HANSER

Leseprobe

Florence Maurice

Mobile Webseiten

Strategien, Techniken, Dos und Don'ts für Webentwickler

ISBN (Buch): 978-3-446-43118-8

ISBN (E-Book): 978-3-446-43279-6

Weitere Informationen oder Bestellungen unter

http://www.hanser-fachbuch.de/978-3-446-43118-8

sowie im Buchhandel.

2 Strategien für mobile Webseiten

Dieses Kapitel wird Ihnen die gängigen Strategien im Umgang mit den mobilen Surfern vorstellen. Es erläutert prinzipielle Optimierungen genauso wie das Grundprinzip von Responsive Webdesign, von separaten Webseiten und mobilen WebApps, ohne zu tief in die Technik einzusteigen, was späteren Kapitel vorbehalten ist. Sie erfahren jedoch schon an dieser Stelle die Vor- und Nachteile der unterschiedlichen Herangehensweisen.

2.1 Optimierungen der Desktop-Seite

Zuerst einmal können Sie darangehen, Ihre Desktop-Seite zu optimieren, sodass sie auch auf den mobilen Geräten zumindest *einigermaßen* funktioniert. Dazu gehören zum einen die Performance-Optimierung und zum anderen das Vermeiden kritischer/problematischer Features. Im Detail sind folgende Dinge zu beachten:

- Mit am wichtigsten ist die Optimierung der Performance, d. h. dafür zu sorgen, dass die Seite schneller lädt und die Benutzer schneller zu den Inhalten kommen, die sie interessieren. Von Verbesserungen an der Performance profitieren die Benutzer der Desktop-Seite ebenfalls: Eine Webseite, die auf mobilen Geräten schnell geladen wird, lädt rasant schnell auf dem Desktop. Kapitel 6 ist dem Thema Performance-Optimierungen gewidmet.
- Außerdem sollten Sie schauen, ob die Menüpunkte groß genug sind, dass sie sich mit den Fingern bedienen lassen; die Menüpunkte dürfen auch nicht zu eng beieinander stehen, weil sonst die Gefahr besteht, dass man beim Klicken den falschen erwischt.
- Eigenschaften, die das Scrollen manipulieren, sind ebenfalls kritisch. Dazu zählen die CSS-Anweisungen `overflow: scroll`, position: fixed und auch Effekte wie Parallax-Scrolling.
- Schwierig ist ebenfalls, wenn wichtige Informationen nur beim Hovern eingeblendet werden, weil das im Zweifelsfall bei Touchscreens nicht funktioniert; auch Drop-down-Listen sind anfällig. Dann wäre es zumindest wichtig, dass die Navigationspunkte noch auf einem anderen Web erreichbar sind.
- Ein Suchfeld ist praktisch für einen direkten Zugriff auf die Inhalte und sollte ganz oben auf der Seite platziert werden und einfach erreichbar sein.

- Bei Formularen sollten Sie die neuen Input-Feldtypen wie `email`, `number` oder `tel` nutzen; damit wird automatisch die richtige Tastatur eingeblendet. Browser, die diese Typen nicht unterstützen, behandeln sie einfach als `type="text"` (Genaueres hierzu in Kapitel 4.5).
- Zentrale Informationen wie etwa die Öffnungszeiten bei Restaurants sollten direkt, ohne dass der Benutzer sich durch mehrere Navigationspunkte und Unterseiten klicken muss, erreichbar sein; am besten auf der Startseite oder auf jeder Seite. Wichtig ist ebenfalls eine Telefonnummer, sofern diese Form der Kontaktaufnahme gewünscht ist. Beides sind Sachen, von denen die Nutzer von Desktop-Seiten ebenfalls profitieren.
- Wenn die Webseite auf Flash für zentrale Teile setzt, kommen Sie mit einfachen Optimierungen nicht weiter, da der Flash Player auf iOs nicht läuft.

Diese und weitere Optimierungen können direkt durchgeführt werden, bis dann je nach Situation und Anforderungen weitere Anpassungen für mobile Nutzer erfolgen.

Wesentlich sinnvoller ist es, allgemeine Optimierungen durchzuführen, als Energie darauf zu verwenden, die mobilen Nutzer auf eine eigene Seite umzuleiten, auf der sie erfahren, dass es noch keine mobile Version gibt. Dann sollten Sie lieber gar nichts machen.

2.2 Progressive Enhancement und Feature Detection

Progressive Enhancement ist keine eigenständige Strategie wie die gleich jetzt vorgestellten Ansätze, sondern mehr eine grundlegende Herangehensweise. Dabei geht man von einer funktionierenden Basis aus, die dann für fähigere Geräte optimiert wird, eben „schrittweise verbessert wird“. Progressive Enhancement hilft, mit Browsern/Geräten, die äußerst unterschiedliche Fähigkeiten haben, umzugehen. An mehreren Stellen wird uns Progressive Enhancement wieder begegnen, unter anderem in Kapitel 5.1.

Ein sehr schönes Beispiel für die Umsetzung von Progressive Enhancement ist das Framework jQuery Mobile, wobei das Ergebnis je nach Fähigkeiten des ausführenden Gerätes von einfacher HTML-Seite bis hin zu mobilen WebApp mit dem zugehörigen Look & Feel reicht (siehe auch Kapitel 11).

Hilfreich beim Progressive Enhancement, aber nicht nur dort, ist es, auf Features zu testen, um Browsern mit unterschiedlichen Fähigkeiten auch unterschiedlichen Code zu liefern. Mittel erster Wahl ist hier Modernizr, der in Kapitel 5.1 behandelt wird.

2.3 Responsive Webdesign

Responsive Webdesign ist eine neue Herangehensweise, die Problematik der so unterschiedlichen Geräte zu lösen. Die wichtigste Komponente vom Responsive Webdesign sind CSS3-Media Queries. Über diese kann man je nach Eigenschaften des Ausgebegeräts unterschiedliche Formatierungen anwenden. So kann beispielsweise eine Webseite bei viel verfügbarem Platz dreispaltig und bei wenig verfügbarem Platz einspaltig angezeigt werden.

```
@media screen (and min-width: 500px) {
  /*diese Angaben gelten nur für Geräte mit einer Mindestbreite von 500px */
}
```

Zum Responsive Webdesign gehört aber mehr als Media Queries, klassischerweise auch ein Layout in Prozent – ein sogenanntes flüssiges Layout – und flexible Bilder, die sich an die Bildschirmgröße anpassen. Für die flexiblen Bilder entfernt man die Breitenangaben beim HTML-Code des Bildes und ergänzt Folgendes:

```
img {
  max-width: 100%;
}
```

Damit behalten Bilder ihre ursprüngliche Größe, werden aber nie größer als der sie umgebende Block.

Ein Beispiel für eine Webseite, die nach den Prinzipien des Responsive Webdesign, erstellt wurde, ist *http://www.iso.org*. Bild 2.1 zeigt die Webseite bei viel verfügbarem und Bild 2.2 bei wenig verfügbarem Platz.

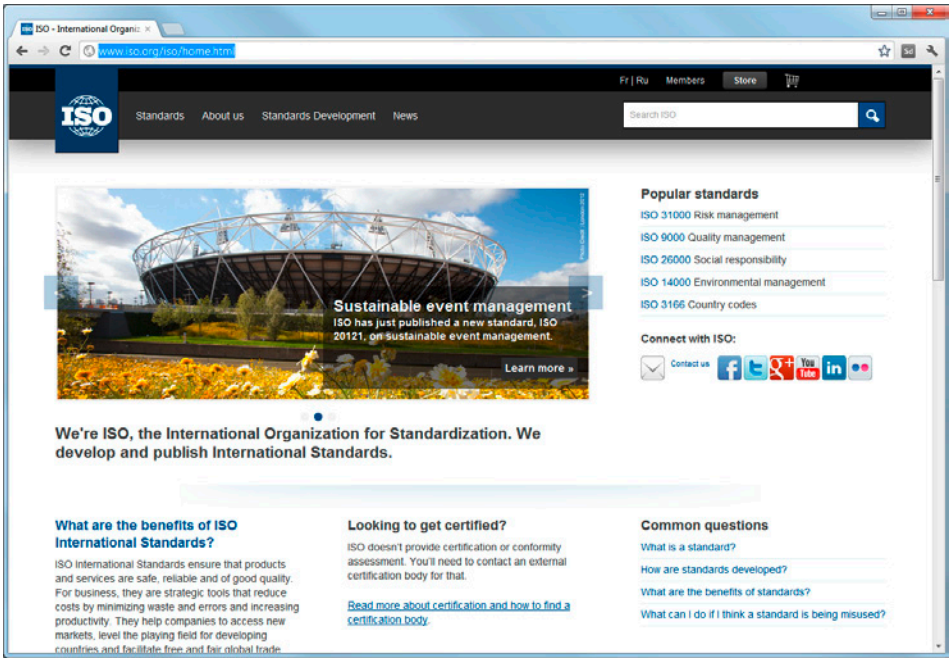

Bild 2.1 ISO-Seite bei viel Platz ...

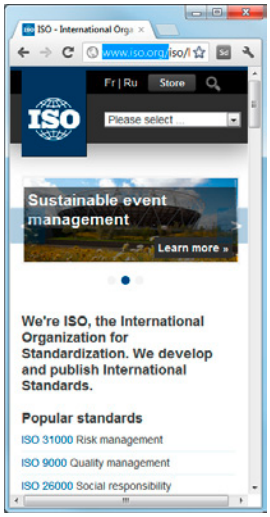

Bild 2.2 … und bei wenig vorhandenem Platz

Der Vorteil von Responsive Webdesign ist, dass es zumindest auf den ersten Blick so einfach scheint: Man muss nur das Stylesheet ein bisschen anpassen und die Angaben für die schmalen Bildschirme ergänzen, und schon hat man eine Webseite, die auf den schmalen Bildschirmen wunderbar aussieht. Diese einfache Variante hat jedoch mehrere Nachteile:

- **Performance:** Bei den über CSS klein skalierten Bildern werden ja immer noch die großen Bilder auf den mobilen Geräten geladen, was die Downloadzeit unnötig erhöht. Auch Inhalte, die einfach mit dem CSS-Befehl `display: none` ausgeblendet werden, müssen trotzdem heruntergeladen werden. Das sorgt für schlechte Performance und die Performance ist gerade für mobile Geräte extrem wichtig.
- **Quellcodeanordnung:** Für Mobilnutzer sind die Inhalte oft nicht in der optimalen Reihenfolge, wenn man von einer bestehenden Desktop-Seite ausgeht. Dann kann es sein, dass der Nutzer lange scrollen muss, um zum Wesentlichen zu kommen. So gilt im Allgemeinen, dass bei einer mobilen Version die Navigation besser nach dem Inhalt untergebracht ist, bei der Quellcodeanordnung von Desktop-Seiten ist es oft anders.
- **Inhaltsmenge:** Überhaupt kann es vorkommen, dass der Nutzer vor lauter Scrollen den Überblick verliert. Auf einer Desktop-Seite ist normalerweise eine Menge Inhalt untergebracht. Wenn man diesen auf einem kleinen Bildschirm unterbringt, wird es schnell unübersichtlich.

Probleme gibt es ebenfalls immer dann, wenn eigentlich unterschiedliche Inhalte auf der mobilen wie auf der Desktop-Version angezeigt werden sollen. Zwei Beispiele hierfür sind:

- Für mobile Geräte, die kein JavaScript ausführen können, gibt es eigene serverseitige Analytics-Tools, beispielsweise Google Analytics for Mobile Websites *(https://developers.google.com/analytics/devguides/collection/other/mobileWebsites)*, aber natürlich sollte man diese nur auf reinen mobilen Seiten nutzen.
- Auf der mobilen Seite ist eventuell eine andere Werbung angebracht als auf der normalen Webseite. Zum Beispiel wären auf der mobilen Webseite Hinweise auf native Apps sinnvoll, weniger auf der Desktop-Seite. Zudem passt nicht jedes Werbeformat für beide Ausgabemodi: Auf beiden problemlos funktioniert textuelle Werbung, aber nicht die Rich

Media-Werbung oder Pop-ups. Zudem ist Werbung oft in einer vorgegebenen Größe und eben nicht flexibel. Wird unterschiedliche Werbung je nach verfügbarem Platz gezeigt, benötigt man eventuell neue Formen der Abrechnung.

Lösungen für die Inhaltsproblematik sind folgende:

- Für Bilder benötigt man eigene Lösungen, die über das klassische Responsive Webdesign hinausgehen und darin bestehen, dass unterschiedliche Bilder je nach anfragendem Gerät ausgeliefert werden. Dafür gibt es eine Reihe von Techniken.

Zum Problem der Inhalte existieren zwei grundsätzliche Strategien:

- Conditional Loading bedeutet, dass man die Inhalte nur bei Bedarf lädt.
- Die andere Strategie kommt ganz ohne Programmiertechnik aus: die Fokussierung auf das Wesentliche. Von diesem Konzeptionsprinzip profitieren auch die Nutzer der Desktop-Seite. Und damit sind wir wieder beim Mobile First-Ansatz: Dieser besagt, dass die mobile Version zuerst konzipiert und erstellt und diese danach für die Desktop-Nutzer angepasst wird. Das allerdings ist nur möglich, wenn Sie eine Webseite von Grund auf neu konzipieren. Außerdem erfordert es ein vollständiges Umdenken, und man muss sich natürlich von Inhalten und Features verabschieden.

Kapitel 10 widmet sich ausführlich dem Thema Responsive Webdesign und der involvierten Komponenten. Performance-Optimierungen sind gerade auch für das Responsive Webdesign wichtig; außerdem können Sie für die bessere Performance auch verstärkt CSS3-Features einsetzen (Kapitel 5) und Techniken zum Einsparen von Bildern nutzen (Kapitel 7).

Responsive Webdesign kann auch mit anderen Strategien kombiniert werden. So können Sie es bei einer separaten mobilen Webseite einsetzen, um mit den unterschiedlichen Bildschirmgrößen der mobilen Geräten umzugehen, wie es beispielsweise von *web.de* praktiziert wird. Oder Sie können Responsive Webdesign bei WebApps anwenden, um die Anordnung der Elemente für die Bildschirme optimal zu gestalten. jQuery Mobile wäre ein Beispiel für Letzteres.

2.4 Separate mobile Webseiten

Das Prinzip der zwei oder mehr separaten Webseiten beruht darauf, dass es verschiedene HTML-Dokumente mit zugehörigen Ressourcen (Stylesheets, Bilder, JavaScript-Dateien) gibt, die je nach Kontext ausgeliefert werden.

Desktop- und Mobile-Version können sich in folgenden Punkten unterscheiden:

- Die Anordnung des **Quellcodes** ist optimiert: Beispielsweise ist die Navigation auf mobilen Webseiten besser unten angeordnet (Content First).
- **Optimierte Bilder:** Auf dem Desktop darf es gerne auch mal groß sein, bei mobilen Geräten sollten Bilder aus Performancegründen nur genau so groß sein, wie man sie braucht. Eine automatische serverseitige Skalierung funktioniert nicht immer, weil dabei manchmal auf einem kleinen Bildschirm wichtige Details nicht erkennbar sind. Hier greifen Sie besser händisch ein und stellen ein optimiertes Bild zur Verfügung, das dann vielleicht auch nur ein Ausschnitt aus dem großen Bild ist.

- **Allgemeine Struktur:** Zweitrangige Inhalte könnten beispielsweise auf der Desktop-Seite direkt integriert werden, aber auf der mobilen Seite macht man sie erst über einen Link erreichbar.
- **Besondere Inhalte für das Smartphone:** Hinweise auf die native App oder ortsbezogene Werbung sind auf der mobilen Webseite wichtig, nicht aber auf der Desktop-Version.

Sichere Annahmen darüber, was Nutzer der mobilen Version brauchen oder nicht, sind sehr schwierig; oft sind die Annahmen falsch und verärgern die Besucher. Eine sinnvolle Alternative zum Nichtbereitstellen ist es, diese erst über einen weiteren Link verfügbar zu machen. Das bedeutet, dass die Grundseite entschlackt ist und schneller lädt, aber insgesamt alle Inhalte erreichbar sind.

- Es wird andere oder keine **Werbung** angezeigt.

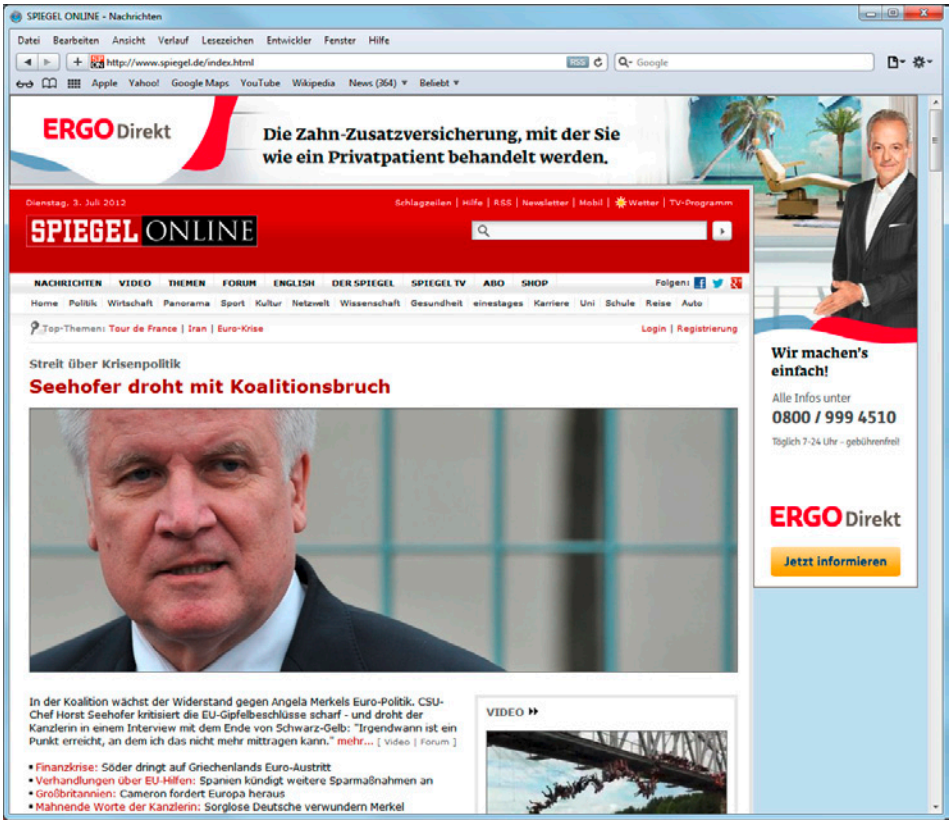

Bild 2.3 spiegel.de: Desktop-Version ...

Üblicherweise findet die Auswahl der passenden Version automatisch statt. Für die separaten Inhalte gibt es zwei Möglichkeiten: Entweder steht die mobile Version unter einer eigenen URL oder aber beide Dokumente sind unter derselben URL erreichbar. Für die eigene

URL werden häufig Subdomains wie *m.beispiel.de* oder *mobil.beispiel.de* eingesetzt. Diese Variante mit der eigenen URL hat den Vorteil, dass Sie diese URL separat publik machen und bewerben können.

Bild 2.4 spiegel.de: Mobile-Version

Die separate Seite ist bei den Top-Alexa-Sites in Deutschland vorherrschend (*http://www.alexa.com/topsites/countries/DE, Stand: Sommer 2012*) und ebenso weltweit, wie die Untersuchungen auf *http://mobiforge.com/designing/blog/server-side-device-detection-used-82-alexa-top-100-sites* zeigen.

Bei separaten mobilen Webseiten gibt es zwei Schwierigkeiten:

Für die Ermittlung, ob es sich um ein mobiles Gerät handelt oder nicht, wird der User-Agent herangezogen, also die Kennung, die der Browser an den Server sendet, wenn er eine Seite anfordert. Dies ist natürlich nur bedingt zuverlässig, zum einen kann der User-Agent-String gefakt werden, zum anderen ist diese Technik per se nicht zukunftssicher. Deswegen ist es an sich wichtig, dem Nutzer die Möglichkeit zu geben, die Version zu wechseln. Und dabei sollte er natürlich nicht nur auf die Hauptseite umgeleitet werden, sondern zur anderen Version der gerade gewählten Seite. Diese Einstellung muss gespeichert werden, damit der Benutzer das nicht dauernd wieder durchführen muss.

Das zweite Problem besteht darin, dass oft bei der mobilen Webseite Inhalte beschnitten und Features reduziert werden. Hiermit werden mitunter mobile Nutzer als Benutzer

der zweiten Klasse disqualifiziert, außerdem basiert das auf Annahmen, die so oft nicht stimmen.

Die Diskriminierung der mobilen Nutzer wird auch durch die Bezeichnung „View Full Site“ (vollständige Seite ansehen) deutlich, die im englischsprachigen Raum verwendet wird, um den Link zum Wechseln zur Desktop-Seite zu kennzeichnen. Beschriftungen wie „Hauptseite“ für die Desktop-Seite oder auch „Internetseite“, „WWW-Seite“ weisen alle darauf hin, dass die mobile Version nur eine beschnittene Teilseite ist und die wichtigen und vollständigen Inhalte auf der Desktop-Seite stehen.

Zusätzlich besteht die Gefahr, wenn die Inhalte separat voneinander gepflegt werden, dass die mobile Version nicht in derselben Art aktuell gehalten wird; so gibt es immer wieder Fälle, in denen Sonderangebote bei der mobilen Version nicht ankommen oder dort noch gelistet sind, nachdem sie nicht mehr zur Verfügung stehen.

PRAXISTIPP: Schöne Beispiele dafür, was auf mobilen Seiten alles so schiefgehen kann, liefern die Screenshots auf *http://wtfmobileweb.com*.

Als die Barrierefreiheit (Accessibility) aufkam, hat man teilweise versucht, eigene barrierefreie Versionen der Hauptseite zu erstellen. Dann hat sich aber deutlich herauskristallisiert, dass das Ziel nicht ist, eine eigene barrierefreie Version zu erstellen, sondern die Standardversion barrierefreier zu gestalten. Dies ist natürlich ein gutes Argument gegen die reduzierte mobile Seite.

Zudem gibt es Untersuchungen, die belegen, dass es Menschen gibt, die das Smartphone oder Feature Phone nicht zusätzlich zum PC verwenden, sondern ausschließlich. Und dann ist es besonders gravierend, wenn diese von den „normalen“ Informationen ausgeschlossen werden.

Der schlimmste Fall ist es, eine separate Webseite für den Hinweis zu verwenden, dass die mobile Seite noch nicht funktioniert – ohne einen Link auf die normale Webseite (siehe Bild 2.5).

Auch wenn man sehr viel falsch machen kann bei der separaten mobilen Webseite, ist sie doch an sich auch eine erwägbare Lösung – eben dann, wenn die Inhalte für die unterschiedlichen Kontexte sehr unterschiedlich sind oder ganz unterschiedlich organisiert werden müssen. Die separate mobile Seite ist auch eine gute Strategie, wenn man eben gar nichts an der Desktop-Seite ändern kann.

Umgekehrt kann die serverseitige Auslese des User-Agents beispielsweise auch beim Responsive Layout zur Auslieferung separater Bilder benutzt werden, um zu entscheiden, welche Version einer Ressource geschickt werden soll. Kapitel 13.7 stellt zudem Kombinationen von clientseitigen und serverseitigen Tests vor.

Bild 2.5 Mobile Version kommt bald, ach was ...

2.5 Mobile WebApps

Was genau ist eine WebApp? Klar ist, was eine native App ist. Das ist eine Anwendung, die in einer vorgegebenen Sprache direkt für ein bestimmtes Betriebssystem geschrieben ist, zum Beispiel kann man Anwendungen für iOS in Objective-C oder Cocoa erstellen, Android Apps mit Java. Im Gegensatz dazu basieren mobile WebApps auf den Sprachen des Webs, d. h. auf HTML, CSS und JavaScript.

Was unterscheidet mobile WebApps aber von mobilen Webseiten? Diese Frage ist schwerer zu beantworten. Es gibt auch Stimmen, die sagen, man müsse die beiden nicht voneinander abgrenzen. Besuchern sei es egal, ob sie eine WebApp nutzen oder eine Webseite besuchen. Aber für Entwickler ist es doch wichtig, sich die Unterschiede zwischen beiden klarzumachen.

Zur Abgrenzung von mobiler WebApp gegenüber mobiler Webseite werden verschiedene Kriterien herangezogen.

Optik – Look & Feel

Eine mobile WebApp vermittelt eine App-like Erfahrung. Das heißt, dass sie vom Benutzer ähnlich wie eine native App wahrgenommen wird. Typisch ist also für WebApps ein bestimmtes Aussehen. Üblicherweise orientieren sich WebApps optisch mehr an den nativen Apps als an der Desktop-Seite.

Typisch sind folgende Komponenten (im Unterschied zu mobilen Webseiten):

- Fixierte Toolbar oben, eine Reihe von Buttons unten.
- Animierte Seitenübergänge bei der Navigation zwischen den verschiedenen Bereichen.
- Typisch sind auch Elemente wie ActionSheet, das sind Meldungsfenster, die mit zusätzlichen Optionen von unten auf dem Bildschirm erscheinen, oder Ladeanzeiger.

Funktionalität

Webseiten besucht man, WebApps nutzt man. Die Interaktion bei Webseiten ist Lesen, bei WebApps ist es mehr. Es gibt natürlich Ausnahmen von diesem Prinzip und im Allgemeinen geht die Tendenz im Web immer mehr in Richtung „Anwendungen“ und weg von den „reinen Webseiten“ (nicht zufällig dient HTML5 der Erstellung von Anwendungen). Deswegen hilft das Kriterium der besonderen Funktionalität bei WebApps, kann aber nicht zur eindeutigen Abgrenzung dienen.

Architektur

Beim klassischen Web ist der Browser ein **Thin Client**, und der Webserver erledigt die Hauptarbeit.

Bei WebApps haben wir es hingegen mit einem **Fat Client** zu tun, Hier macht der Browser wesentlich mehr; involviert sind Techniken wie AppCache (siehe Kapitel 9.1), mit der die Webanwendung offline-fähig wird, WebStorage zur Datenspeicherung und vieles mehr – es wird also das daraus, was man treffend als Rich Internet Application bezeichnet.

Reichweite

Apps laufen auf einer begrenzten Anzahl an Geräten, weil sie bestimmte Techniken voraussetzen. Eine Webseite kann nicht so viel leisten, weil sie für möglichst viele funktionieren muss.

Tabelle 2.1 Gegenüberstellung WebApp vs. mobile Webseite

	Mobile Webseite	WebApp
Optik	Orientiert sich an Desktop-Webseite, CI im Vordergrund	Orientiert sich an nativen Apps
Funktionalität	Lesen, Informationen aufnehmen	Etwas tun, ein konkretes Ziel verfolgen
Reichweite	Smartphones mit bestimmten technischen Voraussetzungen	Nicht per se auf bestimmte Geräte beschränkt
Architektur	Thin Client – Server erledigt Hauptarbeit	Fat Client: Intensive Nutzung von JavaScript, offline-fähig, Datenspeicherung über Local Storage

Interessant ist die diesbezügliche Diskussion zur Abgrenzung von James Pearce *(http://tripleodeon.com/2011/09/of-sites-and-apps)*. Er führt auch die Architektur als Hauptkriterium der Unterscheidung an.

Gängige Smartphone-Betriebssysteme bieten die Möglichkeit, Webseiten direkt zum Homescreen hinzuzufügen, sodass sie darüber gestartet werden. Auch in diesem Punkt erinnern sie an Apps.

Das Problem bei mobilen WebApps kann sein, dass sie den nativen Apps optisch zu ähnlich werden, aber nicht dieselbe Performance bieten. Außerdem wirken sie, wenn die Optik einer Plattform zu sehr imitiert wird, unpassend auf den anderen Plattformen. In Kapitel 11 und Kapitel 12 lernen Sie zwei Frameworks zur Erstellung von mobilen WebApps kennen. Typische APIs, die bei der Programmierung von „Fat Clients“ eingesetzt werden, finden Sie in Kapitel 9.

2.6 Native Anwendungen

Schließlich gibt es noch die nativen Anwendungen, die nicht Thema dieses Buches sind. Eine native App hat folgende Charakteristika:

- Native Anwendungen werden speziell für eine Plattform erstellt. Eine native Anwendung, die fürs iPhone erstellt ist, läuft nicht auf einem Android-Gerät. Man kann aber beispielsweise Apps erstellen, die sowohl auf dem iPhone als auch auf dem iPod laufen.
- Die native App ist in speziellen Techniken erstellt. Android-Applications werden in Java geschrieben, Applications fürs iPhone in Cocoa oder in Objective-C.
- Eine native App wird installiert.
- Typisch sind auch die besonderen Vertriebswege. So werden iPhone-Applications über den Apple Store vertrieben, Android Application über Google Play.
- Eine native App kann auf bestimmte Features direkt zugreifen, etwa auf Adressbuch oder Kamera.

- Sie ist üblicherweise im Look & Feel am Betriebssystem angepasst.
- Es gibt eine unglaublich große Anzahl an nativen Anwendungen. Das bedeutet natürlich, dass native Apps sehr erfolgreich sind. Es zeigt aber auch, dass die Konkurrenz für eine neue Apps groß und dass es schwer ist, die notwendige Aufmerksamkeit für eine bestimmte App zu erhalten.

Die Endlosdiskussion nativer App vs. WebApp möchte ich an dieser Stelle nicht ausführlich aufnehmen, sondern nur auf wichtige Unterschiede zwischen den beiden hinweisen.

Tabelle 2.2 Unterschiede zwischen nativer App und mobiler WebApp

	Native Anwendung	Mobile WebApp
Vertriebsweg	AppStore	offen
Technologie	plattformspezifisch	Webtechniken: HTML, CSS, JavaScript
Zugriff auf Hardware-Komponenten	direkter Zugriff auf Kamera, Adressbuch etc.	eingeschränkt (in diesem Bereich tut sich aber einiges)
Performance	für bestimmtes Betriebssystem optimiert, deswegen performanter	weniger performant
Kosten	teuer, besonders, wenn für verschiedene Plattformen entwickelt werden muss	Im Allgemeinen günstiger

Und was ist wann sinnvoll? Die Zweckmäßigkeit von nativen Apps ist unbestritten, sicher nicht in allen Gebieten, aber diese Diskussion ist eine andere – die Frage, die sich, das ist die Prämisse des Buches, stellt, ist: Native App hin oder her – können Sie es sich erlauben, keine für die mobilen Nutzer geeignete Webseite bereitzustellen?

Es gibt natürlich auch große Firmen wie Facebook, die sich beides leisten: Neben den nativen Apps für iPhone und Android gibt es auch die WebApp. Für Facebook ist der mobile Zugriff aber auch besonders wichtig, mehr als die Hälfte der User nutzen Facebook mobil. Facebook legt einen Schwerpunkt auf die mobile WebApp und diese hat mehr Nutzer als die Nutzer von iPhone- und Android-Version zusammen *(http://www.readwriteweb.com/mobile/2012/04/james-pearce-head-of-mobile.php)*.

2.7 Die Mischung machts

Die verschiedenen Strategien lassen sich auch kombinieren. Ein paar Beispiele für sinnvolle Kombinationen:

- Responsive Layout setzt sinnvollerweise auf eine Technik für separat ausgelieferte Bilder, die serverseitigen Code und so etwas wie Device Detection voraussetzt.
- Responsive Techniken können sowohl bei einer WebApp als auch bei einer separaten mobilen Webseite benutzt werden.

- Auch bei einer normalen mobilen Webseite kann man verstärkt clientseitige (Fat Client) Techniken zur Performancesteigerung einsetzen, beispielsweise schadet Offline-Fähigkeit auch einer normalen mobilen Webseite nicht und auch die Speicherung von Inhalten im LocalStorage kann zur Performance-Optimierung benutzt werden.

Inzwischen gibt es mehrere sehr vielversprechende Ansätze der bewussten Kombination von serverseitiger Erkennung und Responsive Webdesign bzw. von clientseitiger Feature Detection in Kombination mit serverseitiger Kennung. Ausgewählte Ansätze werden in Kapitel 13.7 vorgestellt.

2.8 Kurz zusammengefasst

Dieses Kapitel hat Ihnen die unterschiedlichen klassischen Strategien vorgestellt, um dem mobilen Nutzer eine passende Version bereitzustellen:

- Im einfachsten Fall optimieren Sie eine bestehende Webseite, indem Sie ihre Performance verbessern und kontrollieren, dass bestimmte Features, die auf mobilen Geräten kritisch sind, nicht eingesetzt werden.
- Beim Responsive Webdesign werden Anpassungen für verschiedene Bildschirmgrößen per CSS durchgeführt, es kann aber durch zusätzliche Komponenten erweitert werden.
- Die separate mobile Webseite setzt auf serverseitige Erkennungsstrategien, um dann unterschiedliche und angepasste Inhalte für die einzelnen Geräte auszuliefern.
- Mobile WebApps beruhen wie mobile Webseiten auf Webtechniken. Optisch erinnern sie aber an native Apps und sie zeichnen sich außerdem durch den intensiven Gebrauch von JavaScript und JavaScript-APIs aus.
- Die verschiedenen Ansätze können auch kombiniert werden.

In späteren Kapiteln wird die mögliche Umsetzung noch detailliert vorgestellt. In Kapitel 3 beschäftigen wir uns jedoch zunächst mit allgemeinen Punkten zur Anordnung von Inhalten, bevor wir mit den allgemeinen Techniken anfangen, die unabhängig von der gewählten Strategie sind.