Eine Kategorisierung mobiler Applikationen

Mobile Applikationen lassen sich auf verschiedene Arten entwickeln. Je nachdem welche Anforderungen an eine Applikation gestellt werden, sollte zwischen einer nativen Umsetzung, einer Webapplikation oder einer Mischung aus beidem gewählt werden. Die Wahl der am besten passenden dieser drei Kategorien ist nicht immer einfach und sollte im Vorfeld genau abgeklärt werden. In diesem Artikel zeigen wir die Eigenheiten dieser drei Kategorien auf und vergleichen sie miteinander anhand von Beispielen und einer fiktiven Ticketing-Applikation.

Andreas Hildebrandt, Jürg Luthiger, Christoph Stamm, Chris Yereaztian | juerg.luthiger@fhnw.ch

Mobile Applikationen für Smartphones, so genannte Apps, lassen sich grob in drei Kategorien einteilen: native Apps, Webapplikationen und eine Mischung aus beidem, die hybriden Apps. Nicht alle Anwendungen lassen sich gleich gut mit allen drei Kategorien umsetzen, da jede Kategorie ihre Vorzüge und Nachteile mit sich bringt.

In diesem Artikel charakterisieren wir die drei Kategorien und geben für jede Kategorie ein bekanntes Beispiel aus der Praxis an. Zudem versuchen wir anhand einer fiktiven Ticketing-App abzuschätzen, welche Probleme bei der Umsetzung dieser App in den drei Kategorien auftreten könnten. Die Ticketing-App soll das Bestellen und Verwalten von Tickets vereinfachen. Ähnlich der Ticketing-App der SBB sollen von möglichst vielen Benutzern Tickets über das Web bestellt und lokal auf dem Gerät gespeichert werden können. Nach dem Ticket-Download soll keine weitere Internet-Verbindung mehr notwendig sein. Dies erleichtert und beschleunigt den Abruf des Tickets bei der Ticketkontrolle. Durch einfaches Schütteln des Smartphones soll das letzte Ticket zum Vorschein gebracht werden, ohne dass die Benutzerin das Gerät mit den Fingern bedienen muss.

Native Anwendungen

Native Anwendungen werden in vielen Fällen speziell für die entsprechende Zielplattform geschrieben und direkt auf dem Gerät ausgeführt. So werden beispielsweise iPhone-Apps hauptsächlich in Objective-C und Android-Apps in Java entwickelt. Daneben gibt es aber auch vermehrt Ansätze, native Anwendungen für mehrere Plattformen gleichzeitig zu entwickeln. Die Firma Xamarin[1] bietet zum Beispiel die Möglichkeit, den Quellcode des Logikteils für alle angebotenen Plattformen in C# zu schreiben. Das User-Interface wird danach plattformspezifisch gestaltet und entwickelt.

Weil native, mobile Applikationen plattformspezifisch implementiert sind, kann das plattformtypische Look-and-Feel vollständig umgesetzt werden. Dadurch fühlt sich die Benutzungsschnittstelle der App für eine mit der Plattform vertraute Benutzerin sehr intuitiv an, während ein plattformfremder eine gewisse Eingewöhnungszeit benötigt. Die Bedienung ist zudem bestens auf die Hardware abgestimmt und flüssig. Native Anwendungen können über die betriebssystemspezifische Schnittstelle (API) alle Funktionen des Gerätes direkt nutzen. Durch das API hat der Entwickler auch vollen Zugriff auf alle Komponenten und Sensoren des Gerätes. Somit lassen sich die Vorteile des jeweiligen Gerätes voll ausnutzen.

Typische native Applikationen sind Kamera-Anwendungen, Spiele oder VoIP-Anwendungen. Abbildung 1 zeigt exemplarisch die unterschiedliche Umsetzung der Chatfunktion von Skype auf iOS und Android. Auf dem rechten Bild der iPhone-App ist ein Back-Button oben in der Navigationsleiste vorhanden und unten ist die Tab-Bar gemäss Apple Guidelines platziert. Auf dem linken Bild der Android-App ist die typische Action Bar des Android-Systems zu sehen. Einen Back-But-

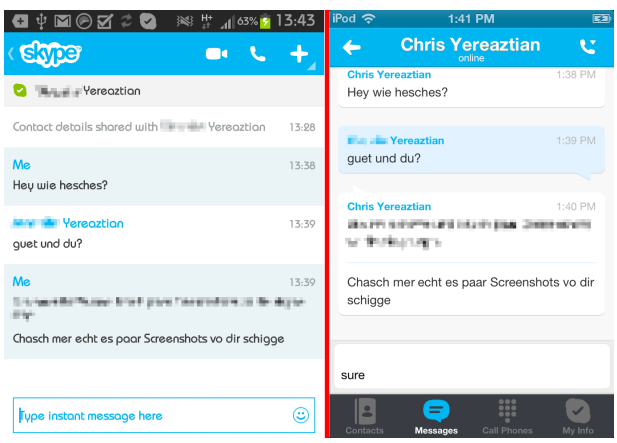

Abbildung 1: Plattformspezifische Umsetzung der Skype-Chatfunktion unter iOS (links) und Android (rechts). Namen und privater Text unleserlich gemacht.

1 Xamarin: http://xamarin.com/

```
<activity android:label="@string/app_name" android:name="com.phonegap.DroidGap"
      android:configChanges="keyboardHidden|orientation»>
  <intent-filter />
</activity>
```

Listing 1: Manifest-Datei mit PhoneGap-typischem Eintrag

ton braucht es bei Android-Geräten nicht, da ein Hardware Button genutzt wird.

Der Vertrieb nativer, mobiler Anwendungen wird meistens über einen entsprechenden App-Store realisiert, zum Beispiel den *App Store* für iPhone oder den *Google Market* für Android. Dabei ist zu beachten, dass bei einigen App-Stores der Genehmigungsprozess viel Zeit beanspruchen kann.

Eine native Realisierung der skizzierten Ticketing-App ist grundsätzlich möglich, da alle notwendigen Funktionalitäten und Sensorzugriffe möglich sind. Performance-Probleme wird es keine geben und das Look-and-Feel entspricht demjenigen anderer nativer Apps. Für den Entwickler bzw. Anbieter der App erhöht sich der Aufwand mit der Anzahl verschiedener Plattformen, die unterstützt werden sollen. Die App und spätere Anpassungen müssen über die App-Stores verteilt werden und stehen somit erst mit einer gewissen Verzögerung von ein paar Tagen zur Verfügung.

Hybride Anwendungen

Hybride Apps bestehen aus zwei unterschiedlichen Komponenten: Die native Komponente kann direkt auf die betriebssystemspezifischen Funktionalitäten zugreifen, während die Web-Komponente in einem entsprechenden Container läuft. Dabei wird angestrebt, möglichst grosse Teile der Applikationslogik in der plattformunabhängigen Web-Komponente zu implementieren. Nur dort wo auf plattformspezifische Funktionen zugegriffen werden muss, wird die native Komponente eingesetzt. So bietet die native Komponente Zugriff auf Kamera, Kompass oder andere Sensoren, für die es keine standardisierten Schnittstellen gibt.

Zur Realisierung hybrider Apps kommen oft Frameworks, wie zum Beispiel PhoneGap oder Titanium, zum Einsatz. Diese Frameworks stellen die Kommunikation zwischen dem nativen und dem Web-Teil sicher. Die zu verwendende Programmiersprache wird dabei durch das Framework festgelegt. Einige Frameworks nutzen die Websprachen HTML, JavaScript und CSS, andere wiederum verwenden C#.

Ein typisches Beispiel für eine hybride, mobile Anwendung ist die Wikipedia-App von Wikimedia Foundation[2]. Wikipedia bietet für mobile Geräte, seien dies Tablets oder SmartPhones, eine optimierte Variante ihrer Webseite (inklusive aller Artikel) an, die mit den üblichen mobilen Browsers kompatibel ist. Die Applikation verwendet auf Android sowie auf iOS PhoneGap[3], um die auf beiden Plattformen verfügbare SQLite-Datenbank verwenden zu können[4]. Einmalig heruntergeladene Artikel werden in der SQLite-Datenbank zwischengespeichert, so dass die Applikation auch ohne Internet-Verbindung verwendet werden kann. Artikel zu bereits gesuchten Begriffen können somit auch unterwegs ohne Internetverbindung gelesen werden.

Da die Wikipedia-App keine speziellen grafischen Elemente wie Check-Boxen oder Auswahllisten verwendet, lässt sich nicht so einfach erkennen, dass es sich um eine Applikation handelt, die mit PhoneGap umgesetzt worden ist. Jedoch lässt sich anhand des Quellcodes oder der Installationsdatei (bei Android das Application Package File) schnell feststellen, ob eine Applikation ein bestimmtes Framework verwendet oder nicht. Die Installationsdatei einer Android-Applikation kann mit dem *Android Asset Packaging Tool* entpackt werden. Im Hauptverzeichnis der Applikation befindet sich die Manifest-Datei, in der die einzelnen Bestandteile der Applikation definiert sind. PhoneGap-Applikationen besitzen üblicherweise einen Eintrag für die Start-Activity, welche die index.html Datei samt dazugehörigen JavaScript lädt (siehe Listing 1).

Das hybride Modell bietet den Vorteil, dass die gleiche Applikationslogik für mehrere Plattformen genutzt werden kann, da sie nicht spezifisch für eine Plattform geschrieben wird, sondern gegen ein Framework wie z.B. PhoneGap oder Titanium[5]. Das Framework liefert den In-Between-Code, der den plattformunabhängigen Code in die native API übersetzt. Je umfangreicher die mobile Anwendung ist und je spezieller die Funktionen sind, desto aufwendiger werden jedoch diese Anpassungen.

Bei UI-Elementen ist speziell zu beachten, dass die häufigsten Frameworks im Look-and-

Abbildung 2: PhoneGap Demo-App auf einem Android SmartPhone mit iPhone Look-and-Feel

2 Wikimedia Foundation: http://www.wikimedia.org/
3 PhoneGap: http://phonegap.com/
4 Wikipedia nutzt PhoneGap: http://wwwimages.adobe.com/www.adobe.com/content/dam/Adobe/en/customer-success/pdfs/wikimedia-foundation-case-study.pdf
5 Appcelerator Titanium: http://www.appcelerator.com/

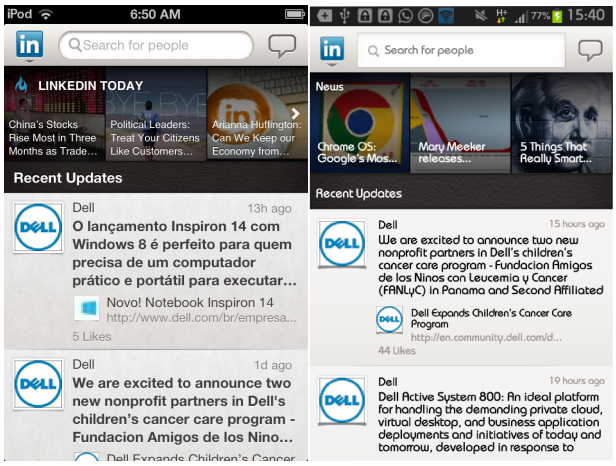

Abbildung 3a: Gegenüberstellung der Webapplikation LinkedIn unter iOS (links) und Android (rechts)

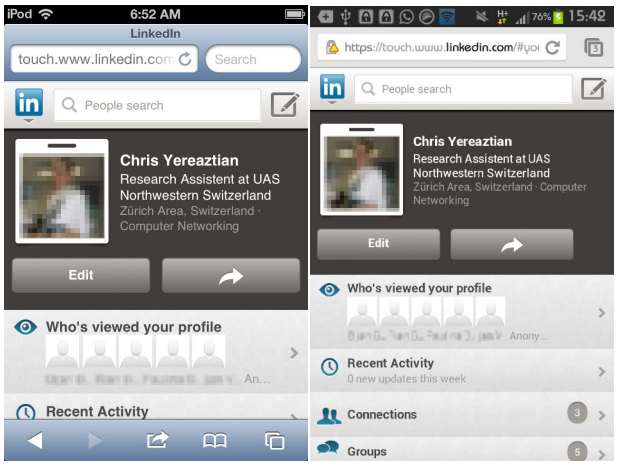

Abbildung 3b: Gegenüberstellung der Webseite LinkedIn im Safari (links) und Chrome Mobile (rechts). Profilbild und Namen unleserlich gemacht

Feel das native iOS nachahmen, so dass derartig umgesetzte UI-Elemente auf einem Android- oder Windows-Gerät nicht wie native Applikationen wirken. Abbildung 2 zeigt eine PhoneGap Demo-Applikation, welche Daten aus dem Beschleunigungssensor ausliest und darstellt. Für Android und Windows Phone bringt der Back-Button oben Links keinen zusätzlichen Nutzen, da diese Funktionalität über einen Hardware-Button abgedeckt wird. Die iOS-Geräte jedoch besitzen nur einen einzigen Hardware-Button, den sogenannten Home-Button, und deshalb sind die iPhones auf solche UI-Buttons angewiesen. Die dargestellten Listenelemente sehen den Listenelementen auf iOS sehr ähnlich, nutzen aber nicht die von der Plattform angebotenen Implementierungen.

Der Vertrieb hybrider, mobiler Anwendungen wird wie bei nativen Applikationen über einen entsprechenden App-Store realisiert.

Eine hybride Realisierung der skizzierten Ticketing-App ist grundsätzlich möglich, da alle notwendigen Funktionalitäten und der Zugriff auf den Bewegungssensor möglich sind. Mit Performance-Problemen ist vorliegenden Fall kaum zu rechnen, da die Benutzungsoberfläche recht einfach gehalten werden kann. Das Look-and-Feel wird höchstens auf einer Plattform demjenigen nativer Apps entsprechen. Weil eine gemeinsame Code-Basis für mehrere Plattformen verwendet werden kann, bleibt der Aufwand für den Entwickler mehrheitlich unabhängig von der Anzahl der unterstützten Plattformen, solange das verwendete Framework diese Plattformen unterstützt. Diese Abhängigkeit eines Frameworks kann zudem nachteilig sein, wenn die Weiterentwicklung des Frameworks aus irgendwelchen Gründen ins Stocken gerät. Wie bei einer nativen App müssen die App und spätere Anpassungen über die App-Stores verteilt werden und stehen somit erst mit einer gewissen Verzögerung von ein paar Tagen zur Verfügung.

Webapplikationen

Eine klassische Webapplikation wird auf einem Web-Server ausgeführt und verwendet üblicherweise den Web-Browser des Clients als Benutzungsschnittstelle. Mit HTML5 und der Verwendung von JavaScript verschiebt sich jedoch wieder ein grosser Teil des ausführbaren Codes vom Server auf den Client. Somit übernimmt der Browser die Aufgabe eines standardisierten Applikationsfensters mit den dazugehörigen UI-Elementen. Anpassungen beschränken sich dadurch hauptsächlich auf die unterschiedlichen Bildschirmdarstellungen bei den verschiedenen Display-Grössen.

Mit dem Verzicht der Nutzung nativer Schnittstellen, gehen gewisse Einschränkungen bezüglich der Nutzung von Sensoren und Kommunikationsschnittstellen einher. Um damit zusammenhängende Performance-Einbussen bei der Ausführung von sequentiellem JavaScript Code zu umgehen, definiert HTML5 einen neuen Web-Worker-Standard, welcher es erlaubt, JavaScript Code in einem separaten Thread auszuführen[6].

Ein Beispiel einer Web-Anwendung ist die LinkedIn-App[7]. Diese verwendet unter Android und iOS nur HTML5 und JavaScript und verzichtet dabei ganz auf ein Framework. Die Applikation nutzt eine sogenannte Web-View, um HTML-Seiten in einer Applikation mit der Rendering-Engine des System-Browsers anzuzeigen, ohne dass dafür spezieller Code benötigt wird.

In Abbildungen 3a und 3b ist zu sehen, wie sich die Grösse der Schrift und die Anordnung der dazugehörigen Elemente unterscheiden. Mit HTML lässt sich die Schriftgrösse ohne weiteres an die Bildschirmauflösung angepasst.

Der Vertrieb von reinen Webapplikationen beschränkt sich auf die Aktualisierung der Applikation auf einem Web-Server. Korrekturen und Ak-

6 Einführung Web-Worker: http://www.html5rocks.com/en/tutorials/workers/basics/

7 Link zur LinkedIn-App: https://www.linkedin.com/

tualisierungen können daher in viel kürzerer Zeit veröffentlich werden. Zudem ist sichergestellt, dass alle Benutzerinnen die neuste Version der Applikation einsetzen.

Eine Realisierung der skizzierten Ticketing-App als Webapplikation ist für die Bestellung und Bezahlung des Tickets problemlos möglich und sogar sinnvoll, da dadurch möglichst viele potentielle Benutzer angesprochen werden können. Sogar das Abrufen des aktuellen Tickets ist mittlerweile möglich, da mittels HTML5 und JavaScript der Bewegungssensor abgefragt werden kann. Mit Performance-Problemen ist allenfalls bei der Detektion der Schüttelbewegung zu rechnen, da der Bewegungssensor in kurzen Abständen mehrfach hintereinander aufgerufen werden muss. Das Look-and-Feel wird sich von demjenigen nativer Apps unterscheiden. Weil eine gemeinsame Code-Basis für mehrere Plattformen verwendet werden kann, bleibt der Aufwand für den Entwickler mehrheitlich unabhängig von der Anzahl der unterstützten Plattformen. Die Verteilung der Webapplikation ist denkbar einfach und Anpassungen oder neue Funktionen lassen sich schnell und einfach verbreiten.

HTML5-Hype

Wie bereits angesprochen, verschiebt sich mit dem Einsatz von HTML5 und JavaScript wieder ein grosser Teil des ausführbaren Codes vom Server auf den mobilen Client. Dadurch verwischen sich die Grenzen zwischen nativen Apps und mobilen Webapplikationen immer mehr. Nicht zuletzt deswegen gewinnt die Entwicklung von HTML5-Apps immer mehr an Bedeutung. Viele der Applikationen, die heute in den App-Stores angeboten werden, sind in HTML5 und JavaScript geschrieben. Weitere Gründe für diesen Hype können folgende Punkte sein:

- Mobile Webapplikationen sind einfacher und schneller zu implementieren. Um eine native Applikation für ein iPhone oder Android Gerät zu schreiben sind jeweils eigene Entwicklungsumgebungen notwendig. Die iPhone Entwicklung kann zusätzlich nur auf einen Computer von Apple erfolgen. Neben diesen Voraussetzungen braucht es Kenntnisse in einer Programmiersprache der entsprechenden Plattform (Android: Java, IPhone: Objective-C, …). Da HTML sowie JavaScript einfacher zu erlernen sind, dürfte die Hürde, eine eigene App zu schreiben, sehr viel tiefer liegen.
- Die momentan starke Fragmentierung der mobilen Betriebssysteme wirkt sich negativ auf die Entwicklung mobiler Apps aus. Unter Android gibt es die komplette Palette von der Version 1.5 bis hin zu 4.1[8]. Unter iOS und Windows Phone sieht es zwar nicht ganz so extrem aus, aber es gibt auch hier verschiedene Versionen. Mit HTML5 kann eine Anwendung geschrieben werden, die unabhängig von der aktuellen und der genutzten OS Version lauffähig ist und keine Einschränkungen besitzt.
- Durch die Plattformunabhängigkeit von Webapplikationen ist es sehr viel einfacher, eine App zu veröffentlichen. Eine App, die in HTML und JavaScript geschrieben ist, kann leicht von einem Android- auf ein iOS- oder Windows-Gerät portiert werden. Damit lassen sich sehr viel mehr Benutzer erreichen, während der Vertriebsaufwand minimal gehalten werden kann.
- Schliesslich sind Webapplikationen nicht an Restriktionen der jeweiligen Systeme gebunden. Das heisst, dass bei einem Release einer neuen Version die App praktisch sofort den Benutzern zugänglich gemacht werden kann. Auf iOS oder Windows Phone muss eine App erst eine längere und kostenpflichtige Prozedur durchlaufen, bevor sie für den Store freigegeben wird.

Native Apps versus (hybride) Webapplikationen

Im eingangs aufgezeigten Szenario mit der mobilen Ticketing-Anwendung spielt die möglichst grosse Verbreitung eine sehr zentrale Rolle und daher ist die Plattformunabhängigkeit von grösster Wichtigkeit. Reine oder hybride Webapplikationen haben in einer solchen Situation einen klaren Vorteil gegenüber nativen Apps. Die Verwendung einer reinen Webapplikationen hängt jedoch nicht zuletzt davon ab, ob Sensoren benötigt werden und sich diese von der Webapplikation ansteuern lassen. Der zweite Knackpunkt ist die Effizienz. Es stellt sich also die Frage, ob die Performance einer Web-Anwendung ausreicht oder ob einzelne Teile oder eventuell sogar die ganze Anwendung nativ realisiert werden müssen.

Die einst klaren Grenzen zwischen nativen Apps und Webapplikationen verwischen sich immer mehr. Speziell die neuen HTML5-Standards[9] ermöglichen es, dass sich eine mobile Webapplikation immer mehr wie eine richtige native Applikation verhält. So erlaubt z.B. der Standard Web-Storage[10], Daten über mehrere Browser-Sessions hinweg offline zu sichern. Dem Benutzer kann so ein Gefühl einer nativen App vermittelt werden auch wenn keine Internet-Verbindung zur Verfügung steht. Hingegen besteht aber immer noch das Problem, dass die Grafikelemente in den HTML5-Applikationen wie Buttons, Listen etc. sich stark von den nativ entwickelten Elementen unterscheiden. Eine Ausnahme dazu liefert aber

8 Fragmentierung Android OS: http://developer.android.com/about/dashboards/index.html, http://mashable.com/2012/05/16/android-fragmentation-graphic/

9 Übersicht der mobilen HTML5-Standards: http://mobilehtml5.org

10 Browser Storage Support: http://www.html5rocks.com/de/features/storage, http://csimms.botonomy.com/2011/05/html5-storage-wars-localstorage-vs-indexeddb-vs-web-sql.html

	nativ	hybrid	Webapp
Grafiken	native API	HTML, Canvas, SVG	HTML, Canvas, SVG
Video/Audio	Ja	Ja	Ja
3D Grafiken (WebGL)	Ja	Ja	Ja
Kamera	Ja	Ja	Nein
Notifikationen	Ja	Ja (1)	Nein
Kontakt	Ja	Ja	Nein
Kalender	Ja	Ja (2)	Nein
Offline Storage	Ja	Ja	Ja
Geolokation	Ja	Ja	Ja
Beschleunigungssensor	Ja	Ja	Ja

Tabelle 1: Unterstützung mobiler Komponenten in den drei Applikationskategorien. (1) Nicht mit jedem Framework und/oder mit Urban Airship. (2) Zum Teil über Third-Party Module.

JQuery[11] mit der JavaScript-Bibliothek JQuery-Mobile. Das Design der Elemente ist dort stark an iOS angelehnt.

Der grosse Vorteil einer hybriden gegenüber reinen Webapplikation liegt darin, dass die erstere über die native Komponente des Frameworks auf Teile der nativen API zugreifen kann, während die letztere Bibliotheken in JavaScript benötigt, die wiederum nur eingeschränkten Zugriff auf die Hardware ermöglichen. In Tabelle 1 ist die Unterstützung der wichtigsten Elemente von mobilen Plattformen in Abhängigkeit der drei Kategorien von mobilen Applikationen aufgelistet. Beispielsweise lässt sich auf die Kontakt- und Kalender-APIs[12] und die diversen Sensoren nur von nativen und hybriden Apps zugreifen. Da nicht alle Frameworks immer sämtliche Funktionen aller Plattformen unterstützen, spielt die Wahl des gewählten Frameworks sowie des Betriebssystems eine entscheidende Rolle über die Integrationsmöglichkeiten nativer Funktionalität.

Werden APIs angepasst oder gar neu bereitgestellt (z.B. Notifications für iOS oder NFC für Android 2.3.x), können diese bei hybriden, mobilen Apps oft nicht gleich genutzt werden. Die Framework-Entwickler brauchen dafür erst Zeit um diese Schnittstellen in das Framework zu integrieren und anschliessend freizugeben.

Performance-Betrachtung

Die Performance einer hybriden, mobilen Applikation mit einer schlicht gestalteten Benutzeroberfläche kann durchaus mit einer nativen App mithalten. Ist das User Interface komplexer aufgebaut, reagiert die Anwendung schwerfälliger, weil die Darstellung von nicht nativen UI-Elementen durch die CPU aufwendig berechnet werden muss[13]. Im Vergleich mit einer reinen mobilen Webapplikation ist eine hybride, mobile Anwendung nicht an die Sandbox des Browsers gebunden, sondern unterliegt den gleichen Einschränkungen (Anzahl Threads, Speicherbedarf) wie eine native App.

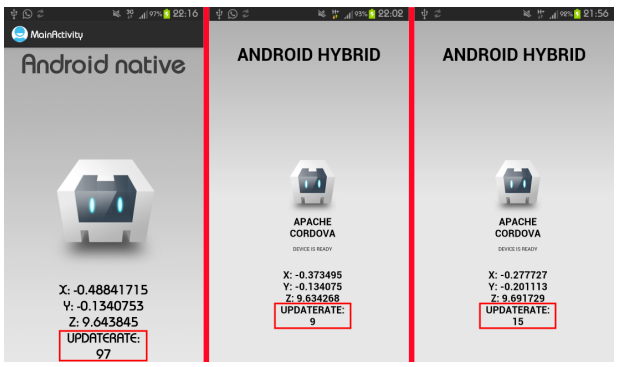

Abbildung 4: Aktualisierung der Beschleunigungssensordaten im Vergleich. Links: native Test-App; Mitte: Test-App mit PhoneGap unmodifiziert; rechts: Test-App mit PhoneGap modifiziert, ohne Beschränkung der Aktualisierungsrate.

Bei einer Webapplikation hängt die Performance stark von der Gestaltung der Benutzeroberfläche ab. Je aufwändiger diese gestaltet ist, desto langsamer läuft die Anwendung. Dabei kann die Performance-Einbusse an der Geräteleistung liegen oder auch durch technische Massnahmen künstlich hervorgerufen werden. Beispielsweise begrenzt Apple bei ihrem Betriebssystem die Leistung aller Browser, damit sichergestellt wird, dass für das OS genügend Leistung bereitgestellt werden kann. Diese Einschränkung führt beim Scrollen zu ruckeln und fühlt sich sehr störend an.

Bei der Facebook-App haben solche Performance-Probleme (Zeichnen des Layouts, Nachladen von News) unter iOS den Ausschlag gegeben, von einer Webapplikation auf eine native Lösung umzustellen. Innerhalb drei Wochen hat sich das Rating von 1.5 auf 4 Sterne im App-Store verbessert[14].

Zugriff auf Sensoren

Für die fiktive Ticketing-App haben wir gefordert, dass das aktuelle Ticket durch eine Schüttelbewegung des mobilen Gerätes zur Ansicht gebracht werden kann. Um eine Schüttelbewegung erkennen zu können, muss der Bewegungssensor in kurzer Folge mehrfach abgefragt werden. Dies gilt auch dann, wenn für die Schüttelbewegung eine vordefinierte Gestenerkennung vorhanden ist. In dem Fall läuft der Sensorzugriff einfach im Hintergrund ab. Aus diesem Grund haben wir eine einfache Test-App entwickelt, welche die maximale Aktualisierungsrate des Bewegungssensors ermittelt (siehe Abb. 4).

In der nativen und der hybriden App wird eine Callback-Methode registriert, die aufgerufen wird, sobald ein Update mit neuen Sensordaten zur Verfügung steht. In Android kann bei der Registrierung des Callbacks die Aktualisierungsfrequenz über einen Parameter spezifiziert werden.

11 jQuery: http://jquery.com/
12 PhoneGap Features: http://phonegap.com/about/feature
13 Browser Performance: https://www.scirra.com/blog/85/the-great-html5-mobile-gaming-performance-comparison
14 Quelle: http://techcrunch.com/2012/09/13/facebook-for-ios-review/

Gerät	nativ	hybrid ohne Modifikation	hybrid mit Modifikation
Samsung Galaxy S (Android 2.3.6)	33 Hz	10 Hz	11 Hz
Samsung Galaxy S (Android 4.1.2 - CM10)	85 Hz	10 Hz	15 Hz
Samsung Nexus S (Android 4.1.2)	45 Hz	10 Hz	—
Samsung Galaxy S III (Android 4.1.1)	155 Hz	10 Hz	15 Hz

Tabelle 2: Ausführung der Test-App auf mehreren Geräten und Betriebssystemversionen

Diesen Parameter haben wir bei unseren Messungen auf SENSOR_DELAY_FASTEST gesetzt, um die Rate nicht künstlich zu beschränken. In PhoneGap ist die API für den Zugriff auf den Beschleunigungssensor äquivalent aufgebaut wie in Android.

Auf der Android-Plattform haben Messungen mit unserer Test-App ergeben, dass auch die Verwendung von Frameworks wie PhoneGap den Performance-Unterschied von HTML5 gegenüber nativen Applikationen nicht markant verringern kann. Der Zugriff auf die Sensor-API ist mittels PhoneGap sieben- bis zehnmal langsamer gegenüber einer nativen Lösung. Die Spannweite hängt damit zusammen, dass PhoneGap die Aktualisierungsfrequenz künstlich auf ca. zehn Aktualisierungen pro Sekunde beschränkt. Einer der Gründe für diese Beschränkung könnte die Schonung der Akkuleistung sein. Durch eine Modifikation im PhoneGap-Framework ist es jedoch möglich, die Aktualisierungsfrequenz als Parameter zuübergeben. So kann die Anzahl Aktualisierungen nochmal gesteigert werden. Selbst mit diesem Trick wird keine ähnlich hohe Aktualisierungsrate wie bei der nativen Applikation erreicht. Die Aktualisierungsrate ist zudem stark von der JavaScript/HTML-Performance des Browsers abhängig und kann deswegen von Gerät zu Gerät variieren, wie in Tabelle 2 zu sehen ist.

Fazit

Die Wahl der adäquaten Kategorie einer mobilen Applikation hängt stark vom Typ der Anwendung, dem gewünschten Funktionsumfang und dem bereits vorhanden Know-how ab. Wird eine Applikation verlangt, die stark auf native APIs oder auf die Hardware zugreifen muss oder deren Performance bei einer aufwendig gestalteten Benutzungsschnittstelle kritisch ist, dürfte eine Webapplikation kaum geeignet sein. Die wohl beste Wahl würde hier eine native App darstellen.

Anders sieht es bei einer Applikation aus, die zur Konsumation von News genutzt wird und die auf möglichst vielen verschiedenen Plattformen lauffähig sein soll. Hier kann eine Webapplikation eine gute Alternative darstellen, sofern keine speziellen Sensoren gebraucht werden. Das grosse Plus der Webapplikation ist die hohe Plattformunabhängigkeit und der geringe Aufwand bei der Verbreitung von neuen Versionen.

Wird eine App gewünscht, welche mit wenig mehr Aufwand auf verschiedenen Plattformen lauffähig ist und dazu noch auf vereinzelte native Funktionen und Sensoren wie zum Beispiel die Kontaktdaten oder die Kamera zugreifen kann, dann ist eine hybride Applikation eine gute Lösung. Dabei gilt es jedoch die Performance im Auge zu behalten, denn laufende Erweiterungen und Anpassungen an Kundenwünsche können zu unerwünschten Nebenwirkungen bei der Leistungsfähigkeit führen.