Jens-Christian Fischer

(jens-christian.fischer@simplificator.com)
entwickelt seit über 20 Jahren Software und schreibt Artikel und Bücher. Er beschäftigt sich seit mehreren Jahren mit mobilen Geräten und bietet auch entsprechende Schulungen an. Wenn er nicht vor dem Computer sitzt, was laut der Aussage seiner Familie viel zu oft vorkommt, betreibt er Aikido.

Webseiten in Bewegung – Wunsch und Wirklichkeit

Mobile Geräte überall! Und damit die Anforderung, dass „man" möglichst schnell und sofort auf den mobilen Geräten präsent ist. Das Problem ist hierbei die Vielfalt der Geräte: Native Anwendungsentwicklung ist nur in den seltensten Fällen für mehrere Geräteklassen möglich. Mithilfe von Webtechnologien wie HTML5, JavaScript und einigen Bibliotheken ist es aber möglich, mit vertretbarem Aufwand Anwendungen für eine Reihe von Geräten zu entwickeln. Dieser Artikel zeigt das Spektrum der Möglichkeiten auf und gibt Hinweise, wann welche Technologie einzusetzen ist.

Die Website in Bewegung bringen

Darüber, dass es heutzutage nicht mehr ohne (Web-)Inhalte auf mobilen Geräten geht, muss man nicht mehr reden. Ständig steigende Verkaufszahlen von Smartphones sowie steigende Zugriffszahlen von mobilen Geräten sind inzwischen völlig normal. In nicht allzu ferner Zukunft werden Zugriffe von mobilen Geräten die Norm und nicht die Ausnahme sein. Für die Anbieter von Inhalten stellt sich die Frage, wie sie die Inhalte am einfachsten und schnellsten auf den mobilen Geräten bereitstellen.

Es gibt ein breites Spektrum von Möglichkeiten – und je nach dem zur Verfügung stehenden Budget (sowohl was Zeit als auch Geld angeht) bieten sich einfache oder komplexere Lösungen an. Der „Goldstandard" ist sicher eine native Applikation, die mit dem vom Hersteller gestellten Software Development Kit (SDK) entickelt wird. Nur hier kann man auf alle Feinheiten der jeweiligen mobilen Plattform zugreifen.

Leider ist genau diese Variante die aufwendigste und teuerste, müssen doch sehr unterschiedliche Konzepte und Programmierumgebungen verstanden und bedient werden. Von Objective-C auf iOS bis zu Java unter Android oder C# für die Windows Phones reicht die Palette (und berücksichtigt noch gar nicht die eher exotischen Plattformen wie Bada von Samsung).

Was allen diesen modernen mobilen Plattformen aber gemein ist, ist eine gute bis sehr gute Unterstützung von Webinhalten. Es liegt daher nahe, zu überlegen, ob nicht eine HTML-Lösung der richtige Weg für eine zeitnahe Umsetzung ist. Und tatsächlich: Diese Variante eignet sich für eine ganze Reihe von Websites und mobilen Anwendungen. Das Spektrum reicht dabei von auf Mobilgeräte angepassten Webseiten, über Anwendungen, die sich im Browser bedienen lassen, bis hin zu paketierten Anwendungen, die auch auf die verschiedenen Sensoren und Funktionen der Geräte zugreifen können, und gar bis zu Anwendungen, die mit nativen UI-Elementen ausgestattet sind und sich nicht von „richtigen" Anwendungen unterscheiden lassen.

Nachfolgend beleuchten wir der Reihe nach (und in steigender Komplexität) die verschiedenen Möglichkeiten und geben Hinweise, für welchen Verwendungszweck sich welche Art eignet.

HTML goes mobile

Die aktuellen mobilen Geräte verfügen im Allgemeinen über ausgezeichnete Browser, die in der Regel auf Webkit basieren. Webkit verrichtet auch in Apples Safari und Googles Chrome seine Dienste. Das heißt, dass die meisten Webseiten auch „einfach so" in guter Qualität dargestellt werden. Mit HTML 5 sowie CSS 3 ist es mit wenig Aufwand möglich, die Darstellung der Website auf mobilen Geräten zu verbessern.

Responsive Design

Als Erstes bietet es sich an, das Layout auf die kleineren Bildschirme der Telefone und Tablets abzustimmen. Natürlich kann man eine spezielle mobile Version der Seite anbieten – zum Beispiel unter einer eigenen Domäne wie „mobile.firma.de". Wesentlich flexibler ist man aber, wenn man sogenanntes „Responsive Design" [ali] nutzt. Dabei werden verschiedene CSS-Regeln definiert, die zum Beispiel je nach der Bildschirmbreite des Clients angewendet werden. So kann man eine einzige Version der Website veröffentlichen, einmal das HTML definieren und verschiedene CSS-Formate verwenden. Die Unterscheidung zwischen den verschiedenen CSS-Stilen geschieht mit den Media Queries:

```
@media screen and (max-width:1024px) {
  header { position:static; }
  body { padding-left: 24px; }
}

@media screen and (max-width:768px) {
  body { padding-left: 0; }
}
```

Der „max-width"-Parameter gibt an, für welche Seitenbreiten die folgenden Stile gelten sollen. Weitere Beispiele für Webseiten mit Responsive Design finden Sie unter [net], zusammengestellt von Ethan Marcotte, der den Begriff in seinem Artikel [ali] das erste Mal benutzte.

Offline Modus

Da mobile Geräte auch gerne mal „offline" sind, wäre es eigentlich ganz schön praktisch, wenn eine Website auch dann zugänglich ist, wenn gerade keine Netzverbindung besteht. HTML5 kennt dazu einerseits das Cache Manifest, andererseits Local Storage. Mit dem Cache Manifest legt der Webdesigner fest, welche Dateien von einem mobilen Client auf dem Gerät gespeichert werden sollen. Die entsprechende Website ist dann auch verfügbar, wenn weit und breit kein WLAN oder Funkmast in Reichweite ist. Das Cache Manifest ist eine Textdatei, die der Reihe nach festlegt, welche Dateien „ge-cachet" werden sollen und welche nicht. Wie ein solches Manifest aussieht, steht zum Beispiel in [htm].

Sobald eine Website mehr als nur statische Inhalte darstellt, ist es nötig, Daten zu speichern. Im Normalfall sind die Daten ja auf dem Server gespeichert und werden bei Bedarf geladen. Offline ist das allerdings „etwas schwierig". Jetzt sollten die Daten möglichst lokal gespeichert werden. HTML 5 kennt dazu „Local Storage", ein Key-Value Store, der sich mit JavaScript ansprechen lässt:

```
if (Modernizr.localstorage) {
  var foo = localStorage.getItem("key");
  // ...
  localStorage.setItem("key", foo);
} else {
  alert('Keine Offline Speicherung möglich');
}
```

In diesem Beispiel wird erst via der Bibliothek „modernizr" [mod] ermittelt, ob Local Storage vom Browser unterstützt wird, und dann erst ein Item gelesen und geschrieben. Local Storage ist sehr einfach gehalten: Man kann Strings laden und speichern. Es gibt Browser, die komplexere Datentypen unterstützen, allerdings sind diese nicht in allen Browsern verfügbar.

Ein gelungenes Beispiel für eine reine Webanwendung, die alle oben genannten Möglichkeiten nutzt, ist die App der Financial Times [app]. Auf dem iPhone und dem iPad ermöglicht sie das Offline Lesen der kompletten Zeitung.

jQuery Mobile

Das wohl bekannteste JavaScript-Framework für das Erstellen von „dynamischen" Webseiten ist jQuery [jqu]. Das bekannteste Framework für das Erstellen von Webanwendungen für mobile Endgeräte ist jQuery Mobile [jqm].

Das ist insofern interessant, als die beiden eigentlich gar nichts miteinander zu tun haben. jQueryMobile (ab jetzt kurz: jQM) nutzt zwar jQuery intern, hat von den Konzepten, die es verwendet aber rein gar nichts damit zu tun. Böse Zungen behaupten, es bestehe ein ähnlicher Zusammenhang zwischen den beiden Frameworks wie zwischen Java und JavaScript. Der Name JavaScript war eine reine Marketingmaßnahme, wie Brendan Eich, der Entwickler von JavaScript (oder LiveScript wie es anfangs hieß), freimütig zugibt [web]. Netscape wollte, dass sich JavaScript eine Scheibe vom damals populären Java abschneiden konnte.

Um eine Anwendung mit jQM zu schreiben, arbeitet man zuerst mit HTML. Dieses HTML, dass einige Besonderheiten aufweist, wird von jQM neu formatiert und anders dargestellt, damit es vom Aussehen her so scheint, als betrachte man nicht eine Webseite, sondern eine mobiles UI. Dazu verwendet jQM ganz klassisch CSS und Bilder.

Eine Anwendung in jQM besteht aus mehreren Seiten, genannt Pages. Jede Page besteht ihrerseits aus verschiedenen Komponenten – etwa einer Kopf- oder Fußzeile (Header, Footer) sowie dem eigentlichen Inhalt. In diesen Komponenten können wiederum Elemente wie Knöpfe (Button) oder Listen (ListView) eingebettet sein.

Die Definition dieser verschiedenen Elemente ist durch eine Reihe verschachtelter HTML-Auszeichnungen (DIV-Elemente) gelöst. Damit jQM weiß, welches Element welche Funktion (und damit welches Aussehen) hat, bekommt jedes Element eine Attribut, das die Rolle festlegt. Da HTML an sich so etwas nicht direkt vorsieht, nutzt jQM das mit HTML5 eingeführte „data"-Attribut. Jeder HTML Tag kann einen oder mehrere data-Attribute enthalten. Diese werden über ein Suffix voneinander unterschieden. Das folgende Beispiel zeigt ein DIV-Element, das die Rolle „page" hat:

```
<div data-role="page"> ... </div>
```

Eine etwas komplexere Seite sieht dann im HTML-Code zum Beispiel so aus:

Listing 1

```
<div data-role="page" id="formPage">
  <div data-role="header">
    <a data-icon="back" data-rel="back"
      data-direction="reverse">Cancel</a>
    <h1>Update Task</h1>
    <a id="saveButton" data-icon="check"
      data-direction="reverse">Save</a>
  </div>
  <div data-role="content">
    <form action="">
      <ul data-role="listview">
        <li data-role="list-divider">Task Details:</li>
        <li data-role="fieldcontain">
          <label for="taskName">Name:</label>
          <input type="text" name="taskName"
          id="taskName" value="">
        </li>

        // ...
        <li data-role="fieldcontain">
          <label
for="taskCompleted">Completed:</label>
          <select name="taskCompleted"
id="taskCompleted"
            data-role="slider">
            <option value="no">No</option>
            <option value="yes">Yes</option>
          </select>
        </li>
        // ...
      </ul>
    </form>
  </div>
</div>
```

Ein HTML-Dokument darf durchaus mehrere solcher Pages haben. Ein Verweis von einer Seite zur nächsten wird dann von jQM animiert. Lädt man hingegen eine andere HTML-Datei, ändert jQM den Aufruf zu einem asynchronen (AJAX) Serveraufruf und fügt das resultierende Element in das angezeigte Dokument ein. Für den Benutzer stellt sich das dann als nahtloser Übergang zwischen den verschiedenen Seiten dar.

Um die Anwendung mit Funktionalität zum Leben zu erwecken, nutzt man dann ganz normales JavaScript mit jQuery, wie man es bei normaler Webentwicklung verwendet.

Für einen versierten Webentwickler ist es einfach, sich in jQuery Mobile einzuarbeiten, und nach wenigen Stunden sind Resultate sichtbar. Sobald man die ver-

schiedenen Rollen, die man in jQM den Seitenelementen zuweisen kann, und ihre Parameter kennt, lassen sich schnell Mobilanwendungen bauen.

jQM unterstützt beinahe alle Smartphones (auch solche, die man nur mit viel gutem Willen als „Smart“ bezeichnen würde) und passt sich in Darstellung und Funktionalität an das jeweilige Gerät an. So sind iOS- und Android-Geräte sehr gut unterstützt, während alte Windows Mobile oder Blackberry-Geräte in Dingen wie Animationen oder AJAX-Aufrufen zurückstecken müssen. Grundsätzlich funktionieren die Anwendungen aber auch auf diesen Geräten.

jQueryMobile ist Open Source und frei verfügbar. Eine große und aktive Entwicklergemeinde sorgt für regelmäßige Updates und Support.

Sencha Touch

Einen gänzlich anderen Weg geht Sencha Touch. Dieses Framework ist aus dem JavaScript Framework Ext entstanden, das für das Erstellen von Internetanwendungen entwickelt wurde. Sencha Touch (ab jetzt ST) ist die „mobile Variante” davon. Es handelt sich um ein komplexes Model-View-Controller-Framework, das es erlaubt, in JavaScript komplexe Anwendungen für mobile Geräte zu schreiben. Es stellt eine ganze Reihe von Komponenten zur Verfügung, die miteinander verdrahtet werden.

HTML findet man in einer SenchaTouch-Anwendung praktisch keines: Ein leeres Body-Element ist alles. Das ganze Aussehen und die Funktionalität werden programmiert. Das ist auf den ersten Blick abschreckend und durchaus mit einer steilen Lernkurve verbunden. Dafür bekommt man ein durchdachtes Framework, das es erlaubt, auch komplexere Anwendungen zu schreiben.

Eine Sencha-Anwendung wird aus ineinander verschachtelten Komponenten gebaut. Listing 2 etwa zeigt ein Aufgabenformular für eine Todo-Listen-Anwendung, wie wir sie grade schon für jQM betrachtet haben:

Listing 2

```
Ext.define('ToDoListApp.view.TaskForm', {
    extend: 'Ext.form.Panel',

    config: {
        id: 'taskForm',
        items: [{
            xtype: 'toolbar',
            docked: 'top',
            title: 'Task',
            ui: 'light',
            items: [{
                xtype: 'button',
                text: 'Back',
                ui: 'back',
                action: 'cancel'
            },
            {
                xtype: 'spacer'
            },
            {
                xtype: 'button',
                text: 'Save',
                action: 'saveTask'
            }]
        },
        {
            xtype: 'fieldset',
            id: 'mainFieldset',
            instructions: 'Enter the details of the task',
            title: 'Task Details',
            items: [{
                xtype: 'textfield',
                id: 'titleField',
                label: 'Title',
                name: 'title',
                autoCapitalize: true,
                placeHolder: 'Enter a title'
            },
            // ...
            {
                xtype: 'datepickerfield',
                id: 'dateField',
                label: 'Due on',
                name: 'dueDate',
                placeHolder: 'dd/mm/yyyy',
                dateFormat: 'D d MY',
                picker: {
                    slotOrder: ['day', 'month', 'year'],
                    yearFrom: (new Date().getFullYear()),
                    yearTo: (new Date().getFullYear()) + 10
                }
            },
            // ...
            ]
        }]
    }
});
```

Um den „Save”-Knopf mit Leben zu erwecken, benötigt man einen Controller, der diesen Knopf kennt (via dessen ID) und eine entsprechende Funktion bereitstellt:

Listing 3

```
Ext.define('ToDoListApp.controller.TaskController', {
    extend: 'Ext.app.Controller',

    config: {
        id: 'taskController',
        refs: {
            saveButton: 'button[action=saveTask]',
            // ...
        },
        control: {
            saveButton: {
                tap: 'saveTask'
            },
            // ...
        }
    },

    // ...

    saveTask: function (button, e, eOpts) {
        var store = this.getTaskList().getStore();
        var task = this.getTaskForm().getRecord();
        this.getTaskForm().updateRecord(task);

        // Is it a new object?
        if (null === store.findRecord("id", task.get('id'))) {
            store.add(task);
        }

        this.showList();
    },

    // ...
});
```

So viel Code mag abschreckend wirken – und bis man sich in das Framework eingearbeitet hat und versteht, welches Element wann wie mit welchem anderen kombiniert werden muss, um die gewünschte Wirkung zu zeigen, dauert es einen Moment. Dann aber hat man ein Werkzeug in der Hand, das es ermöglicht, deutlich umfangreichere Anwendungen zu schreiben.

Secha Touch dürfte einem Softwareingenieur besser liegen als einem Webdesigner. Im Gegensatz zu jQuery Mobile funktioniert Sencha Touch momentan nur auf Browsern, die auf Webkit aufsetzen – namentlich Android und iOS-Geräte. Sencha arbeitet an der Unterstützung von Windows Phone 7.

Sencha ist ein kommerzielles Produkt, das allerdings in einer freien Version verfügbar ist. Die Firma hinter Sencha bietet Schulung, Support und kommerzielle Lizenzen an.

Abb.: Zweimal dieselbe Anwendung: Links in jQueryMobile, rechts in SenchaTouch realisiert

Die Abbildung zeigt die Eingabemasken, die einmal mit jQueryMobile (links) und einmal mit Sencha Touch (rechts) realisiert wurden.

Titanium Appcelerator

Wem die webbasierten Anwendungen zu wenig „echt" aussehen, der muss wohl oder übel zu XCode und dem Android SDK greifen, oder? Nein, denn die Firma Appcelerator hat mit dem Produkt „Titanium" [apc] ein „Zwischending" im Angebot: Mit JavaScript programmieren, aber wirklich native Anwendungen erstellen.

Möglich gemacht durch eine Software-Brücke, die das Betriebssystem von iOS, Android oder neuerdings BlackBerry 10 mit JavaScript verbindet. Das erlaubt es, die nativen Elemente der jeweiligen Plattform mittels JavaScript zu erzeugen und zu bedienen. Das verwendete JavaScript kann zu einem gewissen Teil zwischen den Plattformen verwendet werden – teilweise müssen User-Interface-Spezifika für die jeweilige Plattform separate geschrieben werden. Allgemeine Logik und Steuerung jedoch braucht es nur einmal. Damit hat man zwar nicht „write once, run everywhere" erreicht, ist dem aber einen rechten Schritt näher gekommen.

Seit Kurzem stellt der Hersteller auch verschiedene Cloud Services zur Verfügung, die sich in mobile Anwendungen integrieren lassen. Das reicht von der Verwaltung von Fotos in der Cloud, Chat mit anderen Anwendern bis zu Statusmeldungen und vielem mehr. Mit diesen Cloud Services sind Anwendungen möglich, die eine breite Palette von Interaktionen zwischen verschiedenen Benutzern ermöglichen. Es ist durchaus möglich, Funktionen von sozialen Netzwerken mit einer Appcelerator-Lösung zu bauen.

Titanium ist ein kommerzielles Produkt, liegt aber zu großen Teilen im Quellcode offen. Entwickler können vieles mit der frei verfügbaren Version entwickeln, für verschiedene Zusatzmodule, Datenanalyse oder viel benutzte Cloud-APIs werden monatliche Gebühren fällig. Appcelerator bietet ebenfalls kostenpflichtigen Support und Schulungen an.

Fazit und Empfehlung

Was ist denn jetzt die „richtige" Technologie? Wie Sie in den vorangegangenen Abschnitten sicher gemerkt haben, ist die Antwort hierauf nicht so einfach. Es kommt – wie so oft in der Informatik – darauf an, wie viel Zeit und Geld vorhanden sind. Über welches Know-how verfügen die Entwickler? Müssen Anwendungen in einem der Onlinestores der Hersteller vorhanden sein? Wie viel Zugriff auf native Elemente ist nötig? Auf welchen Geräten soll die Anwendung verfügbar sein? Wie sehr soll die Anwendung wie eine native Anwendung aussehen? Erst die Antworten auf diese Fragen zeigt auf, welche der Technologien die richtige ist.

Klar ist, dass jede ihre Berechtigung hat und für den jeweiligen Zweck angepasst ein gutes Resultat mit vernünftigem Aufwand liefern kann. Die Kompromisse, die man dabei eingehen muss, sind gegen die erwartenden Resultate abzuwägen. Schaut man sich die Webseiten der verschiedenen Projekte an, sieht man durchaus, welche Arten von Projekten sich besonders für die jeweilige Technologie eignet. ■

Links

[ali]: www.alistapart.com/articles/responsive-web-design/
[net]: www.netmagazine.com/features/ethan-marcottes-20-favourite-responsive-sites
[htm]: www.html5rocks.com/en/tutorials/appcache/beginner/
[mod]: http://modernizr.com/
[app]: http://apps.ft.com/ftwebapp/
[jqu]: http://jquery.org/
[jqm]: http://jquerymobile.com/
[web]: http://web.archive.org/web/20070528020012/http://www.mozilla.org/js/language/ICFP-Keynote.ppt
[apc]: http://appcelerator.com