Internet en het World Wide Web

Leereenheid 1

Internet en het World Wide Web

INTRODUCTIE

De titel van deze cursus, Webapplicaties: de clientkant, geeft het onderwerp en de afbakening van de cursus aan. In deze eerste leereenheid preciseren we die termen. Wat is het web, wat is een webapplicatie en wat is de plaats van de clientkant van webapplicaties binnen een webapplicatie als geheel? Wat betekenen de begrippen client en server? Wat valt er te zeggen over de verdeling van functionaliteit over client en server?

We laten u de architectuur van een webapplicatie zien, met de rol van de clientkant en de serverkant. Daarbij laten we u zien hoe de communicatie verloopt bij webapplicaties, ook in het geval van een Ajax-applicatie.

U krijgt hiermee een idee van wat u in de rest van deze cursus zult leren en hoe dit past in het geheel van het bouwen van een webapplicatie. Dat maakt het mogelijk om de komende leereenheden in een groter geheel te plaatsen.

LEERDOELEN

Na het bestuderen van deze leereenheid, wordt verwacht dat u:

- kunt aangeven wat er bedoeld wordt met de termen webpagina, website, webapplicatie en web app
- kunt beschrijven wat er gebeurt wanneer u een HTML-bestand van het filesysteem in uw browser opent, wanneer u een URL invoert in de adresbalk, wanneer u op een link klikt en wanneer u een webformulier instuurt
- kunt beschrijven wat er in de browser gebeurt op het moment dat de HTML binnenkomt
- kunt beschrijven wat de DOM-boom is
- kunt aangeven wat bepaalt of een bestand behoort tot de clientkant van webapplicaties of tot de serverkant
- kunt beschrijven wat een thin client is en wat een fat client is
- kunt aangeven wat DOM-manipulatie is
- kunt beschrijven wat er gebeurt als een applicatie gebruikmaakt van Ajax.

LEERKERN

1 Webapplicaties en web apps

De termen webpagina, website, webapplicatie en web app hebben geen echt formele definities, maar het is handig om voor deze cursus te bepalen wat we onder die termen verstaan.

Webpagina
HyperText Markup Language
Hypertext

Een *webpagina* is een document dat via een webbrowser bekeken kan worden en opgesteld is in HTML (*HyperText Markup Language*), dat uitgebreid aan bod zal komen in leereenheid 2. Die term *Hypertext* houdt in: een document met links naar andere documenten (hyperlinks).

World Wide Web

Het WWW (*World Wide Web*) is oorspronkelijk begonnen als een verzameling gelinkte webpagina's. Het woord 'web' in 'world wide web' slaat dan ook op het feit dat documenten op webservers via hyperlinks verbonden zijn met documenten op dezelfde server of op andere webservers. Dat is mogelijk wanneer die documenten in HTML zijn geschreven.

Weblink: De eerste webpagina

De allereerste webpagina, uit 1990, is niet meer te zien, maar er bestaat nog wel een latere kopie. Het idee was van Tim Berners-Lee: webservers met unieke adressen, die documenten beheerden met hyperlinks die refereerden aan andere documenten op zo'n server (dezelfde of een andere server). Het oorspronkelijke idee van Tim Berners-Lee was dus een WWW van aan elkaar gelinkte webpagina's, die hij destijds simpelweg 'documents' noemde.

Website

Een *website* is een samenhangend geheel van webpagina's, behorend bij één domein (voorbeeld: `ou.nl`) of bij een subdomein (voorbeeld: `studienet.ou.nl`). In sommige gevallen kunnen er ook meerdere websites binnen één subdomein bestaan: een formele definitie van een website geven is lastig. Zo'n samenhangend geheel van webpagina's heeft over het algemeen een gezamenlijke lay-out, en vaak is er een vorm van navigatie op elke pagina die het mogelijk maakt om de rest van de website te bekijken.

OPGAVE 1.1

Twee voorbeelden van websites behorende bij subdomeinen van eenzelfde domein zijn `www.ou.nl` en `studienet.ou.nl`. Hoe zou u een Studienet-site bij een cursus noemen? Is dat een website of zijn het een aantal pagina's binnen een website?

Webapplicatie

Een *webapplicatie* bestaat uit een webpagina of een website met logica: elke applicatie bevat logica. Een webapplicatie is daarbij over het algemeen interactief: de applicatie reageert op bewegingen van de muis of op andere input van de gebruiker. Een bekend voorbeeld van een webapplicatie is een OV-reisplanner. Het is duidelijk dat er berekeningen aan te pas moeten komen om de gebruiker op de door hem of haar gewenste tijd een route voor te stellen van het gewenste beginpunt naar het gewenste eindpunt.

De bouwstenen voor een webpagina zijn tegelijkertijd de bouwstenen voor de user interface van een webapplicatie. Die bouwstenen (HTML en CSS) komen daarom in deze cursus aan bod in leereenheid 2 en 3, maar we zullen ze slechts vrij vluchtig behandelen. Het hoofdonderwerp wordt gevormd door het programmeren van webapplicaties.

Web app

Een *web app* is een webapplicatie (de term is in feite een afkorting voor webapplicatie) die speciaal is ingericht voor mobiele platforms zoals tablets of smartphones.

In deze cursus zijn webapplicaties en web apps het hoofdonderwerp. Alle technieken voor webapplicaties zijn ook van toepassing op web apps.

2 Hoe werkt het web

2.1 DUBBELKLIKKEN OP EEN HTML-BESTAND

De eenvoudigste manier om een webpagina te zien te krijgen, is wanneer u een HTML-bestand op uw filesysteem heeft staan. U kunt dan dubbelklikken op zo'n bestand, en als de extensie `.html` of `.htm` op uw systeem is gebonden aan een browser, zal de browser de pagina laten zien. U kunt hetzelfde bereiken door in de browser te kiezen voor `Bestand - Bestand openen`, of, wanneer u geen zichtbare menubar heeft, met `Nieuwe tab - Bestand openen`, of met de toetscombinatie (Ctrl-O).

FIGUUR 1.1 Een webpagina oproepen via het filesysteem

In figuur 1.1 kunt u zien dat de URL in de adresbalk er anders uitziet dan u gewend bent. Er staat `file:///` gevolgd door het pad dat u gevolgd hebt voor het bestand. Wat hier gebeurt, is dat de browser het HTML-bestand vanaf het filesysteem inleest, interpreteert, en het dan weergeeft. In de HTML kunnen extra bestanden zijn gespecificeerd, zoals CSS-bestanden, JavaScript-bestanden of bijvoorbeeld afbeeldingen. De browser leest ook die bestanden in.

OPGAVE 1.2
Pak de bouwsteen die bij deze leereenheid behoort uit op een door u gewenste plek op uw filesysteem. Dubbelklik op het HTML-bestand, en kijk of u de pagina te zien krijgt. Doe daarna hetzelfde via het menu van Firefox. Bekijk het HTML-bestand ook in een editor (zie de aanwijzingen op Studienet voor installatie en gebruik van Notepad++). Welk extra bestand leest de browser hier in?

2.2 URL IN DE ADRESBALK

GET-Request
HTTP-protocol
HyperText Transfer Protocol

Wanneer u een URL in de adresbalk van de browser typt, gebeurt er iets soortgelijks. In dit geval leest de browser het bestand niet van het filesysteem, maar vraagt de browser de HTML op via een *GET-Request* volgens het *HTTP-protocol*, het *HyperText Transfer Protocol*. De naam van dat protocol geeft aan dat het bedoeld is om Hypertext-documenten te kunnen versturen van het ene programma (bijvoorbeeld uw web-browser) naar het andere programma (een webserver). Die programma's kunnen bovendien op verschillende computers staan, zolang ze beide maar verbonden zijn aan internet.

Een webserver is dus een programma. U kunt een webserver draaien op uw eigen computer (dat zult u tijdens deze cursus ook gaan doen), maar over het algemeen draait een webserver op een computer die daar speciaal voor is ingericht.

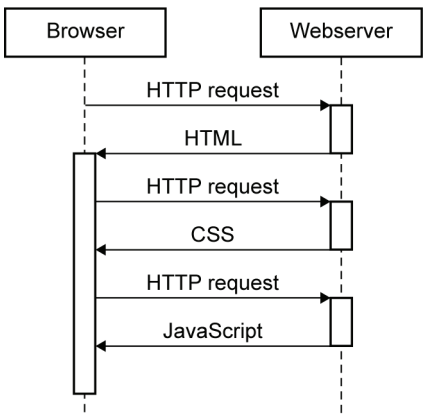
FIGUUR 1.2 Communicatie met een URL in de adresbalk

Figuur 1.2 laat zien wat er gebeurt: de webbrowser stuurt een HTTP-GET-request op naar de webserver. De webserver stuurt een lange string met de HTML terug. De webbrowser en de webserver zijn programma's die draaien op de hardware die in de figuur te zien is, bovenop het op die hardware draaiende besturingssysteem.

Bij elke referentie naar een extra bestand stelt de webbrowser een verzoek op voor dat bestand, en ook die bestanden worden door de webserver naar de browser gestuurd. De browser interpreteert alles en laat het zien.

De webbrowser en de webserver kunnen ver van elkaar staan: het wolkje in figuur 1.2 kan voor het internet staan, maar ook voor een lokaal netwerk. Het is op deze manier ook mogelijk om met de browser een pagina te bekijken, via HTTP, die wordt aangeleverd door een webserver die op dezelfde computer draait. U zult binnen deze cursus uw computer zelf zo inrichten dat dat mogelijk is.

De webserver kan na het ontvangen van een HTTP-request (u zult in deze cursus nog leren wat een GET-request is en wat voor requests er nog meer zijn; voorlopig noemen we het simpelweg request) op een van de volgende manieren te werk gaan:
– De server haalt een HTML-document op van het door de server beheerde gedeelte van het filesysteem en stuurt de inhoud daarvan naar de browser.
– De server activeert een programma. Zo'n programma kan een script zijn dat wordt geïnterpreteerd (bijvoorbeeld een script in PHP) of een gecompileerd programma (bijvoorbeeld een servlet). Dat programma levert als output een string HTML op die naar de browser wordt gestuurd.



FIGUUR 1.3 Communicatie tussen browser en webserver in de tijd

U ziet hier een sequence diagram; u hoeft niet te kunnen uitleggen hoe zo'n diagram gelezen moet worden.

Figuur 1.3 laat zien hoe dit proces in de tijd verloopt. Vanuit de browser en de webserver ziet u een lijn lopen, die een tijdlijn voorstelt; de tijd verstrijkt naar beneden toe. De browser stuurt een HTTP-request aan de webserver. Die heeft even tijd nodig (om het bestand van het filesysteem op te halen, misschien een script te interpreteren, en misschien een programma te starten en te laten draaien). Het feit dat de webserver 'bezig' is wordt aangegeven door het rechthoekje op de tijdlijn. Ten slotte kan er HTML naar de browser worden gestuurd. Dan start de browser een proces: de browser gaat de HTML die binnenkomt interpreteren. Zodra er een verwijzing binnenkomt naar een ander bestand, haalt de browser dat middels een HTTP-request op.

Aan de webserverkant verloopt het proces dan identiek als in het eerste geval. We hebben hier als voorbeeld een CSS-bestand en een JavaScript-bestand genomen, waarnaar wordt verwezen in de HTML, maar het kan van alles zijn; het kunnen ook meerdere CSS-bestanden of JavaScript-bestanden zijn.

2.3 LINK KLIKKEN

HTML heeft z'n naam te danken aan de links naar andere bestanden die het kan bevatten. Een link aanklikken is dan ook de meest voorkomende manier waarop gebruikers een webpagina te zien krijgen.

De gang van zaken bij het aanklikken van een link is precies dezelfde als in het geval er een URL in de adresbalk van de browser wordt getypt. Wanneer de gebruiker op een link klikt, stelt de browser een HTTP-request op en stuurt dat naar de webserver die bij het betreffende domein hoort. Dan treedt het proces op zoals dat in figuur 1.3 staat gemodelleerd.

In de meeste gevallen verloopt dat via tussenliggende domain name servers, zodat de browser niet de directe adressen hoeft te kennen: een domain name server kent van een groot aantal URL's de exacte adressen, en kent andere domain name servers voor gevallen waarin de URL onbekend is. Zo komt er uiteindelijk een verbinding tot stand zonder dat uw eigen PC alle internetadressen hoeft te onthouden.

De browser bouwt dan een geheel nieuwe pagina op en vervangt de oude of vult het lege window, wanneer de link in een nieuwe tab of een nieuw window wordt geopend.

2.4 EEN FORMULIER INSTUREN

Ook wanneer u een webformulier invult en opstuurt, is daar een link aan verbonden.

FIGUUR 1.4 Een webformulier

In figuur 1.4 bijvoorbeeld, hoort bij de knop 'Geef reis en prijs' een link. In dit geval stuurt de browser in het HTTP-request samen met de link ook de ingevulde gegevens mee. Aan de serverkant zal er nu altijd een proces worden gestart (waarin een gecompileerd programma wordt uitgevoerd of waarin een script worden geïnterpreteerd), waarbij de binnenkomende gegevens worden verwerkt. In dit geval zijn de resultaten van die verwerking te zien in de pagina die wordt teruggestuurd: daarin krijgt u het berekende reisadvies te zien.

3 HTML en de DOM

De browser interpreteert de binnenkomende HTML en laat dan de pagina zien, hebben we eerder aangegeven. Wanneer u zelf webapplicaties gaat bouwen, is het van belang iets te weten over de manier waarop de browser dat doet.

3.1 DE INPUT VOOR DE BROWSER

Het uitgangspunt voor de browser is de HTML, die in de vorm van een lange string binnenkomt. De browser leest die string karakter voor karakter, en de eerste taak is om te herkennen wat er staat.

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Een webpagina</title>
5      <link rel="stylesheet" href="stijl.css">
6      <script src="gedrag.js"></script>
7    </head>
8    <body>
9      <div id="page">
10       <h1>Een webpagina</h1>
11       <img height="450" width="600"
              alt="Een browserwindow met een webpagina"
12            src="webpage.png">
13     <p>De browser krijgt een lange string binnen, en bouwt daar
          deze pagina van op.</p>
14   </div>
15   </body>
16 </html>
```

Wanneer de browser de bovenstaande code binnenkrijgt, leest de browser eerst een `<`, dan een `!`, dan een `D`, enzovoort. Na het eerstvolgende teken `>` weet de browser dat het `doctype` van de pagina is gelezen (dat in dit geval aangeeft dat wat volgt als HTML5 geïnterpreteerd kan worden).

Tag
Parsen

Het is voor te stellen dat de browser op deze manier steeds een stukje verder leest, en ondertussen *tag*s herkent (zo'n proces heet *parsen*). Waar de browser een `link`-tag met een `href`-attribuut tegenkomt, of een `script`-tag met een `src`-attribuut, of een `img`-tag met een `src`-attibuut, stelt de browser een HTTP-request op voor het betreffende bestand, stuurt het request naar de webserver, en gaat verder met interpreteren.

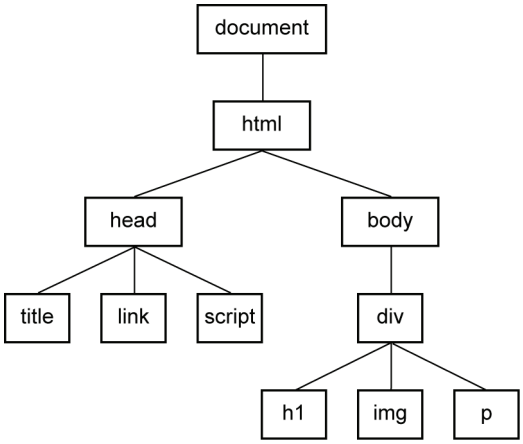
3.2 DOM: BOOM VAN ELEMENTEN

DOM
Document Object Model

De browser bouwt al interpreterend een boom op in het geheugen. Die boom is de *DOM* (*Document Object Model*).

Node

De elementen van de DOM heten *node*s. Een node kan worden gezien als een object: een node bevat attributen. De attributen bij een tag in het HTML-bestand komen terug als attributen van de betreffende node in de DOM. Elk HTML-element wordt tijdens het parsen vertaald naar een node in de DOM.



FIGUUR 1.5 Bij de HTML behorende boom

Wanneer de browser HTML binnenhaalt volgens een van de manieren die beschreven staan in paragraaf 2, vindt het proces van parsen plaats, wordt er een geheel nieuwe DOM-boom opgebouwd, en vertaalt de browser de elementen uit die boom naar een representatie (met alle vormgeving die daarbij hoort) in het window. Er wordt, met andere woorden, een nieuwe DOM opgebouwd, en die DOM moet gerenderd worden als webpagina. In de meeste gevallen bestaat de tijd die u wacht wanneer u naar een nieuwe pagina in de browser gaat, voornamelijk uit de tijd die het kost om de pagina in de browser op te bouwen.

3.3 SCHEIDING VAN VERANTWOORDELIJKHEDEN

Door deze cursus heen zult u zien dat we de nadruk leggen op het principe van het scheiden van verantwoordelijkheden.

Software design principe: Scheiden van verantwoordelijkheden
Het principe van het scheiden van verantwoordelijkheden houdt in dat er bij elke oplossing van een probleem verschillende verantwoordelijkheden te onderscheiden zijn, die apart moeten worden aangepakt.

JavaScript

CSS

In het geval van webapplicaties zijn die verantwoordelijkheden: structuur en betekenis, uiterlijk en lay-out, en gedrag en logica. Gedrag en logica vallen onder de verantwoordelijkheid van *JavaScript*. Een groot aantal leereenheden, te beginnen met leereenheid 4, is aan JavaScript gewijd. Uiterlijk en lay-out vallen onder de verantwoordelijkheid van *CSS*. Leereenheid 3 is gewijd aan CSS. Structuur en betekenis vallen onder de verantwoordelijkheid van HTML. Leereenheid 2 is gewijd aan HTML.

U zult in deze cursus zien hoe u ervoor kunt zorgen dat die verantwoordelijkheden niet door elkaar gaan lopen.

4 Clients en servers

Server

Webserver

Request

Een *server* is een programma dat een dienst kan verlenen. Een server doet dat over het algemeen op verzoek (dus niet op eigen initiatief), en één server kan veel clients bedienen. Een *webserver* is een programma dat een deel van het filesysteem beheert (waaronder over het algemeen bestanden in HTML), en die bestanden bij een verzoek (een *request* in HTTP-termen) kan versturen. De webserver is in staat het HTTP-protocol te hanteren. Bekende webservers zijn bijvoorbeeld Apache en Tomcat.

Client

Een *client* is een programma dat van een dienst gebruik kan maken. Het is de client die een request stuurt naar een server: de client hanteert dus evenals een webserver het HTTP-protocol. Het is de client die het initiatief neemt. Clients die gebruikmaken van webservers zijn onder meer zoekrobots en webbrowsers. Bekende webbrowsers zijn Mozilla Firefox, Opera, Safari, Internet Explorer of Chrome.

Binnen deze cursus zullen we ervan uitgaan dat iedereen Mozilla Firefox gebruikt, om niet geconfronteerd te worden met verschillen tussen browsers (die verschillen worden gelukkig steeds kleiner); wanneer u in de praktijk webapplicaties gaat ontwikkelen, zult u wel op die verschillen moeten letten (we zullen dat niet behandelen in deze cursus).

4.1 CLIENT-SIDE- EN SERVER-SIDE-LOGICA

U heeft in paragraaf 2.2 al gezien dat er logica aan de serverkant gebruikt kan worden om HTML te produceren wanneer er een request komt (in plaats van dat er een bestand van het filesysteem wordt opgehaald). In paragraaf 2.4 heeft u gezien dat de webserver de ingevulde gegevens van een formulier binnen kan krijgen, en die dan kan verwerken (bijvoorbeeld met behulp van een database): ook daar is server-side-logica nodig.

Zonder client-side-logica zijn er twee vormen van interactie in een webapplicatie mogelijk:
– De gebruiker klikt op een link, en krijgt een nieuwe pagina te zien.
– De gebruiker vult een formulier in en stuurt dat op. Er verschijnt een nieuwe pagina, waarbij het mogelijk is dat de ingevulde gegevens gebruikt zijn om die pagina samen te stellen.

Client-side-logica breidt die mogelijkheden enorm uit. Wanneer u een webapplicatie gebruikt die reageert alsof het een desktop-applicatie is, hoeft u niet steeds te wachten tot er een nieuwe pagina is geladen, en weet u dat er client-side-logica in het spel is. Een goed voorbeeld vormen de Office-applicaties van Google docs: die zouden onwerkbaar traag zijn wanneer er alleen sprake zou zijn van server-side-logica.

4.2 ARCHITECTUUR: THIN EN FAT CLIENTS

Thin clients
Fat clients

In de loop der tijd is er een aantal keer een beweging geweest van clients die geen of nauwelijks logica kenden (*thin clients*) naar clients die veel logica gebruikten (*fat clients*) en andersom.

In vroeger tijden werden berekeningen gemaakt op mainframes, die mensen konden gebruiken vanaf een console: een thin client. Die consoles kregen steeds meer logica (zodat het bijvoorbeeld mogelijk werd om terug te gaan in de geschiedenis van gegeven opdrachten): het werden fat clients. Ten slotte ontstonden er computers die zelfstandig konden opereren, doordat ze alle benodigde logica en applicaties aan boord hadden: personal computers.

Dat had wel een nadeel: waar een update van een applicatie op een mainframe met veel thin clients erg gemakkelijk is (het is een kwestie van de update toepassen op het mainframe), is het veel lastiger om een update te verspreiden over alle personal computers die een bepaalde applicatie draaien.

Met de komst van het web ontstonden er opnieuw thin clients: van logica aan de clientkant werd nauwelijks gebruikgemaakt, zodat webapplicaties alleen server-side-logica hadden. De browser fungeerde als thin client (thin wat de webapplicatie betreft: de browser gebruikte natuurlijk wel externe logica om pagina's binnen te halen en op te bouwen).

Naarmate meer client-side-logica werd ingezet, veranderden webapplicaties van thin client naar fat client. Er is wel een belangrijk verschil met de vorige generatie fat clients: de client-side-logica van webapplicaties wordt bewaard op de server. Elke keer dat u een webapplicatie draait in uw browser, wordt de applicatie opgehaald van de server. *Client-side-logica* betekent dus niet dat de logica zich aan de client-side *bevindt*, maar dat de logica aan de client-side *draait*.

Client-side-logica

De situaties wat updates betreft ligt daardoor veel gunstiger in het geval van webapplicaties dan in het geval van personal computers. Personal computers hebben de applicaties die draaien op de harde schijf staan. Bij webapplicaties haalt de browser elke keer dat een webapplicatie wordt gedraaid de gehele webapplicatie binnen: alle bestanden, zowel voor de server-side- als voor de client-side-logica, bevinden zich op de webserver. Een update van een applicatie hoeft dus alleen maar te worden aangebracht op de server.

4.3 WEBAPPS EN ARCHITECTUUR

De komst van mobiele apparaten is een extra stimulans geweest voor het ontwikkelen van webapplicaties met fat clients. Een van de kenmerken van mobiele apparaten is dat de internetverbinding niet altijd aanwezig is. Web apps, webapplicaties voor mobiele apparaten dus, worden daarom vaak zo gemaakt dat ze kunnen blijven fungeren als er geen verbinding is. Alle logica wordt dan door de clientkant uitgevoerd. Wanneer er weer verbinding is, zal de applicatie een en ander synchroniseren met de webserver.

5 Ajax en dynamische pagina's

We hebben hierboven aangegeven dat client-side-logica een steeds grotere rol is gaan spelen, en u hebt ook al gezien dat gedrag en logica onder de verantwoordelijkheid van JavaScript vallen, maar hoe JavaScript een webapplicatie dynamisch kan maken heeft u nog niet gezien. Een groot gedeelte van de cursus is daaraan gewijd, maar hier geven we alvast een blik op de mechanismen.

5.1 MANIPULATIE VAN DE DOM

In sectie 3.2 heeft u gezien dat de browser vanuit de binnenkomende HTML een DOM-boom opbouwt in het geheugen, die de pagina representeert. De browser gebruikt die DOM-boom om de pagina te kunnen afbeelden, maar stelt de boom ook ter beschikking aan JavaScript binnen die pagina. De DOM-boom wordt ter beschikking gesteld in de vorm van objecten, die we in deze cursus *DOM-elementen* zullen noemen.

DOM-elementen

Een script dat wordt meegestuurd met de pagina kan de waarde van attributen bekijken van die DOM-elementen, en kan ook waarden veranderen. Zo is het mogelijk om met behulp van JavaScript de achtergrondkleur van een element in de DOM-boom te veranderen. De browser reageert direct op zo'n verandering, en laat de nieuwe achtergrondkleur zien. Manipulatie van de DOM leidt dus tot veranderingen in de afgebeelde pagina. Daarbij verandert er niets aan de HTML. DOM-manipulatie zult u tegenkomen in leereenheid 8.

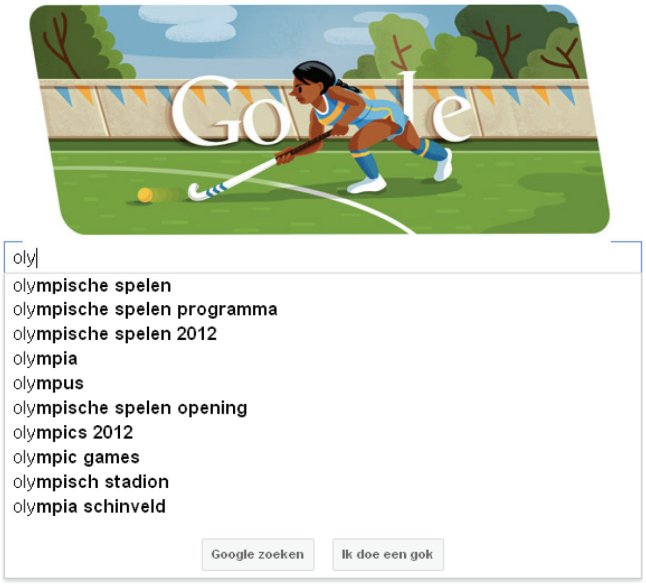
Events

Het is binnen JavaScript mogelijk om te reageren op *events*, zoals muisbewegingen, toetsaanslagen of muisklikken. Als in een script als reactie op een event iets wordt veranderd aan de DOM-boom, is de pagina daarmee interactief geworden. DOM-manipulatie is dus een vorm van client-side-logica waarmee een webpagina veranderd wordt in een webapplicatie. Events zult u tegenkomen in leereenheid 9.

5.2 AJAX: GEGEVENS VERSTUREN EN OPHALEN

Ajax

Met *Ajax* (de naam is een acroniem voor Asynchronous JavaScript and XML, maar Ajax hoeft niet asynchroon te zijn en er hoeft geen gebruik gemaakt te worden van XML) is het mogelijk om een stap verder te gaan. Het is namelijk mogelijk om gegevens te versturen naar de webserver en gegevens op te halen, vanuit JavaScript, zonder dat dat leidt tot het laden van een nieuwe pagina. In plaats daarvan gebruikt het script de opgehaalde gegevens om de DOM-boom te manipuleren: op die manier worden de opgehaalde gegevens verwerkt in de huidige pagina.



FIGUUR 1.6 Aanvulling van zoektermen

Figuur 1.6 laat een situatie zien waarin het duidelijk is dat er van Ajax gebruik wordt gemaakt: terwijl de gebruiker bezig is een zoekterm te typen, toont de browser zoektermen die met de al ingevoerde letters beginnen. Het is duidelijk dat die informatie van de server moet komen: er bestaan simpelweg te veel mogelijkheden om alle zoektermen met de pagina mee te sturen voor hij wordt geladen. Het is ook niet zo dat bij elke ingevoerde letter een nieuwe pagina wordt geladen.

OPDRACHT 1.3

Worden er in bovenstaand voorbeeld alleen gegevens opgehaald of ook verstuurd? En wordt de DOM gemanipuleerd?

5.3 SYNCHROON EN ASYNCHROON

Asynchroon

De eerste A van Ajax staat voor *asynchroon*. Een asynchroon request versturen vanuit een script betekent dat de pagina niet 'bevriest' tot het antwoord van de server is teruggekomen. Er kan in de tussentijd dus nog steeds gereageerd worden op events.

Synchroon

Het is ook mogelijk om requests vanuit een script *synchroon* te versturen. In dat geval is er geen verdere interactie op de pagina mogelijk tot het antwoord van de server er is.

ZELFTOETS

1 Wat gebeurt er precies aan de serverkant nadat u in een webpagina op een link heeft geklikt?

2 Wat is de essentie van Ajax?

3 Alle bestanden van een webapplicatie bevinden zich op de webserver. Waarom wordt er dan toch van server-side- *en* client-side-logica gesproken?

TERUGKOPPELING

1 Uitwerking van de opgaven

1.1 Dit voorbeeld laat zien hoe lastig het is te definiëren wat een website is. Studienet als geheel lijkt duidelijk te vallen onder het begrip website, naast bijvoorbeeld de website www.ou.nl. Maar vanuit het oogpunt van iemand die een cursus bestudeert of een examinator van een cursus, vormen de pagina's die bij één cursus horen één samenhangend geheel: een website. Zo'n cursuswebsite heeft een eigen navigatie en kan een eigen vormgeving hebben. Tegelijkertijd is de site ingebed in het grotere geheel.

1.2 Niet alle browsers laten `file:///` zien voor het pad met het bestand. Het extra bestand dat wordt ingelezen is in dit geval `stijl.css`.

1.3 De al ingevoerde letters moeten verstuurd worden als de server bekende zoektermen moet kunnen teruggeven die met die letters beginnen. Er worden dus ook gegevens verstuurd. Er vindt DOM-manipulatie plaats: anders zouden de zoektermen niet zichtbaar worden.

2 Uitwerking van de zelftoets

1 Nadat het request dat de browser opstelt is aangekomen bij de webserver die verantwoordelijk is voor de betreffende URL, zal de webserver de gevraagde gegevens terugsturen. Het kan zijn dat de webserver die gegevens van het filesysteem in kan lezen; het kan zijn dat de webserver een script interpreteert en het resultaat daarvan opstuurt; het kan ook zijn dat de webserver een programma start en het resultaat daarvan opstuurt.

2 De essentie van Ajax is dat het mogelijk is om vanuit JavaScript gegevens op te halen van de webserver zonder dat dat leidt tot een nieuwe pagina. In plaats daarvan worden de gegevens in de huidige pagina verwerkt.

3 De vraag of logica server-side of client-side is, hangt niet af van de vraag waar de bestanden zich bevinden, maar van de vraag waar de logica *draait*.