Javascript – deel 11 - storage

Deze les bespreekt de verschillende opslagmogelijkheden die een Javascript programma in een browseromgeving kan gebruiken.

Het enige dat je uit dit deel echt moet kennen

voor de eindtest is **het gebruik van local storage!**

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 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 apart in een .zip file.

Globale variabelen

We zagen eerder al dat globale variabelen kunnen gebruikt worden om gegevens te onthouden over meerdere functie-oproepen heen.

> *(terzijde : het is niet omdat twee functies allebei een lokale varibele 'x' hebben, dat je er een globale variabele van moet maken!)*

Zoals in elke programmeertaal is het beter om het gebruik van globale variabelen zo veel mogelijk te beperken. De voornaamste redenen hiervoor zijn

- het risico op een naam conflict : twee onafhankelijke stukken code die samen in een applicatie terechtkomen (bv. twee libraries), kunnen onbedoeld elkaars waarden overschrijven als ze toevallig dezelfde naam kozen voor hun 'eigen' globale variabele.
- de toegang tot globale variabelen is ongecontroleerd : indien er een foutieve waarde in een globale variabele terechtkomt, is het moeilijk om de code te vinden die deze waarde erin stopte.

Hoe complexer en omvangrijker het programma, hoe relevanter deze redenen worden.

In Javascript komen alle globale variabelen samen in één grote globale *scope* terecht (risico op naamsconflicten) en de toegang ertoe is ongecontroleerd.

Voor globale 'var' variabelen (niet 'let' of 'const') is er nog een bijkomend probleem, deze worden nl. properties van het globale 'window' object. Hierdoor is het risico op een naam conflict nog groter

- het window object zit sowieso al propvol met voorgedefinieerde properties
- de namen van 'losse' functies leiden eveneens tot properties van het windows object
- sommige zaken uit de DOM-tree leiden tot properties van het window object
    - nml. alle waarden van id-attributen en sommige name-attributen http://w3c.github.io/html/browsers.html#named-access-on-the-window-object
        - (maak hier trouwens nooit gebruik van!!)

Om de vervuiling van deze globale scope tegen te gaan, kan men slim gebruik maken van *closures* en *Immediately Invoked Function Expressions* (IIFE) om variabelen af te schermen. Deze geavanceerde techniek valt echter buiten het bestek van deze cursus.

Een eenvoudige manier om de risico's van globale variabelen enigszins te beperken, is ze samen in één enkel object te bundelen en slechts 1 globale variabele te definiëren.

Bijvoorbeeld :

```
const global = {
  score : 0,
  playerNames : [],
  PATH_PREFIX : "images\sprite",
  PATH_SUFFIX : ".png"
}
```

Dit heeft als bijkomend voordeel dat je ze makkelijk kan terugvinden bij het debuggen, open gewoon een Watch voor deze ene variabele en je kan via diens properties al je globale variabelen samen zien.

Belangrijk : een globale variabele hoort bij de uitvoering van een Javascript programma, binnen de context van het huidige HTML document. Telkens de browser naar een andere pagina navigeert (of dezelfde pagina herlaadt!), wordt deze context (inclusief de waarden van globale variabelen) weggegooid. De browser maakt bij het inladen van de andere pagina een nieuwe context voor de globale variabelen van die andere pagina.

Opdracht

Unzip het bestand 'demo storage.zip' en open index.htm uit folder 'demo 1 globale variabelen' in je browser.

Druk een paar keer op de knop.

- Blijft de data behouden als je de pagina herlaadt?

- Blijft de data behouden als je het tabblad sluit en de pagina opnieuw opent?

- Gebruiken twee tabbladen die dezelfde pagina tonen, dezelfde data?

Gegevens onthouden over meerdere pagina's heen

Globale variabelen gaan dus verloren als we naar een andere pagina navigeren, maar wat als we nu gegevens langer willen bijhouden dan de context van 1 pagina?

Wel, als onze applicatie een server component heeft, zouden we de gegevens op de server kunnen bewaren. Bv. de inhoud van de winkelwagen op de site van een webwinkel kan op de server bijgehouden worden. Soms is er echter geen server component, of willen we om praktische redenen de gegevens aan de browserkant bijhouden (bv. om offline te kunnen werken)

Aan de browser kant zijn er vier manieren om gegevens bij te houden over meerdere pagina's heen :

- **cookies**, die met elke request naar een server in een domein worden meegestuurd
- web storage api :
    - **session storage**
    - **local storage**
- **web sql database**

Welke je kiest hangt af van wat je met de bewaarde data wil doen en hoe lang je ze wil bewaren.

Je kan trouwens voor elk van deze vier mogelijkheden, de opgeslagen data bekijken die relevant is voor een pagina. Open de pagina en kijk in de Chrome developer tools op het 'Application' tabblad.

Cookies

Een cookie is een stukje tekst dat door de browser bewaard wordt, op vraag van de server of javascript code. Het cookie is gekoppeld aan een bepaald domein (en eventueel een bijbehorend path).

De inhoud van een cookie bestaat uit key/value pairs en wordt met ELKE http request naar een server op dit domein meegestuurd (mogelijk zelfs door elke webapplicatie op dit domein). De maximale omvang (qua data hoeveelheid) en efficientie van cookies is eerder beperkt.

De cookies samen kleiner zijn dan 4096 bytes per domein en worden gedeeld door alle tabbladen die een resource van het gekoppelde domein tonen.

Er bestaan twee soorten cookies :

- Een **permanent cookie** wordt in een bestand bewaard en blijft behouden als je de browser afsluit. Het krijgt bij constructie doorgaans een houdbaarheidsdatum mee en wordt door de browser verwijderd als het vervalt.

- Een **session cookie** gaat verloren als je de browser helemaal afsluit. Merk op dat een tabblad sluiten geen impact heeft op het cookie, zelfs niet als dit het laatste tabblad voor het gekoppelde domein was

Het cookie mechanisme is dus een manier om gegevens aan de client kant te bewaren die eigenlijk voor de server bedoeld zijn, vaak zijn die gegevens ook in eerste instantie afkomstig van de server zelf.

> *Naargelang hoe je het domein en optioneel path instelt, zit het wel iets ingewikkelder in elkaar dan hier wordt beschreven maar in wezen klopt de uitleg wel.*

Een belangrijke toepassing van cookies is het bijhouden van een **session id**, dit is een unieke identifier die door een server gebruikt wordt om data over jouw 'sessie' aan de serverkant terug te vinden.

Bedenk dat HTTP een toestandsloos protocol is, dus de server kan in principe niet weten of twee opeenvolgende requests van eenzelfde gebruiker afkomstig zijn. En neen, het IP-adres is onvoldoende om daarover conclusies te trekken.

Het is nochtans heel nuttig om verschillende requests van eenzelfde gebruiker te herkennen, maar dit is niet in het protocol ingebouwd en moet dus middels een omweg gebeuren. Webservers doen dit doorgaans door de session id in een cookie te stoppen. Een alternatief is om met URL rewriting de session id aan elke URL toe te voegen maar dit is omslachtiger, fragieler en heeft enkele security nadelen.

Een andere vaakgebruikte toepassing voor cookies vinden we bij het inloggen : het 'onthou mij' aankruisvakje zorgt ervoor dat je login gegevens in een cookie terechtkomen. Zolang dit cookie niet vervallen is wordt het met elke request meegestuurd, vandaar dat je bij een volgend bezoek niet meer moet inloggen.

Zie http://en.wikipedia.org/wiki/HTTP_cookie voor (veel) meer uitleg.

Web Storage API

De web storage API is een relatief nieuwe mogelijkheid om data in de browser te bewaren middels javascript code. Deze data wordt per domein (protocol, host, poortnummer) afgebakend.

Web storage is net als cookies beperkt tot key/value pairs (beiden strings). Vermits de data enkel via javascript toegankelijk is, wordt vaak een JSON string gebruikt voor de values. Deze code zal dus doorgaans JSON.stringify() en JSON.parse() oproepen bevatten om objecten, arrays of teksten als strings te bewaren.

Er zijn twee soorten opslagplaatsen in de Web Storage API :

- **session storage** is gekoppeld aan het gebruikte tabblad en blijft behouden zolang dit tabblad open blijft
- **local storage** kan gedeeld worden door meerdere tabbladen en blijft ook na het afsluiten van de browser behouden.

Beiden zijn toegankelijk via de globale variabelen sessionStorage en localStorage.

Je kan waarden uitlezen met getItem en setItem :

```
localStorage.setItem("myKey", "myValue");
console.log( localStorage.getItem("myKey") );          (*)

sessionStorage.setItem("myKey", "myValue");
console.log( sessionStorage.getItem("myKey") );        (*)
```

(*) indien er geen waarde voor "myKey" gevonden wordt, retourneert getItem(..) de null waarde.

Bedenk dat de waarden Strings zijn, als je iets ingewikkelds wil bewaren zoals tekst, arrays of objecten dan kun je hiervoor best het JSON-formaat gebruiken.

Om bv. een array in local storage te bewaren onder de naam 'scores'

```
let scores=[1,2,3,4,5];
localStorage.setItem('scores', JSON.stringify(scores));
```

Om bv. een array uit local storage te lezen dat onder de naam scores bewaard werd :

```
let scores = JSON.parse( localStorage.getItem('scores') );
```

Om waarden te verwijderen kun je de volgende methods gebruiken

removeItem(key)
verwijdert het item voor de gegeven key

clear()
verwijdert alle items (ook handig om bv. vanop de console local storage leeg te maken)

Omdat meerdere applicaties (binnen een domein) hun data in web storage kunnen stoppen, zal men elke key een applicatie-specifiek voorvoegsel geven bv. 'angrybird.scores' en 'spaceinvaders.scores' i.p.v. gewoon 'scores'.

De grootte van de data in session storage en local storage is beperkt. Browsers verschillen qua maximale grootte maar de data beperken tot 5MB per domein is een goede richtlijn.

Bij het starten van een applicatie (bv. in de setup functie) wordt eerst gekeken of er in de web storage relevante data aanwezig is. Zoja, dan zal deze gebruikt worden om de pagina op te vullen. Indien niet wordt default data gebruikt of blijven bepaalde delen van de pagina gewoon leeg.

Het beste moment waarop een applicatie haar data in web storage wegschrijft, ligt moeilijker dan je misschien zou verwachten : er is namelijk geen makkelijke en betrouwbare manier om te herkennen wanneer de gebruiker de pagina verlaat of het tabblad sluit of de browser afsluit.. Of wanneer de computer crasht 😉. Het beste dat je kunt doen is de data wegschrijven zodra ze beschikbaar is. Bijvoorbeeld :

- de highscore tabel telkens wegschrijven na een succesvol spel
- de inhoud van een editor tekening wegschrijven na elke edit
- de favoriete filter settings wegschrijven telkens de gebruiker deze filter aanpast

Web Storage API : Session Storage

Session storage koppelt key/value pairs binnen een domein aan een bepaald tabblad.

Als je in dit tabblad naar een andere pagina surft (binnen hetzelfde domein) gaan de gegevens niet verloren. Pas als je het tabblad (of de browser) sluit, wordt de data weggesmeten.

Je kan dit dus als doorgeefluik gebruiken tussen javascript programma's op verschillende pagina's.

> Je kan ook data doorgeven naar de volgende pagina via de url parameters van die pagina.
>
> Er bestaan ook libraries die het mogelijk maken naar een andere pagina te surfen door de inhoud van die pagina via aan AJAX call in te laden en dynamisch in de DOM-tree van de huidige pagina te injecteren. Dan is voor de browser geen echte pagina wijziging gebeurt en blijft het javascript programma gewoon doorlopen op de 'nieuwe' pagina. Zie bv. e JQUery load method :
>
> http://api.jquery.com/load/

Session storage wordt bv. gebruikt om te onthouden in welke toestand een pagina zich bevond toen je ervan weg navigeerde, zodat de bijbehorende javascript code de toestand kan herstellen mocht je ooit ernaar terugkeren. Bijvoorbeeld de toestand van een game dat je op die pagina aan het spelen was.

Omdat er in de session storage niet staat van welke pagina onthouden data afkomstig is, stop je deze best in een key die je pagina op unieke wijze identificeert, anders loop je het risico je op de data van een andere pagina te baseren of deze te overschrijven. Gebruik bv. een combinatie van je domein en naam van de webapplicatie.

Opdracht

Unzip het bestand 'demo storage.zip' en open index.htm uit folder 'demo 2 session storage' in je browser.

Druk een paar keer op de knop.

- Blijft de data behouden als je het tabblad sluit en de pagina opnieuw opent?
- Blijft de data behouden als je de pagina herlaadt?
- Wordt de data gedeeld door verschillende tabbladen als je de pagina meermaals opent?
- Denk je dat een andere pagina (andere file of url) aan de data geraakt als je ze in hetzelfde tabblad opent?

Web Storage API : Local Storage

Local storage onthoudt key/value pairs binnen een bepaald domein, zelfs nadat de browser wordt afgesloten.

Local storage wordt bv. gebruikt om de data van een webapplicatie te onthouden indien er geen server gebruikt wordt (of indien de server tijdelijk niet bereikbaar is).

De data is per domain afgebakend maar ook hier wordt niet onthouden bij welke web applicatie de data hoort, dus stop ze weg onder een key die de applicatie context op unieke wijze identificeert (bv. via een voorvoegsel).

Zoals eerder werd getoond, kun je data schrijven met setItem en lezen met getItem :

```
localStorage.setItem("myKey", "myValue");       // bewaart "myValue" voor "myKey"

localStorage.getItem("myKey");                  r// etourneert de value  voor "myKey" (of null)
```

Ook de eerdere opmerkingen over web storage zijn van toepassing op local storage :

- de opgeslagen value is altijd een string en vaak in JSON formaat
- getItem() levert null op indien de key niet aanwezig is
- keys krijgen een applicatie-specifiek voorvoegsel
- het gebruik van clear() en removeItem() om zaken te verwijderen
- checken en gebruiken van aanwezige data bij de start van de applicatie
- wijzigingen wegschrijven zodra de data beschikbaar wordt

Opdracht

Unzip het bestand 'demo storage.zip' en open index.htm uit folder 'demo 3 local storage' in je browser.

Druk een paar keer op de knop.

- Blijft de data behouden als je het tabblad sluit en de pagina opnieuw opent?
- Blijft de data behouden als je de pagina herlaadt?
- Blijft de data behouden als je de browser afsluit, opnieuw start en de pagina weer opent?
- Blijft de data behouden als je de pagina in een andere browser opent?
- Wordt de data gedeeld door verschillende tabbladen als je de pagina meermaals opent?
- Denk je dat een andere pagina (andere file of url) aan de data geraakt?

Web SQL Database

In sommige moderne browsers kun je vanuit javascript ook data opslaan in een lokale SQL database. De data kan dan aangepast en ondervraagd worden met de gewoonlijke SQL instructies.

De werkwijze is (veel) ingewikkelder dan bij local/session storage en wordt in deze sectie niet verder besproken. Het voordeel van de ene WEB SQL database is dat je er veel data in kwijt kunt en deze data makkelijk en selectief kunt opvragen SQL SELECT opdrachten. Bij session/local storage zul je zelf moeten filteren als je slechts een deel van de data nodig hebt.

Helaas ondersteunen niet alle moderne browser deze optie. Voor meer info, zie

https://en.wikipedia.org/wiki/Web_SQL_Database

Er werd recentelijk trouwens een alternatieve database specificatie gefinaliseerd, die zo te zien een breder draagvlak heeft :

https://en.wikipedia.org/wiki/Indexed_Database_API

Opdracht Colorpicker (zonder uitbreiding) met localStorage

Begin met de oplossing van opdracht 'Colorpicker'. Dit was de hele eerste colorpicker die we maakten, zonder Save knop dus.

Wijzig de code zodat de actuele slider settings bewaard blijven zelfs als je het browservenster afsluit. Telkens je de pagina opent moeten de slider waarden teruggezet worden.

Opdracht Colorpicker Pro

Begin met de oplossing van opdracht 'Colorpicker uitbreiding'. Dit is de uitbreiding die we maakten met een Save knop om onze favoriete kleuren onderaan te krijgen.

Wijzig de code zodanig dat de favoriete kleuren én de actuele slider settings bewaard blijven zelfs als je het browservenster afsluit. Telkens je de pagina opent worden de favorieten en slider waarden hersteld.