hogeschool vives

Web development I – JS labo 4

studiegebied **HWB**

bachelor in het **toegepaste Informatica**

campus **Kortrijk**

academiejaar **2022-2023**

Javascript – deel 04

Deze les behandelt voornamelijk formulierelementen en strings in Javascript.

Permanente evaluatie

In de volgende les kan je docent je oplossingen van de opdrachten opvragen en laten meetellen voor je permanente evaluatie.

Verslag

In dit deel van de cursus staan verschillende vragen die je moet beantwoorden en opdrachten om iets te maken of uit te proberen. Het is belangrijk dat je alle opdrachten zorgvuldig uitvoert!

Documenteer je werk in een verslag document 'javascript deel 04' waarin je

- voor elke uitprobeer opdracht een entry maakt met screenshots ter staving van wat je deed
- je antwoorden op de gestelde vragen neerschrijft

Oplossingen van grotere opdrachten (met veel code) bewaar je aparte folders in een Webstorm project.

Lees eerst hoofdstuk 5 uit de HTML cursus volledig (zie document cursus HTML5_deel5_form.pdf).

Bij een invulformulier op een webpagina hoort doorgaans aan de serverkant een stukje programma dat de doorgestuurde data moet verwerken, bv. een Java servlet of een stuk ASP.NET (CORE) code om de gegevens in een databank te bewaren.

Indien een website meerdere invulformulieren heeft, zullen er dus meerdere zulke programma's op de server staan en zal elk aangesproken worden op een andere url. Het 'action' attribuut van het <form> element geeft aan naar welke url de ingevulde gegevens moeten doorgestuurd worden. Indien dit attribuut ontbeekt zullen de gegevens van de form naar dezelfde url gesubmit worden als de huidige pagina waarop het invulformulier staat.

Action url en HTTP method

We zagen reeds eerder dat er meerdere HTTP methods bestaan. We beperken ons hier tot GET en POST.

- GET moet beschouwd worden als een 'lees van resource' opdracht, m.a.w. de meegestuurde data wordt niet op de server toegevoegd. Als je een pagina opvraagt via de adres balk of door op een hyperlink te klikken verstuurt de browser een GET request.
- POST is een 'schrijf naar resource' opdracht, dus als alles goed gaat zal er data op de server bewaard worden.

HTML invulformulieren hebben een 'method' attribuut waarmee je kunt aangeven met welke HTTP method de data moet verstuurd worden naar de server.

- kies GET als de data in de form gewoon meer details toevoegt aan de leesopdracht (bv. details voor een zoek opdracht)
- kies POST als de data in de form moet bewaard worden (bv. klantgegevens registreren)

Het is belangrijk dat je je hieraan houdt, de infrastructuur op het internet gaat ervan uit dat GET requests mogen gecached worden (bv. door een proxy server of in de browser).

Je kan trouwens ook per submit button afzonderlijk instellen hoe de data moet verstuurd worden, o.a. via diens 'formaction' en 'formmethod' attributen.

Vraag : waarom is het potentieel een probleem als je GET gebruikt in een situatie waar het POST had moeten zijn?

Op p. 51 in de HTML cursus staat een limiet van 8192 karakters voor een GET request. Dat cijfer is wellicht gebaseerd op de default instelling van sommige webservers. In de HTTP specificatie werd geen maximum lengte vastgelegd, maar in de praktijk hanteert elke browser (en server) een eigen limiet.

De browser met de kleinste limiet is internet Explorer met net iets meer dan 2000 karakters, dus in de praktijk beperk je een GET opdracht best tot 2000 karakters. Als je meer data verwacht kun je een POST opdracht gebruiken.

Name attribuut

De elementen in een form kunnen een 'name' attribuut hebben. De waarde van dit 'name' attribuut wordt gebruikt bij het doorsturen van de data. Bijvoorbeeld :

```
<form>
   <input type="text" name="postcode">
   ...
</form>
```

Als de gebruiker in dit tekstveld als postcode **8500** invult, zal deze data bij het doorsturen van de gegevens in de HTTP request voorgesteld worden als

postcode=8500

De servercode die de doorgestuurde gegevens moet verwerken, zal ook de 'name' waarde moeten gebruiken om de data in de request terug te vinden (bv. "**postcode**").

Een input element kan natuurlijk ook een id attribuut hebben. Vaak voorziet men zo'n id om iets met het invulveld te kunnen doen in de javascript code (aan de browserkant dus) en dan hebben het 'name' en het 'id' attribuut dezelfde waarde. Bijvoorbeeld

```
<form>
   <input type="text" id="postcode" name="postcode">
   ...
</form>
```

Merk op dat een 'id' uniek moet zijn op een pagina, maar een 'name' niet.

Label

Gebruik een <label> element om de betekenis van een bijbehorend input element te duiden. Als je een <label> gebruikt in combinatie met een <checkbox> of <radiobutton> zul je in deze geen tekst meer stoppen natuurlijk.

Vraag : waarom niet?

Gebruik in een label ook steeds een 'for' attribuut waarin je de name plaatst van het bijbehorende input veld. Dit is een semantische toevoeging, maar het zorgt er ook voor dat als de gebruiker op het label klikt, dit hetzelfde effect heeft als een klik op het bijbehorende element.

- Dit is soms handig als je bv. een checkbox hebt zonder eigen tekst (omdat de tekst al in het label zit), dan moet de gebruiker niet mikken om precies op het kleine aanvinkhokje te klikken.

Hidden input velden

Hidden input velden hebben geen zichtbaar nut voor de gebruiker, maar wel voor de code aan de serverkant. De waarde van zo'n veld wordt immers meegestuurd naar de server samen met de overige gegevens van de form. Je kan hierin dus extra informatie kwijt die voor de gebruiker niet relevant is maar wel nodig is voor de goede werking van je webapplicatie.

Bijvoorbeeld, een medewerker wil op een detaipagina de informatie over een product uit de winkelcatalogus bewerken. De servercode zal een pagina genereren met een form waarin een hidden veld verstopt zit met het de productcode van het product (bv. 12345) :

```
<form>
  <input type="hidden" name="productCode" value="12345">
  ...
</form>
```

Wanneer de form data wordt doorgestuurd naar de server, kan de servercode achterhalen bij welk product de wijzigingen horen door naar de waarde van de productCode parameter te kijken.

Deze techniek is nodig omdat bij elk invoerformulier 2 requests horen (eerst het formulier opvragen, dan de data terugsturen). De server kan deze echter niet zondermeer aan elkaar relateren omdat HTTP een stateless protocol is.

Mocht je een simpelere oplossing in gedachten hebben (iets met cookies, of gewoon ervan uitgaan dat 2 opeenvolgende requests van eenzelfde gebruiker en formulier bij elkaar horen), bedenk dan dat een gebruiker op meerdere tabbladen tegelijk actief kan zijn. Bijvoorbeeld een medewerker die meerdere producten op verschillende tabbladen tegelijk wil bewerken.

Een andere techniek is iets herkenbaars in de submit url te plaatsen, bv. submitten naar een url als

- .../product/ 12345
- ... /product?productCode=12345

Naargelang welk product de gebruiker wil bewerken, zal de server dus een een andere submit url moeten genereren.

Radiobuttons

De browser weet welke radiobuttons samen een groep vormen a.d.h.v. hun 'name' attribuut, dus om een groep te maken geef je elke radiobutton in die groep dezelfde naam. Zorg ervoor dat er altijd een optie geselecteerd is (bv. de eerste, of een extra button met 'geen van bovenstaande').

Submit buttons

Geef submit buttons altijd een 'name' waarde (en elk een verschillende naam natuurlijk), dit maakt het makkelijker om in de servercode te achterhalen op welke button werd geklikt. Sommige forms hebben immers meer dan 1 submit button, bv. eentje om te bewaren en eentje om te annuleren en dan is het wel handig als je op basis van de 'name' parameter in de request kan achterhalen op welke button werd geklikt.

Testen op de 'value' parameter is hiervoor geen goede keuze, die zal immers anders zijn naargelang de gekozen taal ('Bewaar', 'Save', ...) en dat probleem wil je liever niet oplossen in je servercode.

Correctie p.59 : schrap de laatste 3 zinnen van sectie 5.3, dus vanaf 'Let op de volgende attributen [...]' wegens bovenstaande.

Varia

Sommige input elementen (zoals bv. telephone) bieden weliswaar niets extra's bovenop een 'gewoon' tekst invoerveld, toch zijn ze nuttig om tools duidelijk te maken wat de betekenis van het invoerveld is.

Vraag : <select> en <datalist> elementen bevatten beiden <option> elementen. Wat is het verschil tussen beide?

Opdracht invulformulier

Zie document: opdrachtFormulierElementen.pdf

Strings

Een javascript programma kan teksten voorstellen a.d.h.v. *strings*, net als in andere programmeertalen.

Er zijn 2 manieren om een String aan te maken :

```
let s1 = "blablabla";                 // dit is de aanbevolen manier

let s2 = new String("blablabla");     // dit kom je maar zelden tegen
```

Technisch gezien zijn beide mogelijkheden niet 100% gelijk maar voor deze cursus is het verschil niet zo belangrijk.

Er bestaat in javascript trouwens een **typeof** operator die een tekstvoorstelling produceert van wat voor soort waarde iets voorstelt. Bijvoorbeeld

```
let s1 = "blablabla";
typeof s1 geeft "string"

let s2 = new String("blablabla");
typeof s2 geeft "object"
```

Opdracht

Wat voor soort waarden bevatten onderstaande variabelen ?

```
let leeftijd = 34;
let intrest = 0.12;
let isGevaarlijk=true;
let vandaag = new Date();
const print = (message) => {
  console.log(message);
}
```

Op onderstaande link kun je zien wat voor soort waarden er in javascript bestaan, het zijn er minder dan je wellicht zou verwachten :

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/typeof

De lengte van een string kun je opvragen met de .length property, bv.

```
let tekst="Hello world";
console.log( tekst.length );                    // toont 11
```

Individuele karakters uit een string kun je opvragen met de .charAt() method die een index als parameter vereist.

```
console.log( tekst.charAt(1) );                    // toont "e"
```

Ook in javascript nummeren we de posities in een string beginnend vanaf 0. Het tweede karakter vind je dus op indexpositie 1.

Let op, soms kom je wel eens code tegen die hiervoor de [] operator gebruikt ipv de .charAt() method, bv. tekst[1] , maar dit werkt niet op alle browser(versies). Bovendien wekt het de illusie dat een string een array van karakters, maar dit is NIET het geval! De analogie gaat bv. niet op voor toekenningen :

```
bv. tekst[1]="a" zal NIET werken
```

Gebruik dus liever de .charAt() method in je eigen code.

Opdracht : spaties op console

Maak een webpagina met een inputveld en een button. Als er op de button geklikt wordt, plaatst je javascript code de tekst uit het inputveld op de console waarbij de inidivuele karakters door spaties gescheiden zijn. Bijvoorbeeld, bij input

```
Dit is een tekst.
```

Verschijnt er op de console

```
D i t i s e e n t e k s t .
```

Om teksten aaneen te plakken kun je ofwel de .concat() method gebruiken of wel de '+' operator, bv.

```
let s1 = "Hello ";
let s2 = "world!";
let s3=s1.concat(s2);          // s3 bevat nu "Hello world!"
let s4=s1+s2;                  // s4 bevat nu "Hello world!"
```

Doorgaans gebruikt men de '+' operator.

Opdracht : spaties met functie

Schrijf een functie 'maakMetSpaties' die voor een gegeven String een nieuwe String opbouwt waarin de karakters en spaties vervat zitten zoals beschreven in de vorige opdracht.

Bijvoorbeeld

```
const maakMetSpaties = (inputText) => {
  let result="";
  // hier wordt result opgevuld op basis van de parameter inputText
  return result;
}
```

Test je functie door de oplossing van de 'spaties op console' oplossing te herwerken zodat deze de maakMetSpaties functie gebruikt (en de return value op de console zet).

Je kunt een zoektekst opzoeken in een string met de volgende twee methods :

- .indexOf(zoektekst) // begint zoektocht vooraan de tekst
- .lastIndexOf (zoektekst) // begint zoektocht achteraan de tekst

Beide methods geven als resultaat de positie waarop de deeltekst gevonden werd, of -1 indien deze niet gevonden werd.

Bijvoorbeeld, als je je afvraagt of een bepaalde tekst het woordje 'hond' bevat :

```
let tekst = "Man bijt hond met valse tanden";
if ( tekst.indexOf("hond") != -1 ) {
  // tekst bevat het woordje "hond"
} else {
  // tekst bevat het woordje "hond" niet
}
```

De return value van indexOf (en lastIndexOf) geeft dus de karakter positie waarop het woord werd gevonden, in bovenstaand voorbeeld is dit index positie 9. Die exacte positie is bv. handig als je woorden in een tekst wil vervangen (zie verderop).

Je kan de methods ook oproepen met een tweede parameter idx :

- .indexOf(zoektekst, **idx**) // begint zoektocht vooraan de tekst
- .lastIndexOf (zoektekst, **idx**) // begint zoektocht achteraan de tekst

De tweede parameter idx is dus optioneel en kan gebruikt worden om vanop een andere startpositie te beginnen zoeken. Dit is bv. handig als je in een tekst verder wil blijven zoeken nadat je al iets gevonden hebt. De nieuwe startpositie is dan doorgaans het karakter na de treffer van de eerdere zoekactie.

Opdracht : Man van An

Schrijf een kort Javascript programma dat telt hoe vaak de sequentie "an" in de tekst "De man van An geeft geen hand aan ambetante verwanten". Probeer dit eerst met indexOf en daarna ook eens met lastIndexOf op te lossen.

-