Methoden komfortabler Datenbank-Anbindung für Mobile Devices

Torsten von Osten
pdv Technische Automation + Systeme GmbH
Hamburg

Schlüsselworte:

Mobile Devices, Datenbank, Oracle Application Express (Apex) -Listener, RESTful-Webservices

Einleitung

Es existieren viele Möglichkeiten, eine datenbankbasierende Applikation fit für den Einsatz auf Mobile Devices zu machen. Um dieses zu realisieren, kann man unterschiedliche Ansätze verfolgen. Angefangen von einer Standard-Webapplikation bis hin zu einer Device-spezifischen nativen Applikation.

Abhängig vom gewählten Ansatz und den Fähigkeiten der Devices sowie der Einsatzumgebung ergeben sich auch unterschiedliche Methoden bezüglich des Datenbankzugriffes, deren Ansatz nativ und geräteabhängig oder aber generell und allgemeingültig sein kann.

Im Folgenden sollen daher die unterschiedlichen Möglichkeiten mit ihren Stärken und Schwächen beleuchtet werden. Dabei soll insbesondere am Beispiel des Oracle Apex Listener und dessen Unterstützung für RESTful-Webservices gezeigt werden, dass auch ein allgemeiner Ansatz einen durchaus komfortablen Datenbankzugriff ermöglichen kann, ohne den Implementierungsaufwand wesentlich zu steigern.

Mobile Devices - ein heterogenes Umfeld

Wer sich mit der Anwendungsimplementierung auf mobilen Endgeräten beschäftigt, merkt schnell, dass er sich in einem weitaus heterogeneren Umfeld als im Desktop-Bereich bewegt. Während man sich im Desktop-Bereich hauptsächlich auf Windows-Plattformen beschränken kann, trifft man im mobilen Bereich, neben den starken Unterschieden in Ausstattung und Leistungsfähigkeit der Hardware, auch auf diverse Betriebssystem-Plattformen, die ihrerseits diverse Einschränkungen und Vorgaben mit sich bringen. So legt das Betriebssystem auch fest, welche Programmiersprachen und welche Frameworks bzw. Drittanbieter-Software zur Verfügung stehen. Selbst die als plattformunabhängig geltenden Sprachen stehen nicht auf allen prominenten mobilen Betriebssystemen zur Verfügung.
Daher ist hier die Entscheidung, ob man für die Entwicklung einen nativen Ansatz oder einen allgemeineren Ansatz wählt, von größerer Bedeutung. Wer sich auf die Unterstützung nur eines Device-Typs beschränken oder sich diesen sogar aussuchen kann, wählt eher einen nativen Ansatz. Wer ein breiteres Feld abdecken muss und damit vielleicht auch zukunftssicherer ist, versucht wohl eher einen generelleren Ansatz zu finden, um Aufwand und Kosten für Entwicklung und späteren Wartung zu senken.

Neben den Beschränkungen, die für das jeweilige Device und Betriebssystem gelten, hat auch der gewählte Entwicklungsansatz Auswirkungen auf die Möglichkeiten und Arten des Zugriffs auf eine externe zentrale Datenbank.

Entwicklungsmethoden für Mobile Devices

Beim nativen Ansatz bedient man sich einer vom Hersteller für das spezielle Device vorgesehenen Programmiersprache. Dabei erhält man Zugriff auf alle Funktionen wie z.B. die Kamera, Bewegungssensoren, Ortungs- und Kommunikationstechnik des Gerätes und des Betriebssystems mit seiner Standardsoftware (z.B. Adressbuch). Des Weiteren kann sich der Entwickler der reichhaltigen und speziell an die Ein- und Ausgabe-Fähigkeiten angepassten GUI-Elemente bedienen. Daraus ergibt sich für den Anwender eine meist flüssige und intuitive Bedienung mit den für ihn gewohnten Eingabemöglichkeiten. Gegebenenfalls ist neben dem nativen Zugriff auf eine lokale Datenbank auch ein nativer externer Datenbankzugriff möglich.

Der wohl generellste Ansatz ist die Entwicklung einer Web-Applikation, die auf einem Applikations-Server läuft. Im besten und einfachsten Falle existiert diese Anwendung schon für den Desktopbereich und die Webseiten können fehlerfrei von den verwendeten Mobile Devices dargestellt und bedient werden. In den meisten Fällen wird zumindest eine Anpassung an der Rendering-Engine notwendig sein, um einen akzeptablen Bedienkomfort auf den limitierten Ein- und Ausgabemöglichkeiten der Endgeräte zu erreichen. Verwendet man „Java Server Faces“, dann könnte man z.B. auf eine Faces - Bibliothek wechseln, die entsprechend angepasste Darstellungs-Komponenten enthält. Die Anwendung an sich läuft weiterhin geräteunabhängig auf dem Applikationsserver ebenso wie der Datenbankzugriff.
Diese Anwendung braucht aber auch eine permanente und stabile Konnektivität zum Applikationsserver, da nahezu jede Benutzeraktion auch eine Interaktion mit dem Server notwendig macht.
Der Zugriff auf Gerätefunktionalitäten bleibt ebenfalls weitgehend verwehrt. Die Flüssigkeit der Bedienung hängt stark von der Leistungsfähigkeit des Browsers und des Endgerätes ab, auch wenn häufig versucht wird, die native GUI nachzuempfinden.

Moderne mobile Browser mit HTML5 und aktueller JavaScript-Unterstützung verbessern die Möglichkeit der webbasierten GUI, gehen aber auch noch ein Stück weiter. Hier wird im extremsten Fall einmalig die gesamte Anwendung als Webseite inklusiv aller Quellen geladen und als JavaScript-Anwendung im Browser des Endgerätes, unabhängig von einem Applikationsserver, ausgeführt.
JavaScript Frameworks wie Sencha Touch oder jQuery Mobile liefern dem Entwickler entsprechend vorgefertigte Funktionen und GUI-Elemente. Dadurch wird die Anwendung für den Benutzer meist komfortabler und flüssiger zu bedienen, jedoch muss sich die JavaScript-Anwendung nun auch um die Zugriffe auf zentrale Datenbanken selber kümmern, ohne dass sie auf einen nativen Datenbanktreiber zurückgreifen kann.

Einen weiteren Ansatz liefern Entwicklungstools, die durch eine generative Lösung aus dem generellen, webbasierten Ansatz native Applikationen generieren. Auch hier werden von den Tools noch nicht alle mobilen Plattformen unterstützt, aber meistens zumindest die gängigen. Bekannte Vertreter dieser Art von Tools sind z.B. Cocona, PhoneGap oder Titanium Mobile.; letzteres wird auch in der Demo zu diesem Vortrag verwendet. Das Prinzip ist bei den meisten dieser Tools ähnlich. Die Entwickler schreiben ihre Anwendung in JavaScript und können dabei auf eine JavaScript-Bibliothek des Tool-Herstellers zugreifen, die zur Laufzeit auf die nativen GUI-Komponenten und Funktionen des Endgerätes verweisen. Als Laufzeitumgebung für die Applikationslogik dient die

JavaScript-Engine des mobilen Browsers, die von einer nativen Wrapper-Applikation aufgerufen wird und die nativen Komponentenzugriffe abfängt und entsprechend nativ umsetzt.

Auch in diesem Ansatz muss sich die Mobile-Applikation selbst um den externen Datenbankzugriff kümmern. Durch die APIs wird auch hier nur eine lokale Datenbank unterstützt. Selbst wenn eines der Endgeräte die Verwendung eines Datenbank-Treibers unterstützt, ist die Einbindung schwierig, da eben auch nicht plattformunabhängig.

Datenbankzugriffsmethoden

Ein nativer Datenzugriff über den Oracle NET -Treiber dürfte mangels Unterstützung auf den meisten mobilen Devices keine Option sein. Anders sieht das mit dem Oracle JDBC-Treiber aus. Er ist zumindest auf den Endgeräten verwendbar, auf dem eine Java-Runtime mit JDBC-Unterstützung vorhanden ist eine Alternative. Dieses ist zum Beispiel unter Android der Fall.

Auch wenn die Verwendung eines nativen Datenbanktreibers wohl die mächtigste und komfortabelste Methode zur Datenbankanbindung ist, scheitert ihr Einsatz trotz Unterstützung durch das mobile Endgerät häufig an der Infrastruktur, in der das Endgerät betrieben wird. Ein mobiles Gerät befindet sich im besten Fall im firmeninternen WLAN, in den häufigeren Fällen jedoch im Internet. Aus Sicherheitsgründen wird und sollte die DB in beiden Fällen nicht direkt erreichbar sein. Zugriffe auf die Datenbank dürfen aus diesen Netzen daher meist nur über eine Middleware erfolgen. Im Falle der traditionellen Web-Applikation ist dieses bereits gegeben. Alle anderen Anwendungsarten benötigen in diesem Fall auf der Middleware einen Service, der sich nur um den gekapselten sicheren Datenbankzugriff kümmert.

Einen solchen Service bietet z.B. die Oracle Database Lite, welche aus zwei Komponenten besteht. Einer Client Datenbank, die auf dem mobilen Device läuft und einer Server Komponente, die als Applikationsserver dient und die Synchronisation der Client Datenbank mit einer Oracle Datenbank auf einem Server ermöglicht. Seit Kurzem bietet Oracle auch die Möglichkeit, die auf mobilen Devices weit verbreitet SQLite DB mit dieser Technik zu synchronisieren. Damit sollten jetzt mehr Plattformen unterstützt werden. Für die Synchronisation ist dann jedoch noch eine Mobile Client Komponente notwendig, die wiederum nicht alle Plattformen unterstützt, wie z.B. das beliebte iOS. Zudem fallen zusätzliche Lizenzkosten an.

Eine so allgemeine Lösung, dass sie nahezu auf allen Endgeräten funktioniert, ist die Verwendung von SOAP - Webservices. Der Datenaustausch zwischen Client und Middleware erfolgt über HTTP(S) und einem speziellen XML-Protokoll. Auf der Middleware müssen dazu Dienste implementiert werden, die die einzelnen Datenzugriffe auf die Datenbank realisieren. Die Allgemeinheit dieser Lösung macht ihr Protokoll jedoch auch komplex, worunter nicht nur der Komfort in der Nutzung leidet und damit auch der Aufwand in der Realisierung steigt, sondern auch der Übertragungsoverhead relativ hoch ist. Hinzu kommt, dass die Verarbeitung der XML-Strukturen auf den mobilen Endgeräten, deren Speicher und CPU-Ressourcen begrenzt sind, schnell zu einer schlechten Performance der Anwendung führt.

Apex Listener und RESTful Webservices

Ein ebenfalls allgemeiner Ansatz, der leichgewichtiger ist als der SOAP-Ansatz, ist die Verwendung von RESTful-Webservices. Des Weiteren ist die Ressourcen(Entity) -orientiert Herangehensweise näher an der Arbeitsweise relationaler Datenbanken angesiedelt. Bei RESTful Webservices wird auf

Ressourcen (Entities) über die bereits im HTTP-Protokoll definierten Methoden GET, POST, PUT und DELETE zugegriffen.

Der gleiche HTTP-Aufruf mit der URI „http://localhost/ipcon/aktivitaet“ würde, gesendet mit der „GET“ –Methode, laut Definition einen Datensatz aus einer Datenbank-Tabelle „Aktivitaeten“ zurückliefern, mit „POST“ einen neue Aktivität Eintragen, mit „PUT“ updaten und mit „DELETE“ löschen.

Parameter können dabei in der URI dem HTTP-Header oder im HTTP-Body mitgeliefert werden. Für den Transport von Parametern und Ergebnissen im HTTP-Body könnte man nun auch wieder XML verwenden. Aus den bereits genannten Gründen empfiehlt es sich jedoch, auf das schlankere und einfachere JSON-Format (JavaScript-Object-Notation) zurückzugreifen. Es besteht aus einfachen ‚key‘:‘value‘ –Werten, die in Listen geschachtelt werden können. Die Einfachheit wird sich aber auch dadurch erkauft, dass keine Typen- und Struktur-Prüfungen oder Struktur-Definitions-Dateien existieren. Der Ansatz entspricht mehr der Methode „Convention over Configuration“.

Ähnlich wie bei den Webservices müsste der eigentliche Datenbankzugriff hinter jeder Ressource und Methode auf der Middleware implementiert werden. Hier springt nun der Oracle Apex Listener ein. Er kann als Standalone Lösung oder in einem J2EE Web-Server, wie z.B. dem Apache Tomcat oder dem Oracle Weblogic Server, betrieben werden. Er erlaubt es weitgehend deklarativ REST-Ressourcen und die dazu gehörigen Methoden basierend auf SQL –Anweisungen oder PL/SQL –Blöcken zu definieren. Dazu erstellt man in der Management-Oberfläche ein neues Ressource-Template und fügt die gewünschten Ressource-Handler für GET, POST, etc. hinzu und definiert darin die entsprechenden SQL-Anweisungen.

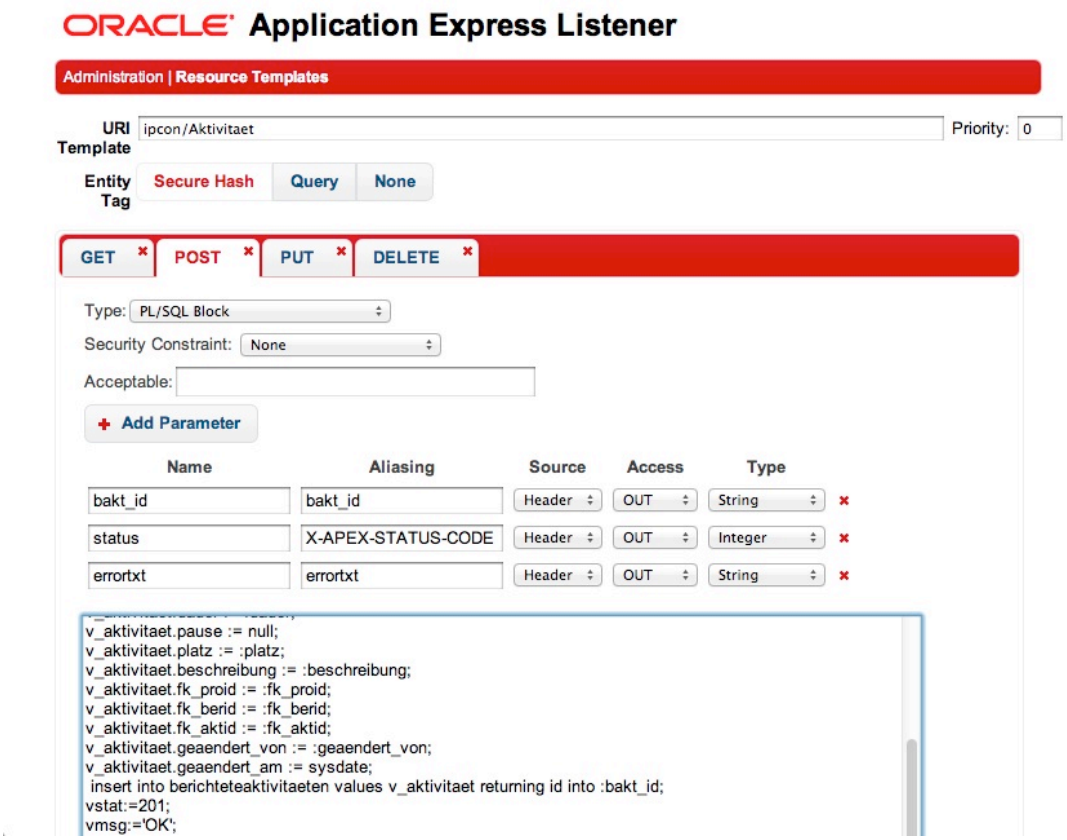

Abb. 1: Apex Listener Ressourcen Definition

Die Ergebnisse eines Selects in einer GET-Methode werden vom Apex-Listener automatisch in das JSON-Format konvertiert und entsprechend zurückgeliefert. Dabei ist zu beachten, dass key – Attribute vom Apex Listener immer klein geschrieben werden und wenn der Wert des Datenbankfeldes „NULL“ ist, wird auch kein JSON Wertepaar zurückgeliefert.

Insert oder update- Statements schreibt man als PL/SQL-Block im POST bzw. PUSH –Handler. Damit man bei POST und PUSH nicht alle Parameter manuell im Header oder der URI definieren und senden muss, bietet der Apex-Listener ebenfalls die Möglichkeit, alle Parameter im Body der Anfrage im JSON-Format zu senden. Diese Parameter stehen dann ohne weiter Deklaration im PL/SQL Block zur Verfügung. Neben den selbstdefinierten Parametern stehen auch ein paar Standradparameter zur Verfügung, über die man einen individuellen HTTP-Status-Code zurückgeben kann, wie zum Beispiel Code „201“, der laut HTTP für „Resource created“ steht.

Auf ausufernde PL/SQL-Blöcke sollte man an dieser Stelle aber verzichten und so etwas lieber in eine Stored Procedure verlagern, da das Debuggen sich an dieser Stelle als etwas schwierig gestaltet.

```
//Öffnen einer sicheren HTTP-Verbinung für eine POST-Methode
var client = Titanium.Network.createHTTPClient();
client.open('POST', 'https://localhost/apex_test/ipcon/Aktivitaet');
client.setRequestHeader('Authorization', 'Basic '
                        + Titanium.Utils.base64encode(user + ':' + password));
//Parameter für den Insert in JSON-Format
client.setRequestHeader('Content-Type', 'application/json');
activity = { "tag": "26",
             "dauer":"4",
             "beschreibung":"Ein insert Test über REST."
           };
//Methode zum Auswertender Antwort definieren
client.onload = function() {
  alert(this.status);
  alert(this.getResponseHeader('Errortxt'));
}
//Anfrage senden
client.send(JSON.stringify(activity));
```

Abb. 2: Aufrufbeispiel eines RESTful Webservices in Javascript unter Appcelerator Titanium Studio

ETag- Unterstützung zur Versionsidentifikation von Ressourcen

Der Apex Listener unterstützt in Teilen das Handling mit ETags. Ein ETag ist eine eindeutige Versionskennung für eine aktuelle Ressource und kann im HTTP-Header beim „GET“ automatisch mitgeliefert werden. Diese Kennung besteht entweder aus einem Hash des gesamten Abfrageergebnisses oder einer eigenen selbstdefinierbaren Query, wenn z.B. bereits ein eindeutiges Versionskennzeichen im Datensatz vorhanden ist. Das ETag kann nun dazu verwendet werden, um zu prüfen, ob sich eine Ressource auf dem Server geändert hat oder um einen Optimistic Locking-Mechanismus abzubilden.

Leider ist die Oracle-Dokumentation nicht sehr ausführlich hinsichtlich der automatisch unterstützen ETag - Funktionalitäten des Apex Listener. Nachweislich wertet der Apex Listener bei der „GET“ Methode ein im Request-Header per Attribut „If-None-Match“ mitgeliefertes ETag aus, in dem das bei

der neuen Abfrage ermittelte ETag mit dem übersendeten ETag verglichen wird und bei Übereinstimmung anstatt der Daten lediglich der HTTP-Status „302“ – „Not Modified“ zurückgeliefert wird. Für eine automatische Unterstützung des Optimistic Locking konnten bisher noch keine Anhaltspunkte gefunden werden, welches jedoch bei eine entsprechenden Konvention der Parametrisierung durchaus möglich wäre.

Security mit dem Apex Listener

Eine Authentifizierung über Oracle Single Sign-On ist eingebaut. Ansonsten empfiehlt es sich, die Container-Managed-Security des J2EE-Servers oder die Authentifizierung über einen Apache und eines dessen Module durchzuführen. Der Apache kann man dann als Reverse-Proxy konfigurieren und auch gleich die SSL Verschlüsselung übernehmen lassen. Damit steht ein weites Feld an Authentifizierung-Methoden zur Verfügung.

Die Anmeldung an der DB durch den Apex-Listener erfolgt wie bei Web-Applikationen üblich über einen zentralen Applikations-Benutzer. Um mehrere Anwendungen mit unterschiedlichen Datenbankbenutzern zu betreiben, müssen auch mehrere Instanzen des Apex Listener „deployed“ werden.

Abschließend lässt sich festhalten, dass es in speziellen Umfeldern möglich ist, mit einem nativen Datenbanktreiber auch direkt auf eine Oracle Datenbank zuzugreifen. Für die Mehrheit der Anwendungen für die dieses keine Option ist, ist die Kombination aus RESTful-Webservice und dem Apex Listener von Oracle eine durchaus schlanke und allgemeingültige Alternative, auch wenn sie sicherlich nicht so mächtig ist wie einer nativer Datenbanktreiber.

Kontaktadresse:

Torsten von Osten
pdv Technische Automation + Systeme GmbH
Dorotheenstraße 64
D-22301 Hamburg

Telefon: +49 (0) 40-69213 266
Fax: +49 (0) 40-69213 278
E-Mail vonosten@pdv-tas.de
Internet: www.pdv-tas.de