Gerade durch diese Flexibilität entstehen jedoch auch Risiken, so kann es ohne konkrete Vorgaben durchaus zu einer chaotischen Entwicklung kommen. Bevor die Entwicklung mit Git innerhalb eines Projekts startet, sollten daher einige klare Regeln für die Entwicklung mit dem System aufgestellt und eine grundlegende Integrationsstrategie festgelegt werden.

Durch die native Unterstützung von Git in JDeveloper 12*c* ist es gerade für Projekte im Umfeld der Oracle Fusion Middleware eine interessante Option. Die Entwicklungsumgebung bietet alle grundlegenden Funktionen für die Interaktion sowie die Erstellung entfernter und lokaler Repositories und sollte in jedem Fall als eine valide Option für die Quellcode-Verwaltung eines neuen Projekts in Betracht gezogen werden.

Weitere Informationen

[1] fournova Software GmbH, Learn Version Control With Git: http://www.git-tower.com/learn/ebook/command-line/introduction

[2] Scott Chacon, Pro Git: http://git-scm.com/book

Carsten Wiesbaum
carsten.wiesbaum@esentri.com

Apex und Phonegap? Das kann Apex doch mit HTML5

Daniel Horwedel, merlin.zwo InfoDesign GmbH & Co. KG

Mobile Anwendungen lassen sich sehr gut mit Apex entwickeln. Insbesondere das jQuery-mobile-Framework und das Responsive-Theme geben dem Entwickler einfach zu nutzende Werkzeuge an die Hand, um seine Apex-Web-Anwendung für Mobilgeräte zu optimieren. Sobald aber der Zugriff auf Hardware-Funktionen des Gerätes nötig ist, ist meist der Einsatz eines Frameworks wie Cordova oder PhoneGap erforderlich. Mit zunehmender Verbreitung von HTML5 lassen sich inzwischen aber die meisten Anforderungen auch ohne Verwendung eines zusätzlichen Frameworks mit JavaScript- und HTML5-Bordmitteln umsetzen.

Die Realisierung einer Web-Anwendung mit HTML5 und JavaScript ohne Verwendung eines Frameworks wie Cordova erfordert leider noch sehr intensives Testen auf den verschiedensten Zielplattformen, da die entsprechenden APIs zwar durch W3C standardisiert sind – aber leider durch die Browser-Hersteller unterschiedlich und nicht immer vollumfänglich umgesetzt werden. Dennoch bieten sich damit interessante, vielversprechende und leichtgewichtige Möglichkeiten zur Umsetzung von modernen, Feature-reichen Web-Anwendungen.

GUI: Responsive

Den einfachsten Teil bei der Erstellung einer modernen mobilen Browser-Anwendung mittels Apex stellt die Benutzeroberfläche dar. Hier bietet Apex dem Entwickler zwei verschiedene, je nach konkretem Anwendungszweck unterschiedlich gut geeignete GUI-Konzepte. Mit der jQuery-mobile-basierten Benutzeroberfläche lässt sich mit geringem Aufwand eine für mobile Endgeräte optimierte Benutzeroberfläche erstellen, die die Besonderheiten dieser Geräte (kleines, meist hochauflösendes Display, Touch-Bedienung) besonders berücksichtigt und versucht, ein Nutzererlebnis im Stil einer nativen App nachzubilden.

Alternativ dazu bietet Apex mit dem Theme 25 ein Responsive-GUI, bei dem eine optimale Bedienbarkeit auf verschiedensten Geräten und Bildschirmgrößen zu Lasten einer möglichst nativen Benutzerführung im Vordergrund steht. Eine weitere interessante Möglichkeit zur Umsetzung einer gut bedienbaren, universellen Benutzeroberfläche stellt Twitters Bootstrap-Framework dar – inzwischen sind auch fertige Apex Bootstrap-Themes *(siehe „http://apex-plugin.com/oracle-apex-plugins/themes/css-layout/bootstrap3-theme_396.html")* verfügbar.

Die Hardware

Moderne HTML5-fähige Browser ermöglichen den einfachen Zugriff auf die am häufigsten benötigten Hardware-Funktionen eines Smartphones: den GPS-Sensor, die Kamera und das Mikrofon. Durch die Kombination dieser beiden Funktionen lassen sich auch weiterführende Anwendungen realisieren, etwa ein QR-Code-Scanner innerhalb der Apex-Anwendung.

```
<input type="file" accept="image/*;capture=camera">
<input type="file" accept="video/*;capture=camcorder">
<input type="file" accept="audio/*;capture=microphone">
```

Listing 1

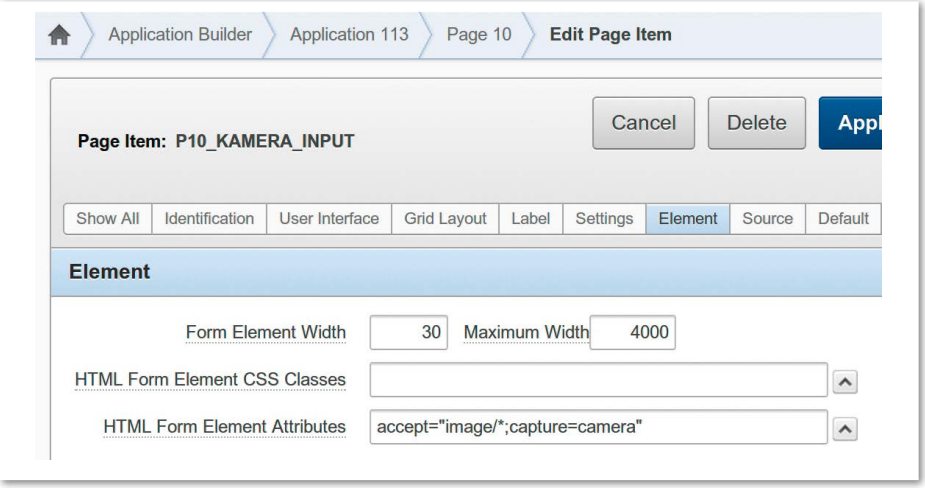

Abbildung 1: Apex-Element-Attribute

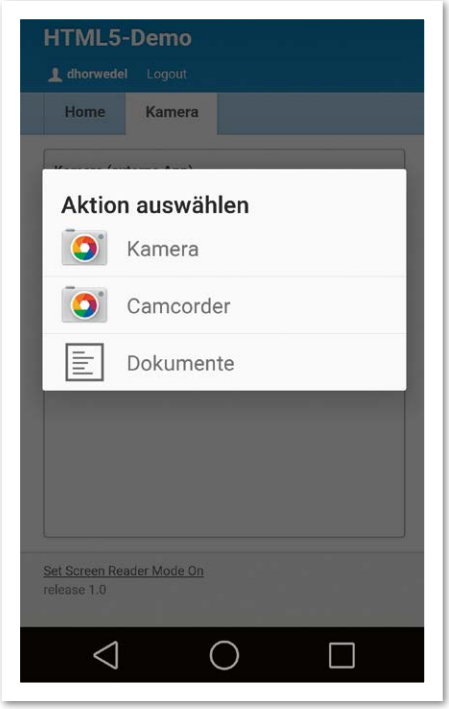

Abbildung 2: Auswahl der App, mit der das Bild aufgenommen werden soll

Zunächst wird der relativ einfach realisierbare Zugriff auf das GPS-Modul betrachtet, der mit dem JavaScript-Funktionsaufruf „navigator.geolocation.getCurrentPosition(callbackFunction);“ umgesetzt ist.

Die aktuelle Position wird nun an die aufzurufende Funktion „callbackFunction“ weitergegeben, die anhand dieser Daten nun zum Beispiel eine Karte darstellen oder eine Dynamic Action aufrufen kann, um die aktuelle Position mithilfe eines PL/SQL-Prozesses an die Datenbank weiterzugeben.

Eine einfache und sehr schnell zu implementierende Möglichkeit für den Zugriff auf die Kamera sowie das Mikrofon bietet das HTML-Media-Capture-API, das das Input-Tag um den Parameter „accept“ mit den jeweiligen Eingabeformaten (Kamera-Snapshot, Video oder Audio) erweitert. *Listing 1* zeigt, wie die HTML-Formular-Elemente für den Upload von Kamera-Bildern, Videos und Audio-Aufnahmen aussehen.

Bei der Verwendung des Apex-Elements „Datei-Upload“ wird durch das Framework ein „input“-Tag mit dem Typ „file“ automatisch generiert – der „accept“-Part wird einfach bei den Formular-Attributen dieses Apex-Elements angegeben *(siehe Abbildung 1)*.

Beim Klicken auf das Input-Element öffnet sich daraufhin die für die jeweilige Aktion vom Betriebssystem vorgesehene App – für den Fall, dass mehrere Apps für diesen Zweck installiert sind, wird dem Benutzer zunächst eine Auswahlmöglichkeit angeboten. Wenn das Medienelement nach dem Upload serverseitig weiterbearbeitet oder nur gespeichert werden soll, lässt sich mit dieser Methode schnell und einfach ein Ergebnis erzielen – das hochgeladene Medien-Element wird ganz einfach wie ein Datei-Upload innerhalb von Apex weiterverarbeitet – wahlweise in einer BLOB-Spalte einer eigenen Tabelle oder über die Apex-eigene Meta-Tabelle „WWV_FLOW_FILES“ *(siehe Abbildung 2)*.

In vielen Anwendungsfällen soll das hochgeladene Element aber nicht nur im Hintergrund gespeichert, sondern auch direkt weiterverarbeitet werden, denn bei einem QR-Scanner ist es nicht zielführend, wenn der Benutzer erst auf eine serverseitige Bearbeitung des hochgeladenen Scans warten muss. In solchen Fällen ist es zweckmäßig, dem Benutzer direkt das Kamerabild wie bei einem klassischen Kamera-Sucher in der Anwendung zu präsentieren und bei Erkennung eines Codes sofort Rückmeldung zu geben.

Dank der neuen Medien-Schnittstellen von HTML5 lassen sich Kamera und Mikrofon nun gänzlich ohne Plug-ins wie Flash oder Silverlight direkt innerhalb der Anwendung verwenden und beispielsweise mittels JavaScript im Browser weiterverarbeiten. Seit den ersten Entwürfen der HTML5-Spezifikation gab es einige Ansätze, diese Funktionen bereitzustellen – durchgesetzt hat sich die Web-Real-Time-Communications-Schnittstelle (WebRTC) mit der Funktion „navigator.getUserMedia(quelle, callbackFunction, errorFunction)“. Als Parameter wird der Funktion mitgegeben, welche Medienquellen (Audio, Video) verwendet und an welche Funktion die Daten übergeben werden sollen sowie wie mit Fehlern umzugehen ist.

QR-Scanner

Einen praktischen Anwendungsfall für das Kamera-API stellt die bereits angesprochene Implementierung eines QR-Code-Scanners dar. Diese gliedert sich in die drei Teilbereiche „Bildquelle“, „QR-Erkennung“ und „Auswertende Funktion“. Zunächst wird mit „getUserMedia“ der Video-Stream der Kamera abgegriffen, um eine Live-Vorschau zu ermöglichen und das aufgenommene Bild mittels QR-Erkennung zu analysieren.

Die permanente Analyse des Video-Streams erfolgt mithilfe einer JavaScript-Portierung der im Android-Umfeld bewährten Open-Source-QR-Decoding-Library „Zxing“, die bei einem erfolgreich erkann-

Abbildung 3: JavaScript/HTML5-basierter QR-Code-Scanner in einer Apex-Anwendung

ten QR-Code die auswertende Funktion (sogenannte „Callback-Function") aufruft und als Parameter den decodierten Wert übergibt. Als Callback-Function bietet sich eine JS-Funktion an, die zunächst ein ausgeblendetes Apex-Seitenelement mit dem ermittelten Wert füllt und anschließend eine „Dynamic Action" aufruft, die auf das Event „Custom" reagiert und die weitere Verarbeitung übernimmt *(siehe Abbildung 3)*.

Lokale Speicherung von Daten/ Offline-Betrieb

Web-Anwendungen haben gegenüber nativen Apps normalerweise einen großen Nachteil, da sämtliche benötigten Ressourcen vom Web-Server geladen werden müssen – und das bei jedem Aufruf der Seite erneut. Dieser Problematik lässt sich im Bereich von statischen Regionen sehr gut mit Apex-Bordmitteln begegnen – hier genügt es, auf Regions-Ebene das Region-Caching zu aktivieren. Sobald allerdings interaktive Reports oder Regionen mit dynamischen Inhalten im Einsatz sind, hilft der Apex-interne Caching-Mechanismus nicht mehr weiter.

Für das Caching statischer Elemente bietet HTML5 das Offline-Application-Caching-API, das anhand einer Manifest-Datei detailliert festlegt, welche Elemente der Anwendung wie lange unter welchen Bedingungen auf dem Endgerät gespeichert werden sollen, sodass sich ein erneutes Laden erübrigt und das Datenvolumen geschont wird.

Beim Einsatz des „AppCache"-Features mit „Shared Components" in Apex gibt es derzeit leider noch eine schwerwiegende Einschränkung, die den Aufwand für die Nutzung erhöht: Die Manifest-Datei benötigt statische Pfade zu den Ressourcen, die bei Nutzung der „Shared Components" nicht automatisch existieren. Sofern es die System-Architektur erlaubt, ist hierbei die Auslieferung der statischen Dateien über das Image-Directory „/i/" vorzuziehen, wodurch statische Dateipfade zur Verfügung gestellt werden, die wiederum im „AppCache"-Manifest nutzbar werden.

Browser-Storage

Für das Caching statischer Ressourcen ist der AppCache hervorragend geeignet. Je nach Anwendungszweck kann es allerdings erforderlich sein, Daten lokal auf dem Endgerät zu speichern. Dabei muss zwischen Anwendungen unterschieden werden, die zwar prinzipiell auf den Online-Betrieb ausgelegt sind, bei denen aber bestimmte Daten lokal gespeichert werden sollen, und Anwendungen, die einen autarken Offline-Betrieb ermöglichen.

Ein häufiger Anwendungsfall für das Caching von Daten in einer Online-Anwendung ist das Speichern großer Objekte wie Produktbilder oder PDF-Dokumente, die in regelmäßigen Intervallen aktualisiert werden können, jedoch nicht bei jedem Aufruf des zugehörigen Datensatzes aus der Oracle-Datenbank gelesen und über das Netzwerk geladen werden müssen. Hier bietet es sich an, zur Reduzierung des Datenvolumens und damit einhergehend der Erhöhung der Performance diese Daten in einer lokalen Datenbank auf dem mobilen Endgerät zu speichern.

HTML5 bietet aktuell zwei verschiedene unterstützte Varianten an: WebStorage ermöglicht die einfache Speicherung von Werten im Key-Value-Format ohne jegliche Relationen, unterstützt allerdings ausschließlich String-Werte. Mithilfe der JQuery-Funktionen „JSON.stringify()" lassen sich jedoch BLOB-Werte in Strings umwandeln und mit „JSON.parse()" wieder in das jeweilige Ursprungsformat bringen – besonders Performance- und Speicher-optimiert ist diese Vorgehensweise allerdings nicht.

Sollen aufwändigere Strukturen abgebildet werden, empfiehlt sich die Verwendung von „IndexedDB", das die Daten ebenfalls im Key-Value-Format speichert, den Zugriff allerdings per Index ermöglicht und dadurch einen enormen Performance-Vorteil bietet. Zudem ermöglicht es den Einsatz weiterer Datentypen wie BLOBs, wodurch das Ressourcen-intensive Parsen der im String gespeicherten Werte bei „localStorage" entfällt. Demgegenüber steht allerdings ein höherer Aufwand bei der Nutzung des API und der Implementierung der Logik. Schlussendlich muss man anhand des konkreten Anwendungsfalles entscheiden, welche Technik besser geeignet ist.

Auch bei der Entwicklung einer offline nutzbaren Anwendung bietet sich der Einsatz von „localStorage" oder einer IndexedDB an. Hierbei muss allerdings berücksichtigt werden, dass eine Apex-Anwendung grundsätzlich darauf ausgelegt ist, online verwendet zu werden – dies beginnt schon bei der Authentifizierung und zieht sich durch die gesamte Architektur typischer Apex-Anwendungen. Soll eine Apex-Anwendung offline zur Verfügung stehen, bietet es sich an, eine separate, auf statischen HTML-Seiten basierende und im Funktionsumfang reduzierte Offline-Webanwendung anzubieten, die ihre Daten aus dem Browser-Storage sowie die Anwendungsdaten aus dem AppCache bezieht.

In der Online-Version der Anwendung wird dann die entsprechende Synchronisations-Logik implementiert, mithilfe derer die offline bearbeiteten Dateien mit der Oracle-Datenbank abgeglichen und die aktuelle Version der Daten auf dem Mobilgerät gespeichert wird. Das Umschalten zwischen den Anwendungen kann abhängig vom aktuellen Netzwerk-Status durch das AppCache-Manifest erfolgen. Wenn der Nutzer online ist, wird die gesamte Apex-Anwendung geladen; sobald er offline ist, kommt die gecachete Offline-Version zum Einsatz.

Weitere Schnittstellen

Die HTML5-Spezifikationen sehen noch einige weitere vielversprechende APIs vor, die bislang allerdings noch nicht oder nur in sehr wenigen Browsern umge-

setzt sind, wodurch der praktische Nutzen für die Verwendung dieser Schnittstellen derzeit noch eher gering ist. So lässt sich zum Beispiel mit dem Battery-API (derzeit unterstützt von Firefox und Chrome) das Verhalten der Anwendung anhand des Batterie-Zustandes festlegen, um Ressourcen-hungrige GUI-Animationen oder durch das Vibration-API (unterstützt von Firefox, Chrome, Opera) erzeugte Vibrations-Benachrichtigungen bei einem niedrigen Ladezustand zu deaktivieren.

Mit dem Ambient-Light-API (derzeit nur von Firefox unterstützt) kann auf die Helligkeit der Umgebung reagiert werden, die durch den in vielen Smartphones verbauten Umgebungslicht-Sensor ermittelt wird; so kann zum Beispiel ein dunklerer und damit blendfreier Nachtmodus der Anwendung realisiert werden.

Eigene Kontextmenü-Einträge ermöglicht das Context-Menu-API, das derzeit leider nur durch Firefox unterstützt wird. Ebenfalls nur durch diesen Browser wird das Proximity-API zur Verfügung gestellt, um den Abstand des Nutzers zum Gerät zu ermitteln. Inwieweit die Browser-Hersteller allerdings diese APIs implementieren, lässt sich schwer vorhersehen. Deshalb ist derzeit davon abzuraten, diese Features außerhalb eines geschlossenen Nutzerkreises mit homogener Browser-Verwendung einzusetzen.

Fazit

Die Kombination von HTML5, JavaScript und Apex bietet viele spannende Möglichkeiten, Funktionen einer nativen Smartphone-Anwendung mit deutlich reduziertem Aufwand und ohne die Verwendung von Kompatibilitäts-Frameworks umzusetzen, die insbesondere im Performance-Bereich einen nicht zu unterschätzenden Overhead generieren.

In einigen Bereichen bestehen derzeit noch Einschränkungen bezüglich der Umsetzung der Features durch die jeweiligen Browser-Hersteller, weshalb unbedingt auf den unterschiedlichen Funktionsumfang der Ziel-Plattformen Rücksicht genommen werden muss. Längerfristig ist aber davon auszugehen, dass die in der HTML5-Spezifikation festgelegten Schnittstellen in allen gängigen Browsern umgesetzt sind und somit ohne Einschränkungen Geräte- und Browser-unabhängig zum Einsatz kommen können. Für die Umsetzung der gängigsten Anforderungen an mobile Anwendungen ist der Einsatz von PhoneGap/Cordova oder ähnlichen Frameworks heutzutage nicht mehr zwingend erforderlich.

Daniel Horwedel
daniel.horwedel@merlin-zwo.de