# Efficiënt scripten met jQuery 1.10

## Ontwerpen van interactieve websites met HTML5, CSS3 en jQuery

Versie 1.0 - juni 2013

Speciale dank aan Lieve en Guy voor hun meer dan gewaardeerde medewerking.

# Inhoud

# 4 Elementen selecteren.

Een jQuery statement bestaat steeds uit twee of meer delen. Het eerste deel is altijd een selector. De volgende delen beschrijven de acties die u op de selector uitvoert.

Een voorbeeld:

```
$('p.foo').addClass('bar').show('slow');
```

- `$('p.foo')`: selecteer alle p-tags met het class-attribuut *foo*.
- `.addClass('bar')`: voeg aan deze selectie de class *bar* toe.
- `.show('slow');` en maak de selectie zichtbaar.

Een goede selectie maken is bijgevolg cruciaal om een krachtig script uit te voeren.

JavaScript kent een zeer beperkte set aan selectors. U kan bijvoorbeeld een bepaald id (*document.getElementById(id)*) of een bepaalde tag opvragen (*document.getElementByTagName(tag)*), maar wat als u de derde rij in een tabel wilt ophalen of alle links met een externe URL? Dit is onmogelijk met één commando.

Dit hoofdstuk beschrijft de belangrijkste selectiemethodes binnen jQuery. De syntax is eenvoudig. Deze volgt immers de selectiemethodes van CSS3, aangevuld met enkele jQuery-specifieke selectiemethodes.

De voorbeelden die volgen, kan u uittesten op het oefenbestand *selectorTest.html*.

» Open **selectors / selectorTest.html**.

Geef in het tekstveld een selectie in ( bv: *p#paragraaf1*). Het script op de pagina gaat op zoek naar de selectie en voegt aan de selectie de class *highLight* toe. Deze class tekent een groen kader rond de selector, maakt de achtergrond geel en kleurt de tekst rood.


Een overzicht van de verschillende selectors vindt u hier:
*http://api.jquery.com/category/selectors/basic-css-selectors/*

## 4.1 Basis selectors.

Zie: *http://api.jquery.com/category/selectors/basic-css-selectors/*

### *4.1.1 Tags, classes en id's.*

Volgende voorbeelden gebruikt u waarschijnlijk dagelijks in CSS:

- `$('a')`: selecteer alle a-tags.
- `$('a, tr')`: selecteer alle a-tags en alle tr-tags (let op de komma als scheidingsteken).
- `$('p#paragraaf1')`: selecteer de p-tag met id *paragraaf1*.
- `$('#paragraaf1')`: selecteer id *paragraaf1*. De selectie is natuurlijk identiek aan het voorgaande. Een id mag immers maar eenmaal voorkomen op de pagina.
- `$('.sublist')`: selecteer alle tags met class *sublist*.

### *4.1.2 Universele selector.*

- `$('*')`: selecteer alle tags (Het sterretje is, net zoals in CSS, de universele selector voor alle tags.)

## 4.2 Hiërarchische selectors.

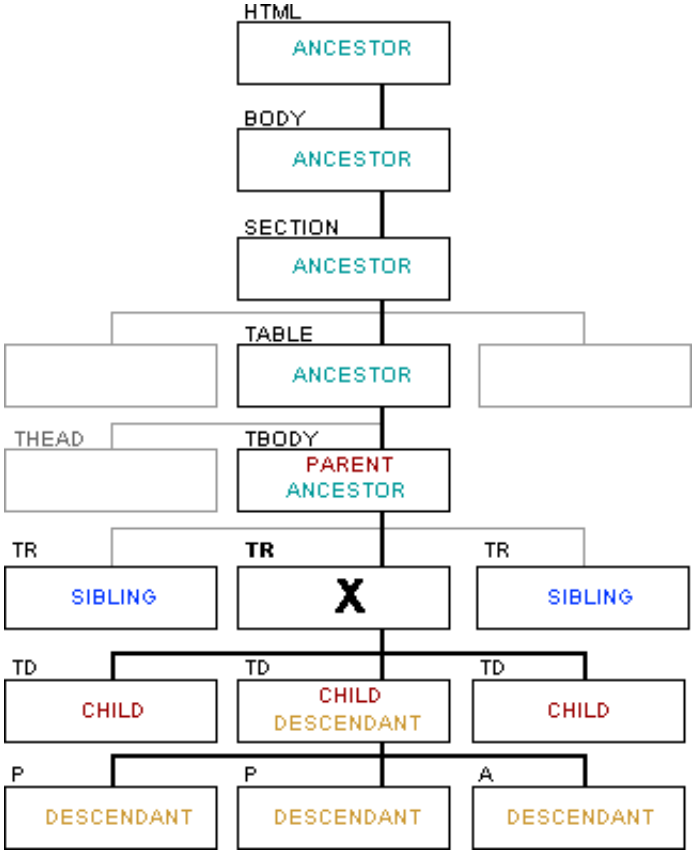
Zie: *http://api.jquery.com/category/selectors/hierarchy-selectors/*

Hiërarchische selectors kan u best vergelijken met een stamboom. Bekijk in onderstaande code hoe de tags in elkaar genest zijn.

```
<body>
  <table class="layoutTabel">
   <thead>
    <tr>
      <th>Opleiding:</th>
      <th>Datum:</th>
      <th>Lokaal:</th>
    </tr>
   </thead>
   <tbody>
    <tr>
      <td>Dreamweaver</td>
      <td>15 december</td>
      <td>PC1</td>
    </tr>
    <tr>
      <td>PHP</td>
      <td>11 december</td>
      <td>PC2</td>
    </tr>
   </tbody>
  </table>
 </body>
```

De hiërarchische selectors zijn:

- **Parent** en **child** zijn geneste tags op het eerste niveau.
  De thead-tag en de tbody-tag staan rechtstreeks onder de table-tag en gedragen zich in een *parent - child* relatie.
  *table* is een parent van *thead* of
  *thead* is een child van *table*.
- **Siblings** (broers of zussen) zijn elementen die op gelijk niveau staan.
  De drie td-tags binnen een tr-tag zijn *siblings* van elkaar.
- **Ancestor** (voorouder).
  Een *ancestor* kan de *parent* zijn, maar ook de *parent* van de *parent*, of nog een verdere relatie.
  *th* heeft als parent *tr*.
  *th* heeft als ancestor *tr*, maar ook *thead*, *table* en *body*.
- **Descendant** (nakomeling).
  Een *descendant* kan een *child* zijn, maar ook een *child* van een *child* of nog een verdere relatie.
  *table* heeft als child *thead* en *tbody*.
  *table* heeft als descendant *thead*, *tbody*, maar ook *tr*, *th* en *td*.




» Controleer onderstaande selecties op het oefenbestand. Enkel het deel tussen de aanhalingstekens invullen. Bijvoorbeeld `table tr` en niet `$('table tr')`

#### *4.2.1 Descendant selector: $('ancestor descendant').*

- `$('table tr')`: selecteer alle tr-tags binnen een tabel.
- `$('table td')`: selecteer de td-tags binnen een tabel.
- `$('table td a')`: selecteer binnen de tabel enkel de links in td-tags.
- `$('ul *')`: selecteer alle tags binnen de ul-tag.

#### *4.2.2 Child selector: $('parent > child').*

- `$('table > tbody')`: selecteer de tbody-tags binnen een tabel.
- `$('table > td')`: selecteer de td-tags binnen een tabel. In ons voorbeeld zit er geen enkele td-tag rechtstreeks onder de table-tag. Er wordt dus niets geselecteerd.
- `$('table > tbody > tr')`: selecteer alle rijen binnen de tbody-tag van de tabel.

#### *4.2.3 Sibling selector (alle volgende aangrenzende) : $('prev ~ sibling').*

- `$('thead ~ tbody')`: selecteer alle tbody-tags die volgen op de thead-tag.
- `$('li ~ li')`: selecteer alle li-tags die volgen op de eerste li-tag. De eerste li-tag zelf wordt niet geselecteerd.
- `$('h3 ~ p')`: selecteer alle p-tags die volgen op de eerste h3-tag.

#### *4.2.4 Sibling selector (dadelijk aangrenzende): $('prev + next').*

- `$('thead + tbody')`: selecteer alle tbody-tags die DADELIJK volgen op de thead-tag.
- `$('li + li')`: selecteer alle li-tags die DADELIJK volgen op de eerste li-tag. De eerste li-tag zelf wordt niet geselecteerd.
- `$('h3 + p')`: selecteer alle p-tags die DADELIJK volgen op de eerste h3-tag. Tussen paragraaf 1 en paragraaf 2 staat nog een lijst. Paragraaf 2 wordt dus niet geselecteerd.

## 4.3 Basisfilters.

Zie : *http://api.jquery.com/category/selectors/basic-filter-selectors/*

De selectors die we tot hiertoe hebben besproken, selecteren telkens een volledig element. `$('tr')` selecteert bijvoorbeeld alle tr-tags in een tabel.

Stel dat we nu enkel de even of oneven rijen in een tabel wensen te selecteren, dan doen we dit aan de hand van filters. Een filter is als het ware een subselectie binnen een hoofdselectie. Alle filters worden met een dubbelpunt als prefix aangeduid. Voor de oneven rijen in een tabel wordt dit: `$('tr:odd')`.

Bij sommige filters moet u eveneens een index meegeven. Het eerste element heeft de index *0*. Om bijvoorbeeld de derde rij binnen een tabel te selecteren, geeft u als index de waarde *2* mee `$('tr:eq(2)')`.

» Controleer onderstaande filters op het oefenbestand.

#### *4.3.1 Subselecties binnen een reeks.*

- `$('table tr:even')`: selecteer alle even tr-tags binnen een tabel.
- `$('table tr:odd')`: selecteer alle oneven tr-tags binnen een tabel.
- `$('img:first')`: selecteer de eerste afbeelding op de pagina.
- `$('img:last')`: selecteer de laatste afbeelding op de pagina.

- `$('img:eq(2)')`: selecteer de derde afbeelding op de pagina.
- `$('img:lt(2)')`: selecteer alle afbeeldingen voor de derde afbeelding op de pagina.
- `$('img:gt(2)')`: selecteer alle afbeeldingen na de derde afbeelding op de pagina.

#### *4.3.2 Inverse subselecties binnen een reeks met :not().*

- `$('img:not(:eq(2))')`: selecteer alle afbeeldingen, behalve de derde afbeelding.

#### *4.3.3 Overige subselecties.*

(De twee laatste selectors kan u niet op het oefenbestand testen.)

- `$('*:header')` of `$(':header')`: selecteer alle hx-tags (h1 t/m h6).
- `$('*:focus')` of `$(':focus')`: selecteer het element dat op dit moment geselecteerd is.
- `$('div:animated')`: selecteer alle div-tags die op dit moment bewegen.

## 4.4 Inhoudsfilters.

Zie : *http://api.jquery.com/category/selectors/content-filter-selector/*

Deze filters testen op de inhoud (tekst of html) binnen een bepaald element.

#### *4.4.1 Het element bevat een bepaalde tekst :contains().*

- `$('td:contains("PC")')`: selecteer binnen een tabel alle cellen die het woord *PC* bevatten.

  ! De tekst is hoofdlettergevoelig. De aanhalingstekens rond *PC* zijn niet verplicht.
- `$('td:not(:contains("PC"))')`: selecteer binnen een tabel alle cellen, behalve de cellen die het woord *PC* bevatten.

#### *4.4.2 Het element bevat een bepaald element :has().*

- `$('td:has(a)')`: selecteer binnen een tabel alle cellen die een link bevatten.
- `$('td:not(:has(a))')`: selecteer binnen een tabel alle cellen die geen link bevatten.

#### *4.4.3 Lege (:empty) of niet lege (:parent) elementen.*

- `$('td:empty')`: selecteer alle lege cellen binnen een tabel.
- `$('td:parent')`of `$('td:not(:empty)')`: selecteer alle cellen die niet leeg zijn.

## 4.5 Attribuut selectors.

Zie : *http://api.jquery.com/category/selectors/attribute-selectors/*

Vanuit CSS kan u elementen selecteren op basis van hun attribuut. Ook deze eigenschap werd door jQuery overgenomen.

! Waardes van een attribuut altijd tussen aanhalingstekens plaatsen.

Bekijk de broncode op het oefenbestand en controleer de verschillende attributen op de tags. Neem bijvoorbeeld een afbeelding. Alle afbeeldingen hebben natuurlijk een src-attribuut en een alt-attribuut. Sommige afbeeldingen hebben ook het title-attribuut.

#### *4.5.1 Attribuut bestaat [name].*

- `$('img[src]')`: selecteer alle afbeeldingen met het src-attribuut.
- `$('img[title]')`: selecteer alle afbeeldingen met het title-attribuut.
- `$('a[target]')`: selecteer alle links met het target-attribuut.
- `$('a:not([target])')`: selecteer alle links zonder target-attribuut.

We kunnen een selectie nog verder verfijnen door de waarde of inhoud van het attribuut te controleren.

#### *4.5.2 Attribuut is gelijk aan [name="value"].*

- `$('a[target="_blank"]')`: selecteer alle links met *target= "_blank"*.

#### *4.5.3 Attribuut is niet gelijk aan [name!="value"].*

- `$('a[target!="_blank"]')`: selecteer alle links zonder *target= "_blank"*.
  (Dus ook alle links zonder target-attribuut.)

#### *4.5.4 Attribuut begint met [name^="value"].*

- `$('a[href^="http://"]')`: selecteer alle externe links.

#### *4.5.5 Attribuut eindigt op [name$="value"].*

- `$('a[href$=".com"], a[href$=".com/"]')`: selecteer alle externe links naar een *.com* domein.

#### *4.5.6 Attribuut bevat [name*="value"].*

- `$('a[href*="wiki"]')`: selecteer alle links die het woord *wiki* bevatten.

#### *4.5.7 Attribuut bevat meerdere filters [filter1][filter2].*

- `$('a[href^="http://"][title]')`: selecteer alle externe links die ook een title-attribuut bevatten.

## 4.6 Childfilters.

Zie : *http://api.jquery.com/category/selectors/child-filter-selectors/*

Een childfilter selecteert het n-de element binnen het parent element.

#### *4.6.1 Eerste child element :first-child.*

- `$('p strong:first-child')`: selecteer de eerste strong-tag binnen ELKE paragraaf.
- `$('p strong:first')`: selecteer de eerste strong-tag op de EERSTE paragraaf.

#### *4.6.2 Laatste child element :last-child.*

- `$('p strong:last-child')`: selecteer de laatste strong-tag binnen ELKE paragraaf.
- `$('p strong:last')`: selecteer de laatste strong-tag op de LAATSTE paragraaf.

#### *4.6.3 Eerste sibling element binnen dezefde parent :first-of-type.*

- `$('a:first-of-type')`: selecteer de eerste a-tag binnen dezelfde parent.

#### *4.6.4 Laatste sibling element binnen dezefde parent :last-of-type.*

- `$('a:last-of-type')`: selecteer de eerste a-tag binnen dezelfde parent.

#### *4.6.5 Het n-de child elementen.*

- `$('p strong:nth-child(2)')`: selecteer de tweede strong-tag binnen ELKE paragraaf.
  ! De index start hier vanaf *1* en niet vanaf *0*!
- `$('p strong:nth-child(even)')`: selecteer alle even strong-tags binnen ELKE paragraaf.
- `$('p strong:nth-child(3n)')`: selecteer alle strong-tags die een veelvoud zijn van 3. Dus strong-tag 3, 6, 9, enz ...
- `$('p strong:nth-child(2n+1)')`: selecteer strong-tag 1 (=2*0+1), 3 (=2*1+1), 5, 7, enz ...
- `$('p strong:nth-last-child(2)')`: selecteer de voorlaatste strong-tag binnen ELKE paragraaf.
- `$('li:nth-of-type(even)')`: selecteer alle even li-tag binnen dezelfde parent.
- `$('a:nth-last-of-type(2)')`: selecteer de voorlaatste a-tag binnen dezelfde parent.

#### *4.6.6 Het enige child element :only-child().*

- `$('p strong:only-child()')`: selecteer de strong-tag enkel indien dit HET ENIGE child element is binnen de p-tag.

#### *4.6.7 Het enige child element dezefde parent :only-of-type().*

- `$('strong:only-of-type()')`: selecteer de strong-tag enkel indien dit HET ENIGE child element is binnen dezelfde parent.

## 4.7 Formulierfilters.

Zie : *http://api.jquery.com/category/selectors/form-selectors/*

Filtert formulierelementen van een bepaald type of met een bepaald attribuut.

#### *4.7.1 De type-selector (button, submit, reset, checkbox, ...).*

- `$('input')`: selecteer ENKEL de input-tags.
- `$('input[type="button"]')`: selecteer alle input-tags met *type="button"*.
- `$('input[type="submit"]')`: selecteer alle input-tags met *type="submit"*.
- `$('input[type="reset"]')`: selecteer alle input-tags met *type="reset"*.

#### *4.7.2 Overige attributen (:checked, :disabled, :enabled en :selected).*

- `$(':checked')`: selecteer alle checkboxen en alle radiobuttons met een *checked-attribuut*. (werkt niet op een keuzelijst.)
- `$(':disabled')`: selecteer alle elementen met een *disabled-attribuut*.
- `$(':enabled')`: selecteer alle elementen die niet disabled staan.
- `$(':selected')`: selecteer het gekozen item uit een keuzelijst. (Werkt niet op checkboxen en radiobuttons.)

! Met `:checked` en `:selected` selecteert u enkel het element, maar haalt u de gekozen

waarde nog niet op. De methodes om deze waardes uit te lezen, komen later aan bod.

## 4.8 Zichtbaarheidsfilters.

Zie : *http://api.jquery.com/category/selectors/visibility-filter-selectors/*

- `$('p:visible')`: selecteer alle zichbare p-tags.
- `$('p:hidden')`: selecteer alle onzichtbare p-tags.

## 4.9 Toepassing 1: gemeentelijst filteren (basisversie).

» Open **selectors/toep_gemeentelijst_1.html**.


De pagina bevat een lijst van alle Vlaamse gemeenten. Het zoekveld bovenaan de pagina filtert de lijst met gemeentenamen.

```
<body>
 ...
 <input type="text" name="filter" id="filter" ... ">
 ...
 <ul>
    <li><a ... >Aalst</a></li>
    <li><a ... >Aalter</a></li>
    <li><a ... >Aarschot</a></li>
    <li><a ... >Aartselaar</a></li>
 </ul>
 ...
</body>
```

» Pas het script als volgt aan.

```
<script>
$(function() {
    $('#filter').keyup(function(){
       var filter = $(this).val();
       $('li').hide();
       $('li:contains(' + filter + ')').show();
    });
});
</script>
```

Telkens u in het tekstveld een letter intypt, wordt het event `keyup()` geactiveerd. De waarde uit het tekstveld komt in de variabele *filter* terecht. Indien u bijvoorbeeld *ka* invult, wordt getest of dit woord binnen een li-tag voorkomt. De twee laatste selectoren bepalen of de li-tag zichtbaar of onzichtbaar wordt.

Merk op dat de variabele dynamisch in de selector wordt verwerkt. Voor het zoekwoord *ka* wordt dit:

```
$('li:contains(ka)').show();
```

» Test de pagina in een browser.
» Maak het tekstveld met de backtoets leeg.

Bij een leeg tekstveld is de variabele *filter* leeg. Geen van beide selectoren geeft een resultaat, zodat niet alle gemeentenamen terug zichtbaar worden. Dit kunnen we makkelijk oplossen door eerst de inhoud van *filter* te controleren. Indien *filter* geen waarde bevat, tonen we altijd de volledige lijst.

» Pas het script als volgt aan en test het resultaat in een browser.

```
<script>
$(function() {
    $('#filter').keyup(function(){
       var filter = $(this).val();
       if (filter != '') {
          $('li').hide();
          $('li:contains(' + filter + ')').show();
       } else {
          $('li').show();
       }
    });
});
</script>
```

! Merk op dat zoeken op *Ka* of op *ka* een totaal verschillend resultaat oplevert. De zoekfunctie is namelijk hoofdlettergevoelig. Met de selecties uit dit hoofdstuk kunnen we dit nog niet oplossen. In het volgend hoofdstuk gaan we dit probleem wegwerken door de selecties verder te verfijnen.

## 4.10 Bronnen.

- **Overzicht jQuery selectors**:
  http://api.jquery.com/category/selectors/

# 9 Inleiding tot AJAX.

Alle moderne websites maken tegenwoordig gebruik van AJAX. Denk maar aan Gmail, Google Drive, Facebook en Twitter.

In tegenstelling tot een klassieke webpagina worden bij AJAX-gestuurde pagina's delen van de pagina ververst zonder de volledige webpagina opnieuw in te laden. De pagina's worden hierdoor veel interactiever en intuïtiever. De gebruiker krijgt meer het gevoel dat hij in een desktopapplicatie werkt dan op een website. Websites die hoofdzakelijk gebruik maken van AJAX noemt men ook wel **R**ich **I**nternet **A**pplicaties of RIA's.

AJAX is geen technologie op zich, maar een algemene term voor het ontwerpen van interactieve pagina's waarbij gegevens uit een extern bestand worden opgehaald en vervolgens dynamisch worden getoond. Het grote voordeel van AJAX t.o.v. uitsluitend klassieke webpagina's is dat alle gegevens dynamisch in de browser worden geladen. Alle interactiviteit (sorteren, filteren, ...) gebeurt dus volledig binnen de browser zonder dat er een nieuwe connectie met de server nodig is.

De communicatie binnen een klassieke website gebeurt als volgt:

1. De browser laadt een pagina.
2. De gebruiker klikt op een link.
3. De browser vraagt een volledig nieuwe pagina op.
4. De webserver stuurt de nieuwe pagina naar de browser.
5. De browser toont de nieuwe pagina.

De communicatie met een AJAX-pagina verloopt als volgt:

1. De browser laadt een pagina.
2. De gebruiker klikt op een link.
3. De JavaScript engine verwerkt de aanvraag en stuurt de aanvraag in de achtergrond door naar een externe pagina.
4. De JavaScript engine ontvangt de gegevens en past het DOM aan.
4. De browser toont de nieuwe gegevens en hoeft daarvoor niet de volledige pagina te herladen.

Omdat de browser minder met de webserver communiceert en omdat de verwerking volledig lokaal gebeurt, krijgen we een zeer snelle respons en een snelle pagina update.

In dit hoofdstuk geven we een korte inleiding tot AJAX en verduidelijken we enkele begrippen. De oefeningen komen in de twee volgende hoofdstukken aan bod.

Een volledig overzicht van alle AJAX-methodes vindt u hier:
*http://api.jquery.com/category/ajax/*

## 9.1 Historiek.

De techniek om asynchroon gegevens op te halen, bestaat al meer dan 20 jaar. In 1998 ontwikkelde Microsoft een systeem om via een *ActiveX control* gegevens in de achtergrond op te halen. Enkele jaren later implementeerden alle andere browsers een gelijkaardig principe, maar nu gebaseerd op het gestandaardiseerde *XMLHttpRequest protocol* (kortweg *XHR*).

Google was het eerste bedrijf dat XHR op grote schaal in zijn toepassingen ging verwerken. In de beginjaren kwamen de externe gegevens uitsluitend uit een XML-document. Zo is de term AJAX ontstaan. AJAX was het acroniem voor **A**synchronous **J**avaScript **A**nd **X**ML.

Ondertussen is de omschrijving van AJAX al achterhaald. Via AJAX kunnen we niet enkel XML importeren, maar eveneens tekst, HTML, JSON en JavaScript.

Het is niet zo evident om op een universele manier AJAX te verwerken. De ene browser gebruikt XHR, de andere ActiveX. Gelukkig hoeven wij ons hier geen zorgen over te maken. jQuery zorgt immers voor een correcte verwerking in de verschillende browsers.

## 9.2 Wat is XML?

**XML** is het acroniem voor e**X**tensible **M**arkup **L**anguage.
Net zoals HTML, beschrijft XML de datastructuur van gegevens, niet de opmaak. Een XML-document is, net zoals HTML, opgebouwd met tags (*nodes* in het XML jargon) en attributen.

Neem als voorbeeld een adresboek in de vorm van een XML-document.

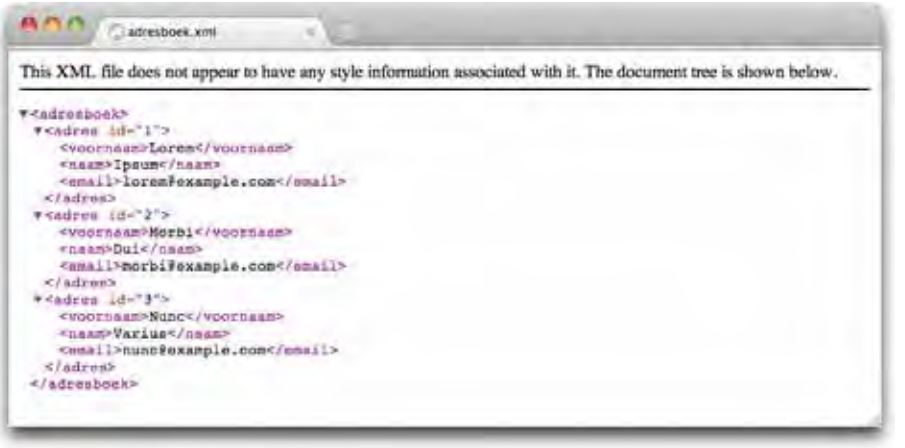
XML begint steeds met de header of proloog. De proloog bevat informatie over de document encoding en de XML versie. Dit is ondermeer belangrijk voor het programma dat de XML-code gaat verwerken (de XML-parser). Na de proloog volgt de rootnode *adresboek*.
Ons adresboek bevat drie personen. Elke persoon staat beschreven in een eigen adresnode met als attribuut een uniek *id*. Binnen de adresnode komen de childnodes: *voornaam*, *naam* en *email*.

```
<?xml version="1.0" encoding="UTF-8"?>
<adresboek>
   <adres id="1">
      <voornaam>Lorem</voornaam>
      <naam>Ipsum</naam>
      <email>lorem@example.com</email>
   </adres>
   <adres id="2">
      <voornaam>Morbi</voornaam>
      <naam>Dui</naam>
      <email>morbi@example.com</email>
   </adres>
   <adres id="3">
      <voornaam>Nunc</voornaam>
      <naam>Varius</naam>
      <email>nunc@example.com</email>
   </adres>
</adresboek>
```

Een goed gestructureerd XML-document is zelfbeschrijvend. Dit wil zeggen dat nodenamen iets vertellen over de inhoud van de node.

Als het document voldoet aan alle syntaxregels van XML, noemt men dit **Well-formed** (of goed gevormd). Een well-formed document kan door de meeste parsers, zoals een webbrowser, correct verwerkt worden.

Een well-formed document kan u makkelijk in Firefox of in Chrome testen. Open het XML-document in de browsers. Als de boomstructuur verschijnt, is het document well-formed en zijn de gegevens te verwerken via jQuery.




Indien het document fouten bevat, is het niet well-formed en krijgt u dit te zien:


In een XML-document dat uitsluitend voor eigen gebruik is ontworpen, kan u de nodenamen vrij kiezen. In een universeel, gestandaardiseerd XML-document zoals RSS, ATOM en XHTML ligt de naamgeving vast.

Om de correctheid van het document te controleren, maakt men gebruik van een **DTD** of van een **XML Schema**.

Een DTD of XML schema documenteert als het ware het XML-bestand. Hierin wordt ondermeer beschreven welke nodes in het XML-document moeten/mogen voorkomen en welke inhoud de nodes bevatten (tekst, enkel getallen, …). Aan de hand van dit controlebestand kan de parser de XML-nodes zowel op syntax als op inhoud valideren.

Een well-formed XML-bestand dat ook nog voldoet aan de bijbehorende DTD of Schema, noemt men een **valid XML-bestand**.

## 9.3 Wat is JSON?

**JSON** is het acroniem voor **J**ava**S**cript **O**bject **N**otation. JSON beschrijft, net zoals XML, de datastructuur van gegevens. JSON maakt gewoon onderdeel uit van JavaScript, en is bijgevolg relatief eenvoudig in het DOM te verwerken.

De eenvoud van JSON heeft geleid tot een grote populariteit ervan, met name als een alternatief voor XML. Binnen JSON staan gegevens gestructureerd in de vorm van een JavaScriptobject of als een JavaScript array.

Ziehier een JSON equivalent van ons XML adresboek.

```
{"adresboek":[
     {
       "id":1,
       "voornaam":"Lorem",
       "naam":"Ipsum",
       "email":"lorem@example.com"
     },
     {
       "id":2,
       "voornaam":"Morbi",
       "naam":"Dui",
       "email":"morbi@example.com"
     },
     {
       "id":3,
       "voornaam":"Varius",
       "naam":"Nunc",
       "email":"nunc@example.com"
     }
  ]
}
```

T Vergelijk de syntax van bovenstaand JSON object met de syntax van een object literal in paragraaf 2.11.4.

! In JSON moeten alle namen en waardes tussen dubbele aanhalingstekens staan, zoniet kan jQuery het bestand niet verwerken!

- Goed: `"voornaam":"Lorem"`
- **Fout**: `voornaam:"Lorem"`
- **Fout**: `'voornaam':"Lorem"`
- **Fout**: `'voornaam':'Lorem'`

## 9.4 Requests filteren met GET en POST.

Het is perfect mogelijk om gegevens (HTML, JSON, XML, ...) te verwerken uit een volledig statisch document. Het wordt natuurlijk nog interessanter indien we de gegevens dynamisch vanuit een database kunnen genereren. Vanuit een statische pagina is de response altijd hetzelfde. Op een dynamische pagina is de response afhankelijk van een filter of parameter. Het filteren gebeurt meestal vanuit een formulier.

Neem bijvoorbeeld de zoekfunctie van Google. De response is afhankelijk van de zoekterm die u in het formulier invult.

Zoals u weet, kan u een formulier verzenden via GET of via POST.

### *9.4.1 Formulier verzenden met de GET-methode.*

**GET** plakt alle formuliervelden achter de URL. Alle zoekmachines gebruiken GET.

» Zoek via Google op **jquery tutorials** en bekijk de URL.

```
http://www.google.be/search?...&q=jquery+tutorials&.....
```

De structuur van de URL met een GET-request is als volgt:
**action?naam1=waarde1&naam2=waarde2&naam3=waarde3**

*Action* is de URL van de pagina die het formulier zal verwerken. Van elk formulierveld wordt zowel de naam van het veld als de waarde in het veld aan de URL toegevoegd. Alle velden zijn gescheiden door een &-teken.

Omdat GET de waarde of de inhoud van elk object zichtbaar maakt in de URL, kan u deze methode bijvoorbeeld niet gebruiken op een loginpagina. Het paswoord mag immers niet zichtbaar zijn in de URL. De lengte van de URL is ook beperkt tot enkele honderden karakters (browserafhankelijk).

### *9.4.2 Formulier verzenden met de POST-methode.*

De tweede methode, **POST**, geeft de gegevens via de header door. De gegevens zijn nu niet zichtbaar in de URL. Het aantal karakters dat kan worden doorgestuurd, is vrijwel onbeperkt.

! Voor een AJAX request naar een statische pagina hebt u geen webserver nodig. Voor een AJAX request met GET- en POST-variabelen in combinatie met een database hebt u altijd een webserver nodig. Formuliergegevens kan u dan enkel verwerken via een server-side scripttaal zoals PHP of ASP.NET. Hierdoor bent u wel verplicht om via een webserver te werken.

### *9.4.3 GET-methode zonder formulier.*

Omdat de GET-methode alle parameters achter de URL plaatst, is het perfect mogelijk om de parameters dadelijk in een link te verwerken. Dit wordt vaak gebruikt in een master/detail relatie.

De masterpagina toont een beknopt overzicht van alle items. Elk item heeft een link naar de detailpagina en stuurt in de URL een unieke identificatiecode mee.

Neem als voorbeeld de startpagina van Campinia Media (http://www.campiniamedia.be).


Elke link "Lees verder" verwijst naar dezelfde detailpagina:
http://www.campiniamedia.be/fondslijst_detail.asp?**ISBN=xxxxxx**

De parameter *ISBN* filtert de juiste gegevens uit de database en toont de gedetailleerde informatie over het gevraagde boek.

## 9.5 Zes soorten AJAX requests.

De methode `$.ajax()` is een jQuery's low-level AJAX implementatie. Dit is tevens de meest uitgebreide, maar ook moeilijkste methode.

Gelukkig zijn er ook nog vijf afgeleide methodes met minder toeters en bellen, maar wel veel eenvoudiger te begrijpen en te gebruiken. Onderstaande tabel geeft een overzicht van de verschillende methodes met hun mogelijkheden en beperkingen.

| | Moeilijkheidsgraad | GET request | POST request | Response | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | Tekst | HTML | Script | JSON(P) | XML |
| $(selector).**load**(url, [data], [callback]) | * | x | x | x | x | | | |
| $.**getScript**(url, [callback]) | ** | | | | | x | | |
| $.**getJSON**(url, [data], [callback]) | ** | x | | | | | x | |
| $.**get**(url, [data], [callback], [dataType]) | ** | x | | x | x | x | x | x |
| $.**post**(url, [data], [callback], [dataType]) | ** | | x | x | x | x | x | x |
| $.**ajax**(settings) | *** | x | x | x | x | x | x | x |

Zonder webserver kan u enkel koppelen met statische bestanden. Om gegevens uit een database te verwerken, zal u altijd server-side script moeten gebruiken en bent u natuurlijk wel verplicht om een webserver te gebruiken.

Omdat niet iedereen vertrouwd is met server-side scripts (PHP, ASP.NET, ...) gaan we de oefeningen dadelijk opsplitsen over twee verschillende hoofdstukken.

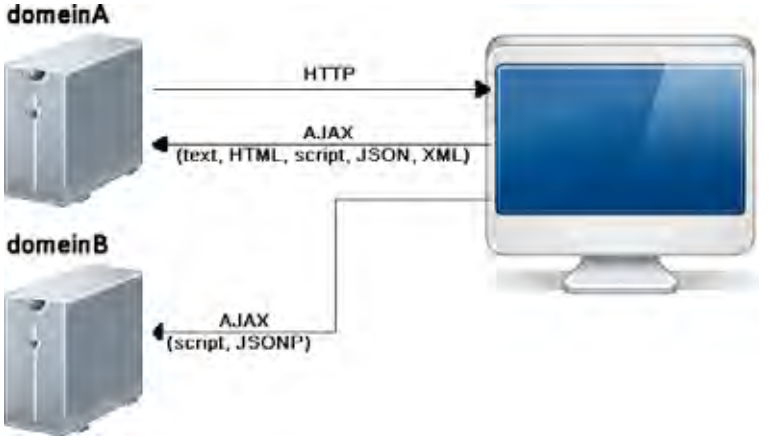
## 9.6 Same origin policy.

Uit veiligheidsredenen bevatten alle browsers enkele beperkingen. Eén van deze beperkingen is de *same origin policy*. Of, met andere woorden, alle gegevens die u via AJAX ophaalt, moeten afkomstig zijn van hetzelfde domein. Hetzelfde domein betekent meer specifiek: dezelfde server, zelfde protocol, zelfde domeinnaam en dezelfde poort. Het is bijvoorbeeld niet mogelijk dat een webpagina op *domeinA* gegevens ophaalt uit *domeinB*.

Er zijn echter twee uitzonderingen. Het ophalen van externe JavaScripts en gegevens in JSON**P** fomaat (let op de P achteraan) zijn wel toegestaan.

JSONP staat voor "JSON with Padding". Wanneer de webserver van *domeinB* zodanig staat geconfigureerd dat deze toelaat dat andere gebruikers zijn gegevens in JSON-formaat mogen ophalen, spreekt men van JSONP. De structuur van een JSON-bestand en van een JSONP-bestand is identiek.

Onderstaande afbeelding toont de AJAX mogelijkheden/beperkingen, rekening houdend met de *same origin policy*.

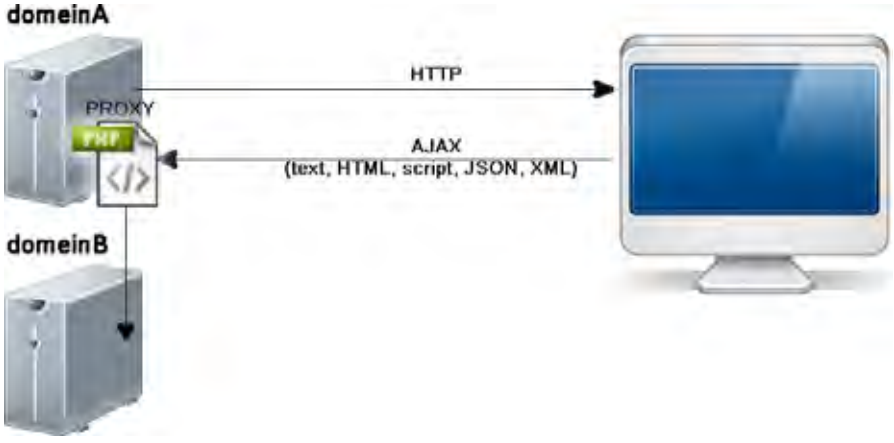



! Same origin policy is een beperking van de browser. Ook indien u zonder webserver werkt (domeinA is dan de lokale computer) geldt bovenstaand schema.

## 9.7 Cross-site scripting.

Indien u over een webserver beschikt, kan u de *same origin policy* makkelijk omzeilen. Op de webserver plaatst u een proxyscript. Dit proxyscript haalt de inhoud van een externe pagina (html, XML, JSON, ...) op en toont dit in zijn eigen pagina.

In plaats van een AJAX request te verwijzen naar de externe pagina, verwijst u nu naar de proxypagina. Voor de browser lijkt het alsof de gegevens afkomstig zijn van het eigen domein. Op deze manier kunnen we dus alle externe gegevens perfect via AJAX verwerken. Hierover later meer.




! Het proxyscript is een server-side script en is niet gebonden aan de *same origin policy*.

## 9.8 Bronnen.

- **jQuery AJAX methodes**
  http://api.jquery.com/category/ajax/
- **AJAX terminologie**
  http://en.wikipedia.org/wiki/Ajax_(programming)
  http://en.wikipedia.org/wiki/JSON
  http://en.wikipedia.org/wiki/JSONP
  http://en.wikipedia.org/wiki/Same_origin_policy
  http://en.wikipedia.org/wiki/Cross-site_scripting