Moderne Webanwendungen mit HTML5

Studienarbeit an der Fakultät für Informatik und Mathematik der Hochschule München

Studiengang: Wirtschaftsinformatik

Veranstaltung

Technologien zur Entwicklung verteilter Java-Anwendungen

SS 2013

eingereicht von: Michael Reißig

Matrikelnummer 34269013

Veranstaltungsleitung: Dipl.-Inform. Michael Theis

Inhaltsverzeichnis

Verzeichnis der Code Listings

Tabellenverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

W3C	World Wide Web Consortium
WHATWG	Web Hypertext Application Technology Working Group
DOM	Document Object Model
API	Application Programming Interface, zu dt. Programmierschnittstelle
DRM	Digital Rights Management; bezeichnet technische Maßnahmen zur digitalen Rechteverwaltung, mittels derer die Einhaltung von Urheberrechten sichergestellt werden soll
HTML	Hypertext Markup Language
Chrome	WebBrowser der Firma Google
CSS	Cascading Stylesheet
SVG	Scalable Vector Graphics
JSON	JavaScript Object Notation

1 Aufgabenstellung

Diese Arbeit soll sich mit dem derzeit in der Entwicklung befindlichen Standard von HTML 5 beschäftigen. Obwohl seine endgültige Fassung noch aussteht, werden viele der neuen Funktionalitäten bereits von den Webbrowsern unterstützt. Die Berücksichtigung dieser Neuheiten in aktuellen Projekten gewinnt immer mehr an Bedeutung.

„HTML 5 soll das bestehende HTML um "Rich Web Content" erweitern. Neue Schnittstellen, um zweidimensionale Grafiken zu zeichnen, die bessere Einbettung von Video- und Audioinhalten sowie zahlreiche neue Elemente zur besseren Auszeichnung in HTML-Seiten sind die Besonderheiten von HTML 5. Zudem hält die Spezifikation neue Möglichkeiten für Web-Applikationen bereit“ [golem.de].

2 Historie

Die aktuelle vom W3C 1999 verabschiedete HTML Spezifikation trägt die Versionsnummer 4.01. Betrachtet man die rasante Entwicklung des Internets und die propagierte Halbwertszeit der enthaltenen Informationen von 18 Monaten [Wenz, 2012], so erscheint eine Weiterentwicklung des HTML Standards nach 14 Jahren, um den aktuellen Anforderungen gerecht zu werden, mehr als überfällig.

Die WHATWG, welcher mehrere Browserhersteller angehören, sowie das W3C befürworten hierbei allerdings stellenweise unterschiedliche Philosophien. Während das W3C eher den akademischen Ansatz vertritt, nachdem der neue Standard möglichst auf alle Fragen eine Antwort liefern soll, praktiziert die WHATWG einen agilen Standardisierungsprozeß, welcher neue Funktionalitäten frühzeitig in die Browserversionen aufnimmt, auch wenn dies bedeutet, daß hier später möglichweise nachgebessert werden muß. Deploymentseitig stellt dies insofern kein Problem dar, da die meisten Browser heutzutage eine automatische Aktualisierungsfunktion besitzen. Bei Firefox gab es beispielsweise sogar die Diskussion, ob die Versionsnummer relevant sei, und HTML nicht eher als „lebendiger Standard“ gesehen werden sollte [Wenz, 2012]. Auf Grund dieses versionslosen Ansatzes verzichtet die Arbeitsgruppe auf die „5“ und spricht nur vom „HTML-Standard“ [WHATWG, The WHATWG Blog, 2011].

Einer der Streitpunkte ist das Markup (HT**M**L) selbst. Zu den Grundprinzipien des W3C zählt die Bariierefreiheit, welche nach Ansicht des W3C (nur) durch eine XML Notation gewährleistet wird. Deshalb auch im Jahr 2000 die Verabschiedung des XHTML Standards, als überarbeitete Version von HTML 4.01.

Demzufolge muß jedes Element mit dem backslash beinhaltenden End-Tag geschlossen sein , bspw. Blockelemente wie *html* und *title* gleichermaßen wie z.B. Standalone-Elemente dem *br*. Dies auch dann, wenn letztere nie Inhalt beinhalten :

```
<!DOCTYPE html>
<html>
<head>
    <title>Mein Titel</title>
</head>
<body>
    Lorem ipsum dolor sit amet,
    < br />
    consetetur sadipscing elitr,
</body>
</html>
```

Listing 1: W3C Schließ-Tag

Nach Ansicht der WHATWG ist diese Pauschalisierung allerdings unnötig, da sich anhand der Sprachelemente erkennen läßt, ob ein Inhalt zu erwarten ist oder nicht, und somit ein End-Tag benötigt wird. Das Element *br* müßte demnach nicht geschlossen werden :

```
<!DOCTYPE html>
<html>
<head>
    <title>Mein Titel</title>
</head>
<body>
    Lorem ipsum dolor sit amet,
    <br >
    consetetur sadipscing elitr,
</body>
</html>
```

Listing 2: WHATWG Schließ-Tag

[Wenz, 2012]

2.1 Fertigstellung von HTML5

Gemäß ihrer Webseite will das W3C HTML5 2014 offiziell verabschieden. Im Mai 2011 erhielt HTML5 den Status „Last Call“, welcher als letzte Aufforderung dienen soll, Kommentare zum HTML5-Entwurf einzureichen. [W3C, W3C Confirms May 2011 for HTML5 Last Call, Targets 2014 for HTML5 Standard, 2011] Die WHATWG hat den Status „Last Call“ 2009 ausgerufen. [WHATWG, HTML5 at Last Call, 2009] Der Status „Last Call“ bedeutet aber auch, dass HTML5 faktisch bereits einen fertigen Zustand angenommen hat, wenngleich die Browser die entsprechenden Funktionen teilweise noch erlernen müssen.

3 Bestandteile von HTML5

HTML5 ist ein Oberbegriff, welcher sich aus verschiedenen Spezifikationen zusammensetzt:

HTML + CSS 3 + JavaScript APIs = 

Diese erweitern die Sprache um mehrere neue Funktionalitäten:

• Markup	• Selectors	• Geolocation
• Forms	• Media Queries	• Web Storage
• ARIA	• Fonts	• Web Sockets
• Microdata	• Transforms	• FileAPI
• Canvas	• Transitions	• Media Capture
• Video	• Animations	• IndexedDB
• Etc.	• Etc.	• Etc.

Tabelle 1: neue HTML Funktionalitäten

HTML erweitert den semantischen Inhalt und standardisiert das DOM[1] sowie die DOM APIs[2]. [Grosland, 2012]

Auf einige der Neuerungen soll im Folgenden näher eingegangen werden.

[1] Document Object Model
[2] Application Programming Interface, zu dt. Programmierschnittstelle

4 Html Spezifikation

Mit HTML 5 wird der *DOCTYPE* einfacher (und vor allem einprägsamer). Dieser sagt aus, nach welchem Standard die Website erstellt worden ist.

```
<!DOCTYPE html>
```

Listing 3: HTML5 Doctype

4.1 Semantische Elemente

Die Spracheerweiterung enthält zudem neue HTML-Elemente um Bereiche einzuteilen. Was bisher mit *div* und einem Attribut wie *id* oder *class* erfolgte, kann nun direkt mit einem logischen HTML-TAG ausgezeichnet werden. Diese Schlüsselwörter sind das Ergebnis der statistischen Auswertung zahlreicher bestehender Webseiten durch das W3C, in welcher die gebräuchlichsten *id* Definitionen näher untersucht wurden.

Die wesentlichen strukturierenden Elemente sind:

Element	Anwendungsfall
• article	repräsentiert einen Abschnitt
• aside	Definition von Querverweisen, zusätzliche Informationen usw.
• footer	Fusszeilen von Abschnitten
• figcaption	Fußzeile von gruppierten Medieninhalten
• header	Kopfbereich eines Abschnittes
• hgroup	dient dem Gruppieren von Überschriften (<h1> – <h6>).
• mark	Hervorheben eines Textes, i.d.R. um den Bezug zu einem übergeordneten Kontext herzustellen
• nav	Navigationsbereiche
• section	Gruppierung von Abschnitten
• time	auszeichnen von Datums- und/oder Zeitangaben

Tabelle 2: Semantische Elemente

Auch wenn die Namen sprechend sind, ein Layoutbeispiel für einen typischen Einsatz :

Abbildung 1: Semantische Elemente. Quelle: [Marsman, HTML5 Part 1: Semantic Markup and Page Layout, 2011]

Diese semantischen Erweiterungen bringen keinerlei zusätzliche Funktionalität mit sich. Aber was ist dann ihr Zweck? Auf Grund dessen, daß die Bereiche einer Webseite nicht mehr ausschließlich über *div*, sondern durch einheitliche konkretere Elemente beschrieben werden, werden Webseiten besser intepretierbar. Suchmaschinen könnten bspw. zwischen Schlagwörtern im *nav* Bereich und allgemeinem Text in *article* differenzieren, wodurch die Suchergebnisse in einer höheren Qualität geliefert werden können. Lesehilfen können den Anwender zielführender unterstützen, um nur ein paar Anwendungsfälle zu nennen.

Der Einsatz dieser Sprachelemente ist somit zum heutigen Stand eine Investition in die Zukunft, sofern die Browser dies kommend auswerten. [Wenz, 2012]

4.1.1 Beispiel

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>Semantische Visualisierung</title>
</head>
<body>
    <header>
        <h1>Header in h1</h1>
        <h2>SubHeader in h2</h2>
    </header>
    <nav>
        <ul>
            <li><a href="*">Menü Option 1</a></li>
            <li><a href="*">Menü Option 2</a></li>
        </ul>
    </nav>
    <section>
        <article>
            <header>
                <h1>Article #1</h1>
            </header>
            <section>
                Das hier ist der erste Artikel. Dieser Text ist <mark>markiert</mark>.
            </section>
        </article>
    </section>
    <aside>
        <section>
            <h1>Links</h1>
            <ul>
                <li><a href="#">Link 1</a></li>
                <li><a href="#">Link 2</a></li>
                <li><a href="#">Link 3</a></li>
            </ul>
        </section>
        <figure>
            <img width="85" height="85"
                src="http://w3-
mediapool.hm.edu/mediapool/media/dachmarke/dm_lokal/presse/news_1/bilder_48/2013_3/04_
8/Cover_forschungsNEWS_ContentTeaser.jpg"
                alt="forschungs News" />
            <figcaption>Hochschule München</figcaption>
        </figure>
    </aside>
    <footer>
        Footer - Diese Seite wurde am 1.04.2013 erstellt
    </footer>
</body>
</html>
```

Listing 4: Semantische Visualisierung

4.2 Webforms - Typisierte Felder

Beim Entwickeln von Eingabemasken gelangt man über kurz oder lang an Fall, daß komplexere Daten als Text, wie Datum oder Uhrzeit, abgefragt werden. Das besondere an diesen Datentypen ist, daß Sie einem bestimmten Format unterliegen oder einen konkreten Wertebereich haben. Bislang fehlten den Browsern geeignete Boardmittel um Eingaben dieser Typen zu unterstützen, sodaß die Entwickler versuchten über Workarounds, seien es nun Controls oder einfach mittels Dokumentation, dem Anwender entgegenzukommen.

HTML5 schließt diese Lücke und führt zahlreiche neue *input type* s ein, wie

- date
- datetime
- email
- month

usw.

Auf Basis dieser Angaben können Browser für die Eingabe optimierte UI Elemente zur Verfügung stellen, welche die Usability erheblich verbessern. Neben dem Vorteil, die technischen Möglichkeiten des Clients auszureizen, könnten weitere Synergien generiert werden.

Der *inputtype="email"* ist hierfür ein gutes Beispiel. Denkbar wäre bei Betreten eines solchen Feldes auf Kontaktdaten der lokalen Kontakt-Datenbank zuzugreifen, und diese mittels Auswahl oder Wortvervollständigung zur Verfügung zu stellen, und dies ist erst der Anfang.

Ein weiterer Vorteil ergibt sich durch die damit verbundene clientseitige Validierung der Eingaben, indem unnötige Roundtrips zum Server reduziert werden. Auch wenn dies eine serverseitige Prüfung in keinster Weise hinfällig macht.

Eine detailiertere Übersicht zu den möglichen Input-Attributen findet man unter https://developer.mozilla.org/en-US/docs/HTML/Element/Input. Mindestens genauso interessant ist die dortige Kompatibilitätsaufstellung von Browserversionen und HTML-Funktionen.

4.2.1 Beispiel

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>Typisierte Felder</title>
</head>
<body>
   date: <input type="date"  /><br/>
   time: <input type="time"  /><br/>
   email: <input type="email"  /><br/>
   month: <input type="month"  /><br/>
</body>
</html>
```

Listing 5: Beispiel neue Eingabetypen

Das Dokument in obigem Beispiel enthält lediglichein Label sowie den Inputtyp des Feldes. Diese Informationen genügen Chrome, um eine (an-)sprechende Eingabemaske für jedes der Felder zu generieren (im Screenshot zu sehen für das Datumsfeld):

Abb. 1: Darstellung Feld date im Browser Chrome

4.3 Multimedia

Eine weitere Neuerung ist die Wiedergabe multimedialer Streams. Wofür bislang ein Plugin wie Flash oder Silverlight benötigt wurde, kann nun über die Tags *audio* und *video* abgebildet werden. Neu ist dabei, daß die Webseite die Ressourcen lediglich zur Verfügung stellt, die Wiedergabe aber vollständig dem Client überläßt. Auf Grund dessen wird hierrüber auch kein DRM[3] unterstützt. Multimediadateien liegen in einem bestimmten Format vor, welches durch einen Codec[4] gekennzeichnet wird. Clients können ein Format nicht wiedergeben (dekodieren), wenn ihnen der entsprechende Codec fehlt oder auch aus anderen Gründen.

Darauf reagierend wurde die Möglichkeit geschaffen, eine multimediale Ressource in verschiedenen Formaten anzubieten. Der Client kann über diese Liste an Quellen iterieren, bis er auf ein durch ihn wiedergabefähiges Format stößt. Endet dies nicht erfolgreich, kann eine entsprechende Meldung oder ein alternativer Fallback definiert werden, bspw. das Zurückgreifen auf besagte Plugins.

Der Vorteil solcher Vorgehensweisen besteht darin, daß die Steuerung der Wiedergabe, also das UserInterface, in einer auf den Client optimierten Form erfolgen kann, welche sich aktuell bspw. bei einem mobilen Endgerät und einem Desktop-Pc unterscheiden.

In den Tests konnte der Internet Explorer 10 das entsprechende HTML nicht interpretieren und warf eine Fehlermeldung. Chrome hatte im Gegensatz keine Schwierigkeiten, ignorierte allerdings die Fallbackdefinition.

[3] Digital Rights Management bezeichnet technische Maßnahmen zur digitalen Rechteverwaltung, mittels derer die Einhaltung von Urheberrechten sichergestellt werden soll

[4] Ein Codec ist Software, die zum Komprimieren oder Dekomprimieren digitaler Mediendateien wie Musiktitel oder Videos verwendet wird [Microsoft, 2013]

4.3.1 Audio & Video

Das folgende Listing stellt eine Audio Datei bereit:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>Audio Beispiel</title>
</head>
<body>
    <audio controls="controls" autoplay="autoplay" preload="auto">
        <source src="Test.mp6" type="audio/ogg" />  <!-- 1.Versuch einer Wiedergabe
                                                        wenn erfolgreich exit Tag-->
        <source src="Test.mp3" type="audio/mp3" />  <!-- 2.Versuch einer Wiedergabe
                                                        wenn erfolgreich exit Tag-->
        Der Dateityp wird nicht unterstützt.        <!-- Fallback, i.d.F. Meldung im
                                                        Browser-->
    </audio>
</body>
</html>
```

Listing 6: Audio Tag

Die *audio* Attribute stehen dabei für:

`controls="controls"`: dem Browser wird mitgeteilt, daß Steuerelemente zur Verfügung gestellt werden sollen. Andernfalls kann der Anwender keinen Einfluß auf die Wiedergabe nehmen.

`autoplay="autoplay"`: der Browser startet selbständig mit der Wiedergabe

`preload="auto"` : puffert/cacht den Stream

Gleiches gilt für die Bereitstellung und Wiedergabe von Videodateien, das entsprechende Tag lautet hierfür *video.*

4.4 Zeichnen in HTML5 mittels Canvas

Ein weiteres neues Element ist das *canvas*. Es steht für das dt. Wort „Leinwand“ und stellt dementsprechend eine leere Fläche zum Zeichnen dar. Es ist somit ein Container für Grafiken.

Der Zugriff um zu Zeichnen erfolgt mittels javascript, es können allerdings Attribute wie Höhe und Breite direkt beim Element angegeben werden. Ein Beispiel ist unter 4.4.2 dargestellt.

4.4.1 Vergleich Canvas vs. SVG

Der aktuell führende Standard zum Zeichnen im Browser ist Scalable Vector Graphics (SVG). SVG ist die vom W3C empfohlene Spezifikation zur Beschreibung zweidimensionaler Vektorgrafiken. Mit Veröffentlichung des neuen *canvas* stellt sich die Frage nach der Koexistenz beider Standards bzw. ihrer Vergleichbarkeit.

Der markanteste Unterschied besteht nach [Grosland, 2012] darin, daß canvas „immediate mode rendering“ und SVG „retained mode rendering“ verwendet.

immediate mode:

- anzuzeigende Daten liegen im Pixelbuffer
- nur Pixel, keine Grafikobjekte

retained mode:

- anzuzeigende Daten in Display List
- Grafikobjekte besitzen Attribute
- 2 Renderingverfahren

[Zins, 2011]

Das bedeutet, daß *canvas* die Grafiken direkt rendert. Zur Laufzeit wird die Grafik aus dem Bsp. unter 4.4.2 deshalb auf Grund des *immediate mode* nach dem Rendering verworfen, und der Status wird nicht gespeichert. Eine Änderung würde somit ein vollständiges Neuzeichnen erfordern. *Canvas* zeichnet sich somit durch eine vergleichsweise geringe Komplexität aus.

Im Unterschied dazu hebt SVG das komplette Grafikmodell auf. Dadurch können die Grafikobjekte nachträglich modifiziert werden und der Browser kümmert sich um das Rendering, welches grunsätzlich auf das Delta beschränkt erfolgen kann. Des Weiteren kann bei SVG zusätzlich zu javascript über CSS Einfluß auf die Grafik genommen werden.

Hier muß der konkrete Anwendungsfall entscheiden, welche Technologie geeigneter ist.

Einen etwas granulareren Vergleich soll die folgende Gegenüberstellung liefern:

	Canvas	SVG
Abstraction	Pixel-based (dynamic bitmap)	Shape-based
Elements	Single HTML element	Multiple graphical elements which become part of the Document Object Model (DOM)
Driver	Modified through Script only	Modified through Script and CSS
Event Model	User Interaction is granular (x,y)	User Interaction is abstracted (rect, path)
Performance	Performance is better with smaller surface and/or larger number of objects	Performance is better with smaller number of objects and/or larger surface

Tabelle 3: Canvas vs. SVG, Quelle [Marsman, HTML5 Part 2: Canvas, 2011]

4.4.2 Canvas Beispiel

Ein gutes Beispiel für den Gebrauch des *canvas* ist das Zeichnen folgender Flagge. Zielstellung soll diese Grafik sein:

Abb. 2: Canvas Beispiel Flagge

Der dazugehörige Code stellt sich wie folgt dar:

```
<!DOCTYPE HTML>
<html>
<body>
    <!--definiert das Element sowie eine Meldung, sollte der Browser
        die canvas Funktionalität nicht bereitstellen -->
    <canvas id="myCanvas">Ihr Browser unterstützt canvas nicht.</canvas>

    <script type="text/javascript">

        // Zugriff auf den Verweis des Elements
        var canvas = document.getElementById('myCanvas');

        /* läd den 2D-Context. Dieser wird benötigt, weil man letzlich
        nicht auf dem <canvas>-Element selbst, sondern auf einem Context
        des Elements zeichnet.          */
        var ctx = canvas.getContext('2d');

        //setzt die Füllfarbe für alles, was im Context hiernach gezeichnet wird auf
blau
        ctx.fillStyle = '#2222FF';
        //zeichnet ein Rechteck von links-oben nach rechts-unten
        ctx.fillRect(0, 0, 125, 75);

                        // Zeichnen eines weißen X

        //startet den Prozess einen Pfad zu zeichnen
        ctx.beginPath();
        ctx.lineWidth = "15";

        //Farbe des Pfades
        ctx.strokeStyle = "white";

        /*Wegbeschreibung das X
        wobei moveTo den Cursor verschiebt, ohne zu zeichnen.
        lineTo hingegen zeichnet eine Linie
        */
        ctx.moveTo(0, 0);
        ctx.lineTo(125, 75);
        ctx.moveTo(125, 0);
        ctx.lineTo(0, 75);

        // Zeichnen des Pfades
        ctx.stroke();

    </script>
</body>
</html>
```

Listing 7: Canvas Element

5 CSS3

Tablets sowie Smartphones revolutionierten unseren Medienkonsum. Neben einer reduzierten Bildschirmgröße bringen diese Geräte eine um Gesten erweiterte Benutzerschnittstelle mitsich. Sie erweitern damit die Vielzahl an Auflösungen, in denen Webseiten dargestellt werden. Allein durch ihre Fähigkeit die Ansicht in 90° Schritten zu drehen, bringen diese Geräte bereits von Haus aus zwei Auflösungen mit. Statt mehrere gerätespezifische Varianten einer Webseite zu pflegen, ist es i.d.R. effizienter eine universelle Version anzubieten und diese mittels CSS auf das jeweilige Gerät zu optimieren.

5.1 Media Queries

Diesem Ansinnen widmet sich das Responsive WebDesign (auch Resposives Website Design). Basis dieses Designs sind die Media Queries[5] - eine 2010 vom W3C empfohlene CSS-Erweiterung, die mit CSS3 bereits in den gängigen Browsern umgesetzt wurde [Lorenz, 2012]

Eine in diesem Design umgesetzte Website lässt sich meist an einer an die Bildschirmbreite geknüpften Darstellung erkennen. Ein aktuelles Beispiel ist der Webauftritt der ARD (www.ard.de).

Schrittweise mit Ändern der Fensterbreite ändert sich auch die Darstellung. Untersucht man den Quellcode, erkennt man, daß zwischen 5 Breitenbereichen unterschieden wird. 3 davon sollen näher betrachtet werden (bei konstanter Fensterhöhe , Screenshots reduziert auf 22% der Originalgröße):

Style Bereich	XS	S	M
Fensterbreite (px) des Browsers	316	504	752

Abb. 3: ARD 316x854 px

Abb. 4: ARD 504x851 px

Abb. 5: ARD 752x854 px

Tabelle : ARD.de Screenshots

Wie man auf den Abbildungen erkennen kann, ändern sich

- Anordnung der Elemente
- Schriftgröße
- Bildgröße
- sowie die Sichtbarkeit angezeigten Elemente ansich

[5] W3C Candidate Recommondation Media Queries, www.w3.org/TR/css3-mediaqueries

Erreicht wird diese Anpassung durch die wechselseitig formulierten Bedingungen, welche über jQuery geladen werden:

- XS:

```
'screen and (min-width: 0px) and (max-width: 479px)': {
'css': ['/clientCode/101806/4788421527150168942/css_xs.css'],
```

- S:

```
'screen and (min-width 480px) and (max-width: 767px)': {
'css': ['/clientCode/101724/4788421529145489422/css_s.css'],
```

- M:

```
'screen and (min-width: 768px) and (max-width: 1009px)': {
'css': ['/clientCode/101392/4788421526579114741/css_m.css'],
```

Dabei enthält der 1. Parameter die logischen Bedingungen, die sogenannten Media Features, welche mit *and* verknüft werden. Für die „M“ Konstellation wird in diesem Fall eine minimale Fensterbreite von mindestens 768 und maximal 1009 Pixeln angenommen. Der css Parameter enthält den Pfad der Stylesheetdatei . Diese kommt zur Anwendung, wenn die Bedingung erfüllt ist.

In der Praxis sind vor allem die Größenangaben relevant. Diese können entweder mit festen Werten oder in Kombination mit *min* oder *max* als Präfix notiert werden.

Folgende größenbezogenen Features können entsprechend der W3C-Empfehlung verwendet werden:

- width, height
- device-width
- device-height

Daneben gibt es aber noch weitere Features wie *color*, *resolution* usw.

5.2 Media Queries in CSS-Dateien

Statt die Bedingungen innerhalb der aufrufenden HTML-Seite zu notieren und damit für jeden Seitenaufruf neu zu laden, sollten Abschnitte allerdings besser innerhalb einer CSS-Datei definiert werden. Dieser Ansatz bietet sich in der Praxis an, da er wesentlich flexibler und vor allem granularer arbeitet als der Austausch der gesamten CSS-Datei. Der Bereich wird mit *@media* und der Bedingung eingeleitet und mit geschweiften Klammern umschlossen [Lorenz, 2012]:

```
@media screen and (min-width: 768px) and (max-width: 1009px)
{
    /*do something*/
}
```

Listing 8: Media Query in CSS

6 Javascript API

6.1 Web Storage

Web Storage ist der übergeordnete Name der APIs, welche den lokalen Speicherplatz im Client zur Verfügung stellen. Wikipedia nennt zudem die Bezeichnungen „Supercookies“ und „DOM Storage“. Ursprünglich war der Web Storage Bestandteil der HTML5-Spezifikation, inzwischen wurde dieser in eine eigene W3C-Spezifikation ausgelagert [W3C, Web Storage, 2013].

Web Storage beinhaltet die beiden APIs *local storage* und *session storage*. Beide Varianten bieten dieselben Methoden an. Während die im *session storage* abgelegten Daten jedoch beim Schließen des Browsers wieder gelöscht werden, bleiben die im *local storage* gespeicherten Daten bis zum nächsten Start der Anwendung erhalten.

„Während man zu Recht viele HTML5-Erweiterungen wegen der unterschiedlichen Unterstützung durch aktuelle Browser noch mit Vorsicht genießen muss, ist der *local storage* bereits in allen modernen Browsern implementiert.“ [Lauer, 2013]

Unter [Web Storage Support Test] kann man testen, wieviel eine Browser an local-, session- oder global Storage, also Speicherplatz, zur Verfügung stellt. Für den getesten Chrome wurden 2,5 MB *locale storage* verzeichnet.

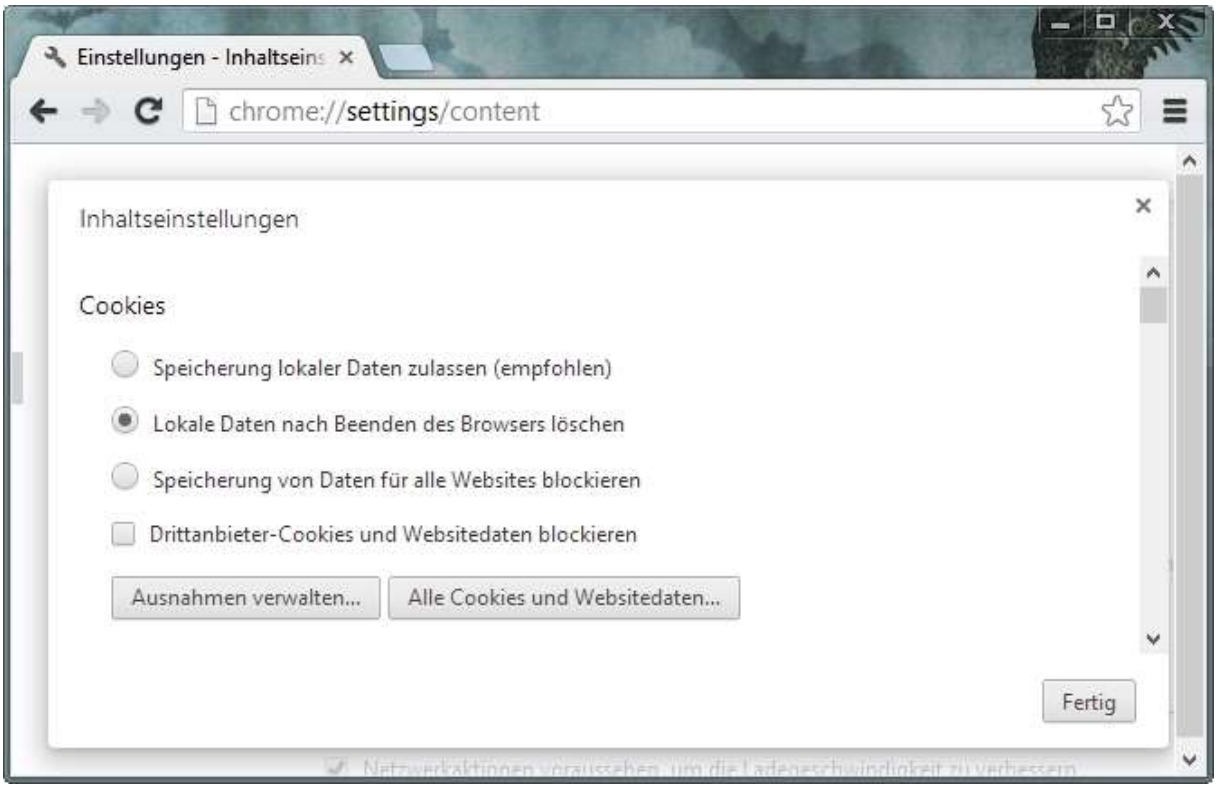

Abbildung 2: Einstellungen für WebStorage im Chrome

Die lokalen Daten werden, wie bei Cookies üblich, nicht zwischen den Browsern geshared. D.h., besucht man eine Webseite beim nächsten Mal mit einem anderen Browser, stehen vormals erstellte Daten nicht zur Verfügung.

6.1.1 Web Storage API

Alle Anweisungen für den *local storage* sind identisch auch für den *session storage* implementiert.

Die wichtigsten Methoden im Überblick:

• *localStorage.setItem(key, value)*	speichert einen Wert (value) und markiert ihn mit dem Schlüssel (key).
• *localStorage.getItem(key)*	liest den unter dem Schlüssel gespeicherten Wert wieder aus.
• *localStorage.removeItem(key)*	löscht den unter dem Schlüsselgespeicherten Wert.
• *localStorage.length()*	liefert die Anzahl der gespeicherten Datensätze.
• *localStorage.key(index)*	übergibt den Schlüssel zum Wert an der Positionindex.
• *localStorage.clear()*	löscht alle von der aktuellen Domain im lokalen Speicher abgelegten Daten – also auch die anderer Anwendungen derselben Domain.

Tabelle 4: die wichtigsten Methoden der *locale storage* API. Quelle [Lauer, 2013]

Und deren praktischer Einsatz:

```
//localStorage.js

function sample() {
    //lesen von Werten aus den Controls
    var key = document.getElementById("key").value;
    var wert = document.getElementById("wert").value;

    //Werte im localStorage speichern
    localStorage.setItem(key, wert);

    //Wert aus localStorage lesen
    var wert = localStorage.getItem(key);

    //Wert im localStorage löschen
    localStorage.removeItem(key);
}
```

Listing 9: Einsatz der wichtigsten *locale storage* Methoden

Locale storage kann neben strings auch Objekte speichern. Hierzu bedient man sich des bekannten JSON, welches über *JSON.stringify* ein Objekt serialisieren[6] bzw. über *JSON.parse* deserialisieren kann.

[6] Serialisieren, auch bekannt als Marshalling

Das serialiserte Objekt, also der String, kann, wie in Listing 9: Einsatz der wichtigsten *locale storage* Methoden dargestellt, daraufhin gespeichert werden.

[Lauer, 2013]

6.2 Offline Modus

Interessant wird die Funktionalität des *locale storage* insbesondere im Zusammenhang mit dem Offline Modus von HTML5, bzw. ist als eine Basiskomponente zu sehen.

Die zentralisierte Bereitstellung von Webanwendungen hat einen wesentlichen Nachteil – sie ist von einer Internetverbindung abhängig. Wird diese unterbrochen ist die Anwendung nicht mehr erreichbar, wodurch Daten verloren gehen können. Aus diesem Grund bietet HTML5 einen vollständigen Offline Modus für Applikationen [Sidorencov & Kirchner, 2011].

Das DOM des HTML5 ermöglicht die Prüfung, ob eine Verbindung zu einem Netzwerk besteht.

```
//offlineModus.js

function getOnlineStatus() {
    var connected = window.navigator.onLine ? 'online' : 'offline';
}
```

Listing 10: Ermitteln des OnlineStatus

Betrachtet man folgendes Szenario:

- In einer Webapplikation wurden clientseitig Daten erzeugt bzw. manipuliert;
- Der Anwender möchte nun die neuen Daten an den Server übertragen und stößt den *submit* an;
- Auf Grund irrelevanter Umstände besteht zu diesem Zeitpunkt keine Verbindung zum Server,

könnte dieses Problem mittels den neuen Funktionen wie folgt bewältigt werden:
Die Applikation prüft bei Aufruf des *submit* ob eine Verbindung existiert. Ist dies der Fall,werden die Daten gesendet. Ist dies nicht der Fall, werden die Daten im *locale storage* gespeichert und der Anwender erhält bspw. einen Hinweis, es später nocheinmal zu versuchen.
Entweder werden die Daten, getriggert durch den Anwender, bei einem der folgenden *submit* Versuche übertragen, oder aber automatisch beim nächsten Aufruf der Webseite.

Aber *locale storage* eröffnet noch weitere Möglichkeiten. Es können Daten vorab clientseitig geladen werden („Caching“), wodurch der Client in die Lage versetzt wird, Pfade innerhalb der Applikation, deren Daten zwischengespeichert wurden, offline, also ohne Verbindung zum Server, zurückzulegen.

Dies alles erhöht aber nicht unwesentlich die Komplexität innerhalb einer Applikation, und sollte deshalb fallspezifisch abgewogen werden.

7 Abwärtskompatibilität

Eine neue Softwareversion wirft immer die Frage nach der Abwärtskompatibilität auf. Als Brandon Satrom auf der “Microsoft tech-ed” 2011 über das Internet sagte: „The web ist the largest Legacy Software System in the History oft he Universe“ [Satrom, 2011] , hat er gewiß nicht übertrieben.

Auch wenn aktuelle Browserversionen HTML5 unterstützen, ist dies kein Garant, daß HTML5 Webseiten korrekt dargestellt werden:

- viele Anwender benutzen noch keine HTML5 kompatiblen Browser
- jeder Browser unterstützt unterschiedliche Teilmengen von HTML5
- die Browser bilden gleiche Feature unterschiedlich ab

7.1 Polyfills

Paul Irish zufolge ist ein Polyfill ein Unterbau, der eine zukünftige API immitiert und Ausweichfunktionalität, sog. Fallbacks, für ältere Browser bietet [Irish]. Dies ermöglicht es eine HTML5 Funktionalität zu implementieren, ohne oben genannte Anwender auszuschliessen.

Eine solche Unterstützung bietet die Javascript Bibliothek *Modernizer* [7]. Durch Einbinden in die Webseite erhält man Zugriff auf Funktionalitätserkennungsfunktionen, welche bspw. prüfen, ob das neue *canvas* Element vom Browser unterstützt wird. Für den Fall daß dieses nicht nicht unterstützt wird, kann eine Ausweichoption definiert werden.

[7] http://modernizr.com

8 Ausblick

Die WHATWG hat Ihre Spezifikation des HTML5 Standards finalisiert und die Browseranbieter haben die Unterstützung der Funktionalitäten teilweise bereits umgesetzt. Auch wenn die finale Spezifikation seitens des W3C noch aussteht, hat dieses ihre Empfehlung gegeben, die bereits umgesetzten Funktionalitäten zu nutzen. [W3C, FAQs, 2012]

Im Zweifelsfalle ist es sicherlich ratsam, sich an der Spezifikation der WHATWG zu orientieren, denn letztendlich ist der Anwender (==Auftraggeber) an einem funktionierenden Resultat interessiert, und seine Schnittstelle zum Internet ist i.d.R. ein Browser.

Es sei aber betont, daß der W3C Standard eine einheitliche Funktionalität beim Zugriff über HTML anstrebt, sprich browserunabhängig, dies ist sowohl im Sinne der Anwender wie auch der Entwickler der Webseiten.

Die meisten neuen Funktionalitäten bieten direkt oder indirekt Fallbackstrategien. Dies ermöglicht es Anbietern von Webinhalten, einem Anwender den für seine Umgebung bestmöglichen Zugriff auf diese Inhalte zur Verfügung zu stellen, sei dieser mit modernen oder älteren, mit auf stationären oder mobilen Endgeräten ausgerichteten Browsern unterwegs.

9 Literaturverzeichnis

golem.de. (kein Datum). Abgerufen am 30. April 2013 von http://www.golem.de/specials/html5/

Grosland, L. (30. Juni 2012). *Grundlagen von HTML5.* Abgerufen am 30. April 2013 von http://channel9.msdn.com/Blogs/Lori/Grundlagen-von-HTML5

Irish, P. (kein Datum). *polyfill*. Von http://paulirish.com/i/7570.png abgerufen

Lauer, B. (Januar 2013). Fünf für jede Domain. *dotnetpro*, S. 22-26.

Lorenz, P. A. (5 2012). Kochen mit Lorenz. *dotnetpro*.

Marsman, J. (1. August 2011). *HTML5 Part 1: Semantic Markup and Page Layout.* Abgerufen am 30. April 2013 von http://blogs.msdn.com/cfs-file.ashx/__key/communityserver-blogs-components-weblogfiles/00-00-00-91-03-metablogapi/5086.HTML5PageLayout_5F00_2.jpg

Marsman, J. (2. August 2011). *HTML5 Part 2: Canvas*. Abgerufen am 30. April 2013 von http://blogs.msdn.com/b/jennifer/archive/2011/08/02/html5-part-2-canvas.aspx

Microsoft. (2013). *Codecs: Häufig gestellte Fragen*. Abgerufen am 30. April 2013 von http://windows.microsoft.com/de-DE/windows7/Codecs-frequently-asked-questions#

Satrom, B. (17. Mai 2011). *http://channel9.msdn.com/Events/Speakers/brandon+satrom*. Abgerufen am 30. April 2013 von http://channel9.msdn.com/Events/TechEd/NorthAmerica/2011/DEV343

Sidorencov, R., & Kirchner, M. (Januar 2011). *Offline Webanwendungen unter HTML5*. Abgerufen am 29. April 2013 von http://winfwiki.wi-fom.de/index.php/Offline_WebAnwendungen_unter_HTML5

W3C. (14. Februar 2011). *W3C Confirms May 2011 for HTML5 Last Call, Targets 2014 for HTML5 Standard*. Abgerufen am 30. April 2013 von http://www.w3.org/2011/02/htmlwg-pr.html.en

W3C. (7. Dezember 2012). *FAQs*. Abgerufen am 30. April 2013 von https://www.w3.org/html/wiki/FAQs#When_can_I_use_HTML5.3F

W3C. (9. April 2013). *Web Storage*. Abgerufen am 30. April 2013 von http://www.w3.org/TR/webstorage/

Web Storage Support Test. (kein Datum). Abgerufen am 30. April 2013 von http://dev-test.nemikor.com/web-storage/support-test/

Wenz, C. (20. März 2012). *WebCamp - Durchstarten mit HTML5*. Abgerufen am 30. April 2013 von channel9: http://channel9.msdn.com/Blogs/Lori/WebCamp-Durchstarten-mit-HTML5

WHATWG. (27. Oktober 2009). *HTML5 at Last Call*. Abgerufen am 30. April 2013 von http://blog.whatwg.org/html5-at-last-call

WHATWG. (19. Januar 2011). *The WHATWG Blog*. Abgerufen am 29. April 2013 von http://blog.whatwg.org/html-is-the-new-html5

Zins, P. (8. Dezember 2011). *HTML Canvas*. Abgerufen am 30. April 2013 von http://www.senaeh.de: http://www.senaeh.de/wp-content/demo/html5canvasslideshow2/#6