1 HTML5 Apps

HTML5 ist die fünfte Version des HTML (Hyper Text Markup Language) Standards und ist der Grundbaustein für moderne Web-Applikationen. Der Standard wird vom World Wide Web Consortium oder kurz W3C[1] erarbeitet und alle nahmhaften Browserhersteller sind an der Entwicklung der Sprache beteiligt. Es ist die erste Version von HTML welche die Verarbeitung von multimedialen Inhalten ohne Browserplugins ermöglicht. HTML5 wurde mit dem Gedanken konzipiert, dass damit erstellte Applikationen von jedem Gerät aus bedienbar sind, im Sinne von "write once, run anywhere", wobei der Web-Browser die Laufzeitumgebung darstellt.

HTML5 wird heute als Oberbegriff für eine Vielzahl von Neuerungen verwendet, welche Entwicklern zur Verfügung stehen um Applikationen im Browser zu realisieren, wie sie früher nur als native oder Flash App möglich waren. HTML5 an sich ist nur die Erweiterung der Markup Sprche an sich und der damit verbundene DOM Level 3 API[2]. Die Medien stellen aber auch Erweiterungen der JavaScript API, wie z.B. die User Media API oder Location API und den CSS3 Standard unter den Begriff HTML5.

In diesem Abschnitt wollen wir erläutern, welche der aus der Einleitung beschriebenen Komponenten in HTML5 verfügbar sind. Da sich HTML5 noch in der Enticklung befindet, sind viele Features nicht konsistent in allen Browsern implemeniert. Die meisten Funktionen, welche den Recommended Status noch nicht erreicht haben sind nur mittels sogenannter Vedor Prefixes abrufbar. Dabei müssen Funktionen im JavaScript oder Eingenschaften im CSS mit einem Präfix im Namen aufgerufen werden. Hier ein Bespiel, wie dies in CSS für die border-radius Eigenschaft gemacht werden muss.

```
.css-class {
    -webkit-border-radius: 8px;
    -moz-border-radius: 8px;
    -ms-border-radius: 8px;
    -o-border-radius: 8px;
    border-radius: 8px;
}
```

Um die Beispiele möglichst kompakt zu halten, werden wir fortan auf die Hersteller-Präfixe verzichten und nur im Text darauf hinweisen, wenn sie benötigt werden.

1.1 User Experience

HTML5 ermöglich es, native ähnliche Applikationen zu erstellen. Was die Gestaltung der Benutzeroberfläche anbelangt, so steht es den plattformspezifischen Umgebungen in nichts nach. Ganz im Gegenteil. Mit Hilfe von HTML5, CSS3

[1] http://www.w3.org/Consortium
[2] http://www.w3.org/TR/DOM-Level-3-Core

und JavaScript können komplexe UI-Konzepte oft schneller und kostengünstiger realisiert werden.

Jeodch sollte man den Aufwand nicht unterschätzen. Verschiedene Browser, Bildschirmauflösungen und Gerätetypen erfordern es, dass man eine Vielzahl an Weichen einprogrammiert um den Benutzern überall die gleiche Erfahrung bieten zu können. Dabei muss man oft mit Bugs oder Differenzen bei der Implementierung der neuen Standards innerhalb der einzelnen Browser kämpfen. HTML5 entwickelt sich stätig weiter und nicht jeder Hersteller implementiert die neuen Funktionen im gleichen Tempo und in der gleichen Qualität. Die Anwendung muss so entwickelt werden, dass auf der einen Seite die Funktionen der Geräte neuerer Generation so gut wie möglich eingesetzt werden. Auf der anderen Seite dürfen die Besucher welche ein älteres Gerät besitzen nicht vernachlässigt werden. Auch für sie muss die Anwendung bedienbar bleiben. Deshalb empfiehlt es sich nach dem Prinzip Progressive Enhancement" vorzugehen. In einem ersten Schritt wird eine Kernversion der Anwendung implementiert, welche auf einer möglichst grossen Fülle an Zielplattformen konsistent funktioniert. Anschliessend werden Schritt für Schritt Optimierungen eingearbeitet, um fortgeschritteneren Browsern mit erweiterter Funktionalität eine verbesserte Version der Anwendung darzustellen.

Man findet online einige Plattformen welche einen Überblick bieten, welche Plattformen welche features unterstützen. Eine der umfangreichsten ist ≪Can I use. . . ≫[3].

1.2 Performance

Auch was die Performance anbelangt, bietet einem HTML5 ein Fülle an möglichkeiten um die Anwendung zu optimieren. Hier wollen wir einige der wichtigsten vorstellen.

1.2.1 CSS Transitionen / Animationen

CSS Transitionen, welche ein Teil der CSS3 Spezifikation sind, erlauben es Änderungen an der Benutzeroberfläche zu animieren. Dabei kann man sehr detailliert Steuern, wie schnell und in welcher Art die Übergänge stattfinden. Wenn man z.B. bisher in CSS2 die Farbe eines Elementes von Weiss nach Schwarz geändert hat, so war der Übergang unmittelbar. Es gab keine Möglichkeit Zwischenstufen zu definieren. Wollte man dies trotzdem Bewerkstelligen, mussten die Animationen mittels JavaScript bewerkstelligt werden. Mit CSS3 ist dies nun direkt mit CSS möglich. Dabei kann man für jede animierbare Eigenschaft seperat defineiren ob und in welcher weise sie bei Änderungen animiert werden soll.

```
1 .color-transition {
2     transition-property: width, height, color;
3     transition-duration: 1s, 1s, 2s;
4     transition-timing-function: linear, linear, ease-out;
5 }
```

[3] http://caniuse.com

Im obigen Beispiel würden bei allen Elementen welche die Klasse color-transition haben Änderungen an der Breite, Höhe und Farbe animiert werden. Alle gängigen Browser erfordern einen Präfix um die Funktion zu nutzen. Gemäss Spezifikation sollten alle Transitionen Hardware-Beschleunigt werden, d.h. auf der GPU berechnet werden. Genau hier liegt der enorme Performance-Vorteil gegenüber den bisherigen JavaScript Animationen. Detailierte Informationen zu den Transitionsfunktionen können der Spezifikation entnommen werden[4].

1.2.2 Web Workers Bisher gab es keine Möglichkeit im Browser Berechnungen im Hintergrund, geschweige denn parallel auszuführen. Dies hatte oft zur Folge, das die Benutzeroberfläche nicht bedienbar war, bis eine Aktion ausgeführt wurde. Diese Lücke wurde nun mit Hilfe von Web Workers geschlossen. Diese ermöglichen es Prozesse in Hintergrund-Threads auszuführen.

Wie der Name schon sagt, setzt dieses Feature auf dem Actor-Model auf[5]. Die Worker agieren in einem eigenen Kontext haben aber auch Zugriff auf den Globalen Scope. Viele der Herausforderungen welche man aus der Multithread-Programmierung von Java oder C++ her kennt trifft man auch hier an. Jedoch ist die API so konzipiert, dass sie einem möglichst um die meisten Probleme herumleitet. So ist der Zugriff auf nicht threadsichere Komponenten, wie z.B. das DOM[6], nicht möglich und Daten müssen mittels serialisierten Objekten ausgetauscht werden. Hält man sich an die vorgegebene API so ist es sehr einfach Konflikte zu vermeiden.

Initialisierung eienes Workers:

```
1 var myWorker = new Worker("worker-code.js");
2
3 myWorker.addEventListener("message", function (event) {
4   console.log("Daten empfangen!\n");
5 }, false);
6
7 myWorker.postMessage(""); // start the worker.
```

Datenaustauch mit einems worker:

```
1 myWorker.postMessage("Dies sind einige Daten");
```

Mittels postMessage() werden Daten an den Worker geschickt. Erhält ein Worker Daten, so wird das message-Event ausgelöst. Mit einem entsprechenden Eventhandler kann man die empfangenen Daten verarbeiten.

1.2.3 Web Sockets Neben der Möglichkeit zur Parallelisierung war die bidirektionale Kommunikation eine der grössten Schwächen von Web Applikationen. Man konnte diese fehlende Funktionalität mithilfe von Flash oder einem anderen Browserplugin ergänzen, jedoch war dies immer ein Steiniger weg. Web Sockets

[4] http://dev.w3.org/csswg/css-transitions
[5] http://en.wikipedia.org/wiki/Actor_model
[6] http://en.wikipedia.org/wiki/Document_Object_Model

bilden die Grundlage dafür, diese Funktion ohne etwaige Plugins zu realisieren. Dabei wird einem ermöglicht, persistente Verbindungen mit dem Server aufzubauen und Daten auszutauschen ohne eine Polling-Funktion auf der Clientseite implementieren zu müssen.

Aufbau einer Verbindung:

```
var exampleSocket = new WebSocket(
    "ws://www.example.com",
    "protocol"
);
```

Versand von Daten:

```
exampleSocket.addEventListener("open", function (event) {
  exampleSocket.send("Das sind einige Daten.");
});
```

Empfang von Daten:

```
exampleSocket.addEventListener("message", function (event) {
  exampleSocket.send("Daten empfangen: " + event.data);
});
```

Als erstes muss ein Socket instanziert werden. Die Angabe des Protokolls ist optional, ermöglicht es einem aber mehrere Endpunkte unter der gleichen URL zu betreiben oder je nach Protokoll das Format für den Datenaustausch anzupassen. Ist dann der Socket geöffnet, so wird das open-Event ausgelöst. Ab diesem Zeitpunkt können Daten ausgetauscht werden. Werden Daten vom Server empfangen, so wird wie bei einem Worker das message-Event ausgelöst. Es können nicht nur wie im Beispiel Strings ausgetauscht werden sondern auch komplexe Objekte.

Neben Web Sockets wird ein weiterer Standard WebRTC (Web Real Time Communication) entwickelt. Dieser kommen dort zum Einsatz, wo konstant Daten gestreamt werden müssen. Einsatzbeispiele sind Liveticker so wie man sie von News- oder Börsen-Apps kennt oder auch Anwendungen für Videotelefonie. Anwendungen wie Skype könnten komplett ohne Hilfe von Browserplugins als Web Applikation umgesetzt werden. Da WebRTC auch Peer-To-Peer Verbindungen erlaubt, wird nicht zwingend ein Server für den Datenaustausch benötigt. Lediglich zum Aufbau der Verbindung zwischen den einzelnen Peers wird er vorrausgesetzt. WebRTC befindet sich jedoch noch in einem relativ frühen Stadium und wird auf mobilen Plattformen bisher nur sporadisch unterstützt. Es wird aber erwartet, dass mit den nächsten grösseren Updates die meisten Hersteller diese Funktion nachrüsten werden.

Dies sind nur drei von vielzähligen neuen performance-kritischen APIs welche HTML5 mit sich bringt. Weitere sind unter anderem App cache, welcher einem ermöglicht Web Apps auch offline zu nutzen. Indexed database und Local Storage welche clientseitige Datenbankfunktionalitäten bringen oder auch WebGL, welches einen Grossteil der Funktionen von OpenGL auch im Browser verfügbar macht. Damit lassen sich auch aufwendige Videospiele im Browser betreiben.

Weiterführende Resourcen zu allen neuen APIs können der offiziellen Seite des W3C entnommen werden. Das Mozilla Developer Network[7] ist ebenfalls eine gute Anlaufstelle, da es eine Fülle an Codebeispielen und ausführlichen Dokumentationen bietet. Zwar sind einige Beispiele spezifisch für den Firefox geschrieben, jedoch lassen sie sich mit geringem aufwand auch für andere Browser Portieren.

1.3 Zugriff auf Gerätefunktionen

Der direkte Zugriff auf Gerätefunktionen stellt eine der grössten Schwächen von HTML5 dar. Die Hersteller führen teilweise mehrmals pro Jahr neue API-Endpunkte ein. Die Ausarbeitung von neuen Spezifikationen durch was W3C nimmt of Jahre in Anspruch und kann demzufolge nicht mit der Geschwindigkeit der Hardwarehersteller mithalten. Dies ist auch verständlich, da beim W3C sich diverse Organisationen auf einen gemeinsammen Nenner einigen müssen. Bei den mobilen Endgeräten haben die Herstellen freie Hand und können neue Schnittstellen einführen ohne auf andere Instanzen angewiesen zu sein. Trotzdem bietet HTML5 schon heute Zugriff auf eineige wichtige Gerätefunktionen.

1.3.1 GPS Der Zugriff auf die aktuelle Position des Benutzers kann mittels der Geolocation API[8] realisiert werden und wird von allen gängigen Plattformen unterstützt. Dabei gibt es zwei Möglichkeiten für die Positionsermittlung. Man kann die Position einmalig abfragen oder konstant in einem gewissen Interval. In jedem Fall muss der Benutzer zustimmen, dass seine Position ermittelt werden darf.

Einmaliges Abfragen der Position:

```
navigator.geolocation.getCurrentPosition(function(position) {
    showPosition( //user defined function
        position.coords.latitude,
        position.coords.longitude
    ),
    ErrorHandler,
    options
});
```

Konstantes Abfragen der Position:

```
navigator.geolocation.watchPosition(function(position) {
    showPosition( //user defined function
        position.coords.latitude,
        position.coords.longitude
    ),
    ErrorHandler,
    options
});
```

[7] https://developer.mozilla.org
[8] http://dev.w3.org/geo/api

Der ErrorHandler ist ein optionaler Parameter welcher aufgerufen wird, wenn der Benutzer nicht will, dass seine Position bestimmt wird oder wenn die Abfrage an sich fehlschlägt. Im dritten Parameter options kann angegeben werden wie die Informationen abfegragt werden (Genauigkeit, Timeout und Caching-Verhalten). Zu beachten ist, dass die Abfrage der Position auf den meisten Geräten sehr ressourcenintensiv ist und die Akkulaufzeit stark verringert.

1.3.2 Gyroskop / Beschleunigungssensor Die aktuelle Ausrichtung des Gerätes welche das Gyroskop misst kann mittels des deviceorientation-Events[9] ermittelt werden und wird von allen gängigen Plattformen unterstützt.

```
window.addEventListener("deviceorientation", function(event) {
    /*
     * process event.alpha, event.beta,
     * event.gamma and event.absolute
     */
}, true);
```

Die drei Werte alpha, beta und gamma wie sich das Koordinatensystem des Gerätes im Vergleich zu einem fix definierten globalen Koordinatensystem ausrichtet (Abb. 1). Der letzte Wert absolute gibt an, ob das Gerät in der Lage war, die Ausrichtung relativ zu einem globalen Koordinatensystem zu bestimmen.

Der Zugriff auf den Beschleunigungssensor wird zur Zeit nur von der iOS Plattform unterstützt. Das dazugehörige Event ist devicemotion. Welche Daten verfügbar sind kann der Spezifikation[10] entnommen werden.

1.3.3 Audio- / Videoinput Der Audio- / Videoinput ist bei HTML5-Apps nur eingeschränkt möglich. Die Media Capture API[11] sieht zwar solche möglichkeiten vor, jedoch ist sie auf den mobilen Plattformen noch nicht umgesetzt worden. Seit kurzem kann auf den aktuellsten versionen von Android und iOS mit dem capture attribut auf einem input-Tag angeben, dass der Benutzer ein Bild mit der Kamera aufnehmen können soll. Die einzige Alternative bietet der normale Fileupload. Hierzu muss der Benutzer jedoch die medialen Inhalte bereits zuvor aufgenommen haben um sie dann hochladen zu können.

```
<input type="file" accept="image/*" capture="camera">
```

1.4 Einnahmemöglichkeiten

Leider gibt es für HTML5 Apps keine Stores wie Google Play oder Apples App Store. Dies erschwert es eine breite Masse zu erreichen. Die ergiebisgte Einnahmequelle für HTML5 Apps sind nach wie vor Werbeflächen. Der Gedanke für Web Anwendungen zu bezahlen hat sich beim grossteil der Anwender immer

[9] http://dev.w3.org/geo/api/spec-source-orientation.html
[10] http://dev.w3.org/geo/api/spec-source-orientation.html#devicemotion
[11] http://www.w3.org/TR/html-media-capture

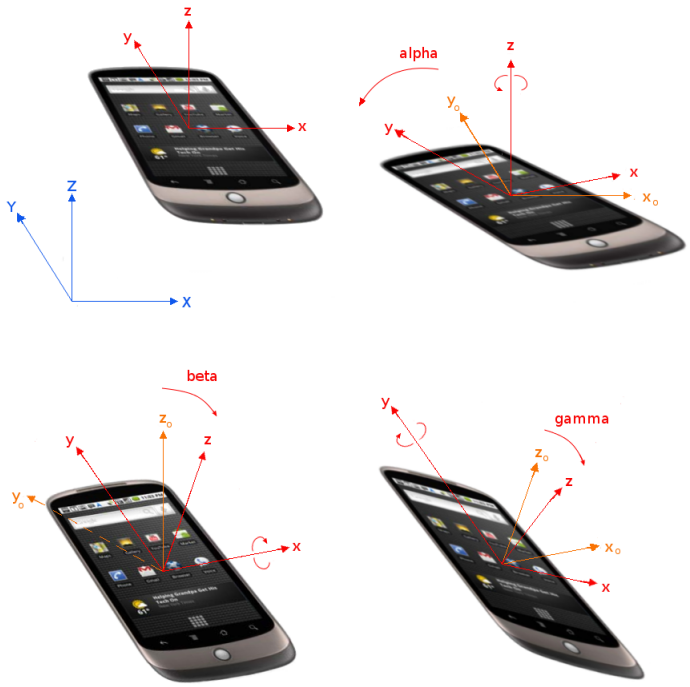

Abb. 1. Ausrichtung des Gerätes, Quelle: http://dev.w3.org/geo/api/spec-source-orientation.html

noch nicht durchgesetzt, weshalb es Paid-Only-Apps nach wie vor sehr schwer haben.

Der Weg HTML5 hat jedoch auch seine Vorteile. Eine HTML5 Seite kann durch eine Suchmaschine indexiert werden, was mit einer Native App nicht möglich ist. Am Beispiel eines Onlineshops kann dieser mit einer Suchanfrage nach einem Produkt gefunden werden. Dies ist ein wichtiger Faktor im Marketing und erhöht die allgemeine Präsenz im Web. Daher macht es in solchen Fällen wenig Sinn, auf native Apps zu setzen.

1.5 Cross Plattform Entwicklung

Da HTML5 als offener Webstandard entwickelt wird, kann theoretisch jeder diesen implementieren. HTML5 ist somit das Entwicklungstool und ein Webbrowser die Laufzeitumgebung. Da es Webbrowser für alle gängigen Desktop- und Mobilgeräte gibt, kann eine HTML5 Applikation für eine breite Masse an Geräten entwickelt werden.

In der Praxis ist es allerdings nicht ganz trivial, da auf der einen Seite HTML5 ein dynamischer Standard ist, der ständig weiterentwickelt wird, und auf der anderen Seite jeder Browser einen unterschiedlichen Implementationsstand aufweist. Dadurch muss gegebenenfalls trotz Plattformunabhängigkeit auf Features geprüft und Browserweichen eingebaut werden.

1.6 Fragmentierung

Für lange Zeit war die Fragmentierung der mobilen Browser sehr gering, da auch die Modelvielfallt sich in Grenzen hielt. Jedoch ist die Zahl an Browser- und Betriebssystemversionen in den vergangenen Jahren explodiert. Vor allem bei Android, wo viele Geräte nur sporadisch Softwareupdates erhalten, ist die Streuung gross. Dies macht den Ansatz «write once, run anywhere» den die HTML5-Gemeinschaft zu Begin verfolgt hat kaum noch haltbar. Das heutige Kredo lautet eher «write once, optimize everywhere». Nichts desto trotz ist der Optimierungsaufwand um die gängisten Geräte zu unterstützen im vergleich zu Native Apps immer noch geringer. Oft sind es nur kleinere Ungereimtheiten im Layout oder leichte Unterschiede in der API welche leicht korrigiert werden können. Viele Frameworks decken den Grossteil dieser Abweichung bereits ab.

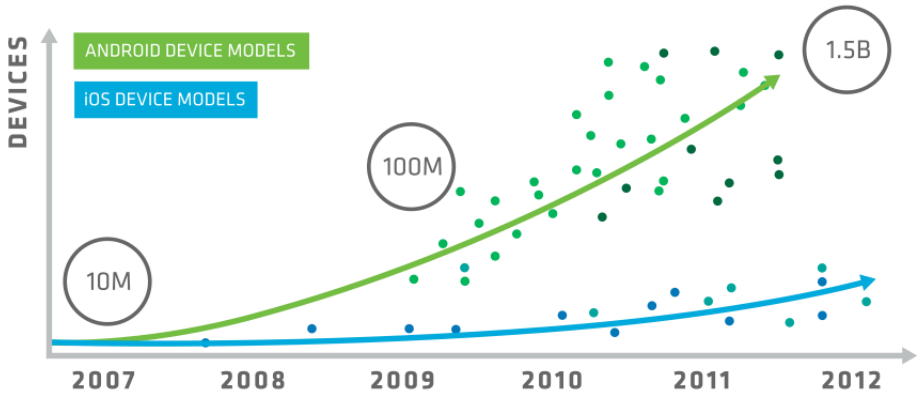

Abb. 2. Streuung von Android und iOS Geräten, Quelle: ComScore.

1.7 Verfügbarkeit von Entwicklerresourcen

Die Verfügbarkeit von Entwicklerressourcen ist ein grosser Vorteil von HTML5. Es gibt eine Vielzahl an Webentwicklern, die mit ihren gewohnten Entwicklungstools weiterarbeiten können und für die App-Entwicklung nicht extra eine neue Programmiersprache lernen müssen. Dadurch ist es relativ einfach auf

dem Arbeitsmarkt Fachpersonal zu finden. Auch Neueinsteiger haben durch eine gute Auswahl an Büchern und Ressourcen im Internet eine relativ kleine Einstiegshürde.

Ein weiterer Punkt ist das Angebot von Drittherstellern. Da es HTML, CSS und JavaScript schon sehr lange gibt, ist ein sehr grosses Angebot an Frameworks, Tools, Code-Beispielen und Dokumentationen vorhanden.

1.8 Distribution

Im Gegensatz zu Native Apps, ist man bei HTML5 Anwendungen von den Stores der einzelnen Hersteller unabhängig. Da die Anwendungen meist diekt auf den eigenen Server betrieben werden, können Updates ohne Umwege und Verzögerungen eingespielt werden. Auf Benutzerseite ist lediglich ein reload der Applikation nötig. So kann man viel schneller auf Fehler in der Software oder Kundenwüsche reagieren.

Die unabhängigkeit von den App Stores hat aber auch eine grossen Nachteil. Es ist schwer den Konsumenten zu erreichen. Es konnte dich noch kein Store für Web-Apps etablieren. Ansätze von Mozilla[12] und Facebook[13] existieren, jedoch ist der Bekannstheitsgrad vom Mozilla Marketplace noch eher gering und derjenige von Facebook ist nicht für mobile Apps ausgelegt.

Weiterhin sind Benutzer es sich nicht gewohnt, Applikationen innerhalb des Browsers zu betreiben. Auf jeder Plattform gibt es zwar die Möglichkeit ein Bookmark zur App auf dem Homescreen abzulegen und diese dann bequem von dort zu starten. Leider ist diese Funktion den meisten Anwendern kaum bekannt.

[12] https://marketplace.firefox.com
[13] https://www.facebook.com/appcenter