HTML5 Local Storage

Local Storage stellt Speicher auf dem System des Benutzers zur Verfügung – z.B. um ein einfaches Umschalten der Sprache bei mehrsprachigen Seiten zu realisieren.

Die Größe des lokalen Speichers hängt vom Browser ab, meist 5 bis 10 MB.

WebStorage-Objekte halten Benutzerdaten auf dem Client vor – entweder für die Dauer der Session (Session Storage) oder ohne definiertes Ende (*Local Storage*). Die User-Daten werden nicht mehr wie Cookies mit jedem HTTP-Request auf den Server übertragen.

Local Storage und Session Storage bilden keine Datenbank, sondern basieren auf einfachen Schlüssel-/Wertpaaren. Der Schlüssel (key) ist ein String.

Beide Arten von Speicher arbeiten mit derselben Schnittstelle.

Wie »persistent« – also wie haltbar Local Storage tatsächlich ist – lässt sich nicht voraussagen.

Einfache String-Verarbeitung

Die Werte können beliebige Datentypen vom String über Boolean, Integer oder Float sein, werden aber in allen Fällen als String gespeichert. Damit gespeicherte Zahlen wieder als Zahlen behandelt werden, müssen sie mit parseInt() oder parseFloat() in die Welt der Zahlen zurück geholt werden.

Local Storage

`localStorage.clear()`	löscht den Speicher
`localStorage.getItem(key)`	gibt null zurück, wenn der key nicht existiert
`localStorage.removeItem(key)`	löscht einen Eintrag
`localStorage.setItem(key, value)`	erzeugt einen Eintrag oder überschreibt den Eintrag ohne Warnung, wenn der key schon existiert

Beispiel HTML5 Local Storage

http://www.mediaevent.de/javascript/local-storage.html

Werte übernehmen überträgt die Formulareingaben in die Tabelle darunter. Bei einem erneuten Laden der Seite (auch wenn der Browser zwischenzeitlich geschlossen wurde) holt *Reload* die Daten aus dem Local Storage zurück und überträgt sie auch in das Formular.

Die Formulardaten bleiben im HTML5 Local Storage gespeichert, auch wenn der Benutzer das Browserfenster oder den Browser schließt und können zu jeder Zeit wieder abgerufen werden. Die Werte werden nicht wie bei HTTP-Cookies auf den Server übertragen. Die gespeicherten Daten haben kein Ablaufdatum.

Da sich die App nicht um die Verwaltung der Benutzer kümmern muss, ist die Implementierung außerordentlich einfach.

Werte werden mit der window-Methode window.localStorage.setItem in den Local Storage eingetragen und mit window.localStorage.getItem aus dem Local Storage gelesen.

```
<form action="#" method="post" id="webstorage">
```

```
    <input type="text" id="val1" value="" />
    …
    <input type="submit" id="submit" value="Werte übernehmen" />
</form>

<script type="text/javascript">
window.localStorage.setItem('Feld1',document.getElementById('val1').
value);
</script>
```

Methoden für Local Storage

Die Verarbeitung von Local Storage und Session Storage ist dieselbe – nur das Objekt unterscheidet Local Storage von Session Storage. localStorage und sessionStorage sind Objekte von Window.

`window.localStorage.clear()`	Löscht die Datenbank
`window.localStorage.getItem(fieldname,value)`	liest Werte aus einem Datenbankfeld *fieldname*
`window.localStorage.key()`	Schlüssel am angegebenen Index
`window.localStorage.setItem(fieldname,value)`	schreibt Werte in ein Datenbankfeld *fieldname*
`window.localStorage.removeItem(fieldname)`	löscht den Wert von fieldname
`window.localStorage.length()`	
`window.localStorage.remainingSpace()`	Verfügbarer Speicherplatz für das Storage-Objekt

HTML5 Javascript Session Storage

Session Storage funktioniert genauso wie Local Storage – nur mit einem anderen Objekt, nämlich mit dem window.sessionStorage-Objekt.

Darüber hinaus kann nur aus dem ursprünglichen Fenster auf den Session Storage zugegriffen werden, nicht aus anderen Fenstern oder Tabs.

Session Storage ist nicht limitiert – begrenzt nur durch die Resouren des Gastrechners.

Aber auf Cookies kann in jedem Browserfenster und jedem Tab zugegriffen werden.

Session Storage lebt genauso wie Local Storage nur im Client und wird nicht mit Requests versendet. Domain-Cookies werden mit jedem Request an diese Domain versendet – das erfordert Bandweite.

Session Storage und Local Storage sind Zeichenketten. Wenn andere Datentypen gespeichert werden soll, muss das Script die Konvertierung übernehmen.

Die Methoden für die Übernahme der Daten in den Local Storage und das Lesen der Daten aus dem Local Storage sind dieselben.

Im Gegensatz zu HTTP-Cookies kann HTML5 Local Storage den Kontext über mehrere Fenster aufrecht erhalten.

Ich kann mehrere Instanzen in unterschiedlichen Tabs oder Fenstern des Browser öffnen und jedes Browserfenster oder -tab behält seinen Kontext: Jedes Browserfenster/Tab nutzt seinen eigenen Session Storage und kümmert sich nicht um weitere Fenster. Mit HTTP-Cookies wäre das ein mühsames Unterfangen.

`window.sessionStorage.setItem(fieldname,value)`	schreibt Werte in ein Datenbankfeld *fieldname*
`window.sessionStorage.getItem(fieldname)`	liest Werte aus einem Datenbankfeld *fieldname*
`window.sessionStorage.removeItem(fieldname)`	löscht den Wert von fieldname
`window.sessionStorage.clear()`	Löscht die Datenbank

Beispiel HTML5 Local Storage

http://www.mediaevent.de/javascript/session-storage.html

Bei einem Klick auf Werte übernehmen werden die Daten des kleinen Formulars in die Tabelle darunter übertragen. Wird die Seite neu geladen, sind die Formularfelder leer – so kennen wir das.

Mit Reload können die Daten dann aus dem lokalen Speicher zurückgeholt werden.

Wird ein weiteres Fenster oder Tab geöffnet, steht im nächsten Fenster ein eigener Kontext und die Anwendung in weiteren Fenstern läuft unabhängig von der Anwendung in diesem Fenster.

Die Aufgabenzettel App- Eigene ToDo-Liste programmieren

Nachdem wir nun einen Überblick haben, wie Local Storage grundlegend funktioniert, können wir das ganze an einem einfachen Beispiel ausprobieren. Unsere Aufgabe wird es sein ein einfach ToDo Liste mit **Local Storage**, ein wenig **CSS3** und **contenteditable** zu implementieren.

Der HTML-Code: localstroge.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>Aufgabenplaner mit localStorage</title>
```

```
        <link type="text/css" rel="stylesheet"
href="localstorage_style.css" media="all" />

        <script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jquery/1.6.4/jquery.min.j
s"></script>
        <script type="text/javascript" src="storage.js"></script>
    </head>

    <body>
        <div id="box">
            <h1>Meine Aufgaben</h1>
            <ol id="todolist" contenteditable="false">
                <li>Local Storage lernen</li>
                <li>Artikel fertig schreiben</li>
            </ol>
        </div>
        <input id="edit" type="button" value="Bearbeiten" />
        <input id="clear" type="button" value="Leeren" />
    </body>
</html>
```

Wie man am HTML bereits erkennt, braucht es nicht viel Markup. Wir benötigen lediglich den **HTML5-Doctype**, inkludieren Stylesheets und JavaScript Dateien und erstellen eine Box in der eine Überschrift und eine Liste mit `contenteditable` liegt. Diese wird später unser Aufgabenzettel. Zum Schluss noch zwei Buttons welche uns Speichern/Bearbeiten und Löschen lassen.

Zum Javascripten wird als Framework **jQuery** verwendet, wird aber nicht zwingend vorausgesetzt. jQuery erleichtert uns aber das auswählen der einzelnen Elemente innerhalb des DOMs.

Als nächstes folgt der CSS Teil, welches unsere Aufgabenliste ein wenig hübscher erscheinen lässt.

Die CSS-Datei: localstorage-style.css

```
@import
url(http://fonts.googleapis.com/css?family=Gloria+Hallelujah);

body {
    background-color: #ddd;
    font-family: 'Gloria Hallelujah', Trebuchet MS, Arial, sans-
serif;
    font-size: 100%;
}

#box {
    width: 300px;
    background: #fff955;
    background: -moz-linear-gradient(top, #fefbb0, #fff955);
    background: -webkit-linear-gradient(top, #fefbb0, #fff955);
    background: -o-linear-gradient(top, #fefbb0, #fff955);
    background: -ms-linear-gradient(top, #fefbb0, #fff955);
```

```
    box-shadow: 0px 0px 3px rgba(0, 0, 0, 0.4);
    -moz-box-shadow: 0px 0px 3px rgba(0, 0, 0, 0.4);
    -webkit-box-shadow: 0px 0px 3px rgba(0, 0, 0, 0.4);
    -o-box-shadow: 0px 0px 3px rgba(0, 0, 0, 0.4);
    border-radius: 4px;
    -moz-border-radius: 4px;
    -webkit-border-radius: 4px;
    -o-border-radius: 4px;
    font-size: 12px;
    padding: 0 25px 25px;
    margin-bottom: 12px;
}

#box h1 {
    margin: 0;
    padding-top: 10px;
}

#box ol {
    margin: 1em 0 0;
    padding: 0 0 0 25px;
}

#box ol li {
    font-size: 16px;
}
```

Zunächst holen wir uns von den **Google Web Fonts** eine nette Schriftart und wählen diese als Standard aus. Unsere Aufgabenliste wird mit einem **linearen Verlauf** im Hintergrund ausgestattet. Da dies nicht viele Browser implementiert bzw. nur mit Prefixen verstehen müssen wir zuerst einen Fallback angeben und danach für jeden Browser neu definieren. Der restliche CSS Teil ist bis auf den neuen CSS3 Eigenschaften `border-radius` und `box-shadow` relativ selbsterklärend.

Zugriff auf Local Storage-Objekte

Nun fehlt uns noch der wichtigste Teil dieser ganzen App. Das Speichern, Bearbeiten und Löschen der Einträge in der Aufgabenliste.

Insgesamt werden wir vier einfach Funktionen schreiben. Zum einen wollen wir Speichern können, bereits fertige Arbeiten aus der Liste entfernen, beim Neuladen der Seite unsere Einträge wieder haben und als letzte Funktion unsere Liste bearbeiten können.

```
function saveContents() {
    var todoList = $('#todolist').html();
    localStorage['todoList'] = todoList;
}
```

Das Speichern der Aufgaben ist leicht. Wir holen uns den Inhalt der Liste und Speichern dieses in das localStorage-Objekt mit dem Key "todoList". Wenn wir das ganze so handhaben, wäre es später ohne Probleme möglich mehrere Listen anzulegen und diese jeweils mit einem eigenen Key im localStorage zu speichern.

```
function restoreContents() {
    var myTodoList = localStorage['todoList'];
    if (myTodoList != undefined) {
        $('#todolist').html(myTodoList);
    }
}
```

Diese Funktion stellt die Liste wieder her. Dabei wird das localStorage-Objekt ausgelesen und zurück in die Liste geschrieben. Sollte localStorage noch leer sein, beispielsweise beim allerersten Aufruf, schreiben wir nichts in die Liste, sondern belassen das unserem im Markup festgelegten Inhalt.

```
function toggleEditContent(e) {
    if ($('#todolist').attr('contenteditable') == 'false') {
        $('#todolist').attr('contenteditable', 'true');
        $('#edit').val('Speichern');
        $('#todolist').focus();
    } else {
        $('#todolist').attr('contenteditable', 'false');
        $('#edit').val('Bearbeiten');
        saveContents();
    }
}
```

Wie der Name der Funktion schon verrät, löschen wir den Inhalt aus localStorage und laden unsere Seite neu. Damit wird der Inhalt des Markups angezeigt.

```
function resetContent(e) {
    localStorage.clear();
    window.location.reload();
}
```

Die letzte Funktion setzt unser `contenteditable`-Attribut richtig. Zum Bearbeiten des Elements muss dieses auf `true` gesetzt werden und sollte auch gleichzeitig den **Fokus** erlangen. Wollen wir nach dem Bearbeiten unsere Eingaben speichern, stellen wir das ganze auf `false` zurück und rufen unsere Speichern Funktion auf.

Was uns grundsätzlich noch fehlt ist die **Klick-Events** auf die Buttons abzufangen und diese mit unseren Funktionen zu verbinden. Zusätzlich sollten wir noch unsere Funktion zum Wiederherstellen der Liste angeben, damit beim Neuladen der Seite oder Beenden und Neustarten des Browsers unsere Einträge wieder erscheinen.

Hier die fertige Javascript-Datei:

Fertige JavaScript-Datei: storage.js

```
$(document).ready(function() {

    restoreContents();

    $('#edit').bind('click', toggleEditContent);
    $('#clear').bind('click', resetContent);

    function saveContents() {
        var todoList = $('#todolist').html();
        localStorage['todoList'] = todoList;
```

```
    }

    function restoreContents() {
        var myTodoList = localStorage['todoList'];
        if (myTodoList != undefined) {
            $('#todolist').html(myTodoList);
        }
    }

    function toggleEditContent(e) {
        if ($('#todolist').attr('contenteditable') == 'false') {
            $('#todolist').attr('contenteditable', 'true');
            $('#edit').val('Speichern');
            $('#todolist').focus();
        } else {
            $('#todolist').attr('contenteditable', 'false');
            $('#edit').val('Bearbeiten');
            saveContents();
        }
    }

    function resetContent(e) {
        localStorage.clear();
        window.location.reload();
    }

});
```

Zusammenfassung

Arbeiten mit Local Storage ist einfach und stellt eine schicke Möglichkeit dar, Daten lokal zu speichern ohne an die Begrenzungen von Cookies zu stoßen. Somit lassen sich kleine **Browserapps** schnell entwickeln. Sei es einen Stundenplan, Termin- und Aufgabenliste oder eine App zum Verwalten von Kontakten. Beachten sollte man allerdings das jeweils nur ein Localstorage Objekt pro Domain angelegt wird und nur moderne Browser dies auch unterstützen. Zusätzlich hat man noch das Problem das die Daten **nicht Cross-Browser** seitig gespeichert werden. In einem anderen Browser sind die Daten dann einfach nicht da.

Quelle

http://www.webmasterpro.de/coding/article/javascript-html5-local-storage-tutorial.html