen moeten we een en ander aanpassen. Voeg eerst twee interne variabelen toe aan de widget:

```
$.widget( "ui.knipoog", {
  // **CHANGE** set default values
  options: {
        location      : "top",
              bgColor       : "silver",
              color         : "black",
              speed         : "slow",
              padding     : 4
              },
  _active:              false,
  _destroyCalled:       false,
  …
```

Bespreking:

- de var `_active` met de waarde `false` zal de *'state'* van animatie bijhouden: is de animatie nog bezig of is hij gedaan
- de var `_destroyCalled` met de waarde `false` zal de bijhouden of de method destroy geroepen werd en uitgevoerd is

Een tweede aanpassing doen we aan:

```
_setMouseHandler: function(){
  //zet hover event handler
        var self      = this;
        var o         = self.options;
        self.element.hover(
              function () {
                    self._active = true;
                    self.element.cap.show("slide",
```

```
                        {direction:"left"},o.speed,function(){});
                },
                function () {
                    self.element.cap.hide('slide',
                        {direction:"right"},o.speed,function(){
                                    self._active = false;
                                    if(self._destroyCalled == true)
                                                self._vernietig();
                                    });
                }
            );
    },
…
```

Bespreking:

- in de eerste event handler van `hover()`, de `mouseenter` event handler, plaatsen we als eerste statement de var `_active` op `true`.
- bij de `mouseleave` event handler, **in de callback functie**, plaatsen we dezelfde var weer op `false` en checken de waarde van `_destroyCalled`: als deze `true` is, zal de widget niet vernietigd zijn en moet dat nog gebeuren.

Een derde aanpassing doen we aan *destroy* zelf:

```
destroy: function() {

        this._destroyCalled   = true;
        if(this._active == false){
            this._vernietig();
            this._destroyCalled   = false;
        }
}
```

Dit lost het probleem van de 'laatkomers' op.

6 Taken

Maak nu de eindtaak: vraag de opdracht aan je coach

7 Bijlage: Selecties met $()

Om een elementen te selecteren kan je gebruiken maken van een **CSS** of een **custom** selector. De syntax is

`$(selector, [context])` of `jQuery(selector, [context])`

Alle types kunnen in combinatie gebruikt worden.

De *expressie* is een CSS selector soms met een custom jquery selector erbij. Het optionele argument *context* kan een DOM element zijn (die je uiteraard ook met `$()` kan selecteren). De expressie wordt in aanhalingstekens gezet.

Enkele voorbeelden van CSS selectors en context:

selector	beschrijving
`$("*")`	alle elementen in het document
`$("p")`	alle `p` elementen in het document
`$(".kolom")`	alle elementen die de `class`'kolom' hebben
`$("#speciaal")`	een element met de `id` 'speciaal'
`$("#container p")`	alle `p` elementen in het element "container"
`$("p","#container")`	alle `p` elementen in het element "container" via *context*. Identiek aan vorige
`$("div>p")`	alle `p` elementen die direct child zijn van een `div`
`$("p, div, ul")`	alle `p` en `div` en `ul` elementen in het document
`$("input:radio")`	alle `input type="radio"` elementen in het document
`$("input:radio", document.frm1)`	alle `input type="radio"` elementen in het formulier met de name 'frm1'
`$("input:checkbox")`	alle `input type="checkbox"` elementen in het document
`$("input:checked")`	alle `input type="checkbox"` elementen in het document die aangevinkt zijn
`$("ul#eerstelijst>li:eq(1)")`	het tweede (index 1) `li` element van het `ul` element met de `id` 'eerstelijst'
`$("div:contains('Jan')")`	alle `div` elementen die de tekst 'Jan' bevatten
`$("input:not(:radio))`	alle `input` elementen die niet van het `type="radio"` zijn
`$("a[target]")`	alle `a` elementen met een attribuut `target`
`$("div[class~='rood']")`	alle `div` elementen met een attribuut `class` waarin het token *rood* voorkomt
`$("a[href$='.pdf']")`	alle `a` elementen met een attribuut `href` dat eindigt op *'.pdf'*

8 Bijlage: Overzicht methodes om inhoud in te voegen/verplaatsen/vervangen

Dit is een kort overzicht van de verschillende JQ methods die inhoud toevoegen/verplaatsen/vervangen **in, rond** of **naast** elementen van een *containerset*. Die *inhoud* is zelf een **element**, een jQ **set**, soms **HTML** of kan ook een **functie** zijn die de inhoud produceert.

Dikwijls zijn er twee identieke methods (bv. `after()` en `insertAfter()`) die enkel verschillen in syntax: **wat staat eerst**?

- ofwel de **containerset** waarop de method inwerkt
- ofwel de **inhoud** die toegevoegd/verplaatst wordt

dit is belangrijk omdat de **returnwaarde** na uitvoering verschilt: ofwel krijg je de containerset terug ofwel de (nieuwe) inhoud.

Deze methods voegen niet noodzakelijk nieuwe inhoud toe: als de 'inhoud' een set is die een aantal **aanwezige** elementen matched, dan worden die verplaatst.

method	Toevoegen BINNEN een containerelement	Beschrijving
`append()`	`$(containerset).append(inhoud)`	voegt de inhoudelementen toe als **laatste child** van elk el van de containerset. Returnt de containerset
`appendTo()`	`$(inhoud).appendTo(containerset)`	voegt de inhoudelementen toe als **laatste child** van elk el van de containerset. Returnt de inhoud
`prepend()`	`$(containerset).prepend(inhoud)`	voegt de inhoudelementen toe als **eerste child** van elk el van de containerset. Returnt de containerset
`prependTo()`	`$(inhoud).prependTo(containerset)`	voegt de inhoudelementen toe als **eerste child** van elk el van de containerset. Returnt de inhoud
`html()`	`$(containerset).html(htmlString)`	inhoud is een *htmlString*. Overschrijft de html binnen elk el van de containerset met de *htmlString.* Returnt de containerset. Gebruikt zonder *htmlString* leest de method de html
`text()`	`$(containerset).text(textString)`	inhoud is een *textString*. Overschrijft de tekst binnen elk el van de containerset met de *textString.* Returnt de containerset. Gebruikt zonder *textString* leest de method de tekstuele

		inhoud. Kan niet gebruikt worden op `input` element. Gebruik `val()`

method	Toevoegen BUITEN een containerelement	Beschrijving
`insertAfter()`	`$(inhoud).insertAfter(containerset)`	voegt de inhoudelementen toe onmiddellijk **na** elk el van de containerset. Returnt de inhoud.
`after()`	`$(containerset).after(inhoud)`	voegt de inhoudelementen toe onmiddellijk **na** elk el van de containerset. Returnt de containerset.
`insertBefore()`	`$(inhoud).insertBefore(containerset)`	voegt de inhoudelementen toe onmiddellijk **voor** elk el van de containerset. Returnt de inhoud
`before()`	`$(containerEl).before(inhoud)`	voegt de inhoudelementen toe onmiddellijk **voor** elk el van de containerset. Returnt de containerset

method	Toevoegen ROND de container	Beschrijving
`unwrap()`	`$(containerset).unwrap()`	**verwijdert** de **parent** van elk element van de containerset. Returnt de containerset.
`wrap()`	`$(containerset).wrap(verpakking)`	verpakt elke **el** van de containerset in de *verpakking* structuur (een element, htmlString, set). Returnt de containerset.
`wrapAll()`	`$(containerset).wrapAll(verpakking)`	verpakt de **volledige containerset** in de *verpakking* structuur(een element, htmlString, set), niet elk set-item afzonderlijk. Returnt de containerset.
`wrapInner()`	`$(containerset).wrapInner(verpakking)`	verpakt de **inhoud** van elk el van de containerset in de *verpakking* structuur (een element, htmlString, set). Returnt de containerset.

method	VERVANGEN	Beschrijving
replaceWith()	$(*containerset*).replaceWith(*inhoud*)	vervangt elk element in de containerset met de nieuwe inhoud. Returnt de set die verwijderd werd.
replaceAll()	$(*inhoud*).replaceAll(containerset)	identiek aan vorige method maar omgekeerde syntax. Returnt de nieuwe inhoud.

9 Bijlage: Tips voor meer efficiëntie

Eigenlijk wilden we niet *pateren (*preken), maar we doen het lekker toch! Dus geven we hier wat gratuite tips om je programmeertechniek en vooral de snelheid van je website te verbeteren.

Wat betreft **snelheid**, willen we erop wijzen dat je enkel zeker bent van een verbetering als de website gepubliceerd is op zijn vaste host. Snelheidstests op *localhost* of op een testserver kunnen onmogelijk de juiste waarden geven.

- Gebruik een **CDN** zoals Google, Microsoft of jQuery.com waarop de jquery libraries gehost staan, zowel de Core als de UI. Sommige plugins kan je er ook op vinden. Zie volgende bijlage

- **Combineer** je scripts en je plugins in één bestand (copy-paste alles in één)

  - sommige browsers zijn niet in staat parallel bestanden te downloaden, het gebeurt de een na de ander.

  - elk bestand (javascript, plugin, ..) dat je downloadt, betekent een apart HTPP request: dat neemt tijd in beslag.
    Eén gecombineerd bestand is natuurlijk de som van de aparte bestanden, maar wordt met één HTTP request geladen

- **Minify** je eigen JS bestanden en plugins.
  Minification verwijdert alle onnodige karakters – commentaar + witruimte - uit een bestand, daarbovenop kunnen sommige "compressors" ook variabelen hernoemen/inkorten en zelf de code verbeteren. Dit reduceert de bestandsgrootte ongeveer tot op de helft, zodat het bestand sneller downloadt.
  Veelgebruikte compressors zijn:

  - *JSMin* op http://crockford.com/javascript/jsmin

  - Dean Edwards *Packer* op http://dean.edwards.name/packer/

  - Google *Closure compiler* op
    http://code.google.com/intl/en-US/closure/compiler/

- Beperk het **aantal keer** dat je de DOM manipuleert, doe zoveel mogelijk achter de schermen en wijzig dan éénmaal. Doe eerder één grote ingreep dan 10 kleintjes.

10 Bijlage: jQuery via CDN

Client-side libraries laden via een ***Content Delivery Network*** (CDN) kan een grote perfomantieverbetering geven. Een CDN *hosts* de libraries voor jou.
Voordelen:

- ***caching***: de browser van de gebruiker leest slechts eenmaal de libs, bij een andere pagina stuurt de CDN de cache. De kans dat de gebruiker al eens op een site met jQuery geweest is, is zeer groot
- ***paralleldownloading***: de library wordt terzelfdertijd als je pagina gedownload, niet erna
- minder belasting van je eigen site
- De CDN heeft:
  - een fenomele response tijd
  - een kortere afstand tot de gebruiker
  - een 100% uptime

Nadelen:

- soms niet beschikbaar, maar zie verder voor lokale fallback

De manier waarop je een CDN gebruikt verschilt van host tot host:

jQuery CDN

jQuery zelf kan je als CDN gebruiken:

```
<script src="http://code.jquery.com/jquery-1.10.1.min.js"></script>
```

Op *code.jquery.com* zie je welke bestanden en versies je kunt koppelen. Ook de UI versies zijn beschikbaar.

Google CDN

De meest gebruikte CDN is die op Google omdat die het meest performant is.

```
<script src="//ajax.googleapis.com/ajax/libs/jquery/2.0.0/jquery.min.js"></script>
```

Bekijk even de documentatie op http://developers.google.com/.
Zoek de term "*hosted libraries*". Daar bemerk je dat o.a. de jQuery libraries aanwezig zijn.

Het is ook mogelijk om de **UI Theme stylesheets** te laden vanuit Google.
Je hebt dit stylesheet altijd nodig want zonder werken sommige *UI widgets* niet. Infeite staan alle themes op Google, alleen, Google adverteert ze niet.

Je vindt de locatie van alle themes altijd via de *jQuery UI Blog*:
http://blog.jqueryui.com/

Een voorbeeld:

```
<link href="http://ajax.googleapis.com/ajax/libs/jqueryui/1.10.3/themes/south-
  street/jquery-ui.min.css" rel="stylesheet" type="text/css"  />
```

Je kan het stylesheet ook zelf hosten natuurlijk.

Microsoft Ajax CDN

Ook op Microsoft wordt jQuery gehost. Je gebruikt het dan zo:

```
<script src="http://ajax.aspnetcdn.com/ajax/jquery/jquery-1.10.0.min.js"></script>
```

Ook de UI:

```
<script src="http://ajax.aspnetcdn.com/ajax/jquery.ui/1.8.22/jquery-ui.js"></script>
```

met de bijhorende stylesheet:

```
<link rel="Stylesheet" href="http://ajax.aspnetcdn.com/ajax/jquery.ui/1.8.10/themes/redmon
  d/jquery-ui.css" />
```

Lokale *fallback*:

Een CDN gebruiken heeft de voorkeur, maar als je gebruiker zich in een onzekere internetsituatie bevindt, of je werkt op een *localhost* zonder verbinding, dan kan een *fallback* op een lokale copy handig zijn.

Om dat te doen gaan we zo tewerk:

```
<script src="http://code.jquery.com/jquery-1.10.1.min.js"></script>
<script>
  (window.jQuery || ↵
        document.write('<script src="/scripts/jquery-1.10.1.min.js"><\/script>'));
  </script>
```

Bespreking:

- de eerste `script` tag laadt de CDN copy
- de tweede tag test of er een `jQuery` object bestaat en zoniet
- schrijft een nieuw `script` tag in het document waar een lokale copy geladen wordt

11 Bijlage: Internet referenties

url	Website
jQuery	
jquery.com	jQuery docs, downloads,…
jqueryui.com	jQuery User Interface
api.jquery.com	de documentatie om de syntax en werkwijze van methods, properties, events na te gaan
CDN	
code.google.com/apis/libraries/devguide.html#jqueryUI	jQuery op Google API
javascript	
developer.mozilla.org/en/docs/Javascript	Mozilla developer center: Javascript
www.w3.org/TR/DOM-Level-2-HTML/html.html	Document Object Model HTML
www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/	DOM Level2 Core
www.w3.org/TR/2000/REC-DOM-Level-2-Events-20001113/	DOM Level2 Events
www.json.org	JSON
www.jslint.com	JSLint, the Javascript verifier
www.whatwg.org/specs/web-apps/current-work/multipage/	HTML5 standard

12 COLOFON

Sectorverantwoordelijke: Ortaire Uyttersprot

Cursusverantwoordelijke: Jean Smits

Didactiek en lay-out: Jan Vandorpe

Medewerkers: Jan Vandorpe
Adinda Mattens

Versie: juli 2013

Nummer dotatielijst: