Viele Unternehmen vollziehen derzeit den Schritt von klassischen Desktop-PCs und Notebooks auf mobile Geräte als Ergänzung oder gar als Ersatz für die herkömmliche Arbeitsumgebung. Apps auf mobilen Geräten bieten weitreichendere Einsatzmöglichkeiten als stationäre Anwendungen. Deshalb sind die Anforderungen an mobile Apps in Unternehmen vielfältig und hängen stark von den jeweiligen Einsatzgebieten ab.

Ansätze für die Entwicklung von mobilen Business-Apps

Jörg Bredlau, OPITZ CONSULTING GmbH

Damit eine mobile Lösung die Geschäftsprozesse eines Unternehmens später wirklich optimal unterstützt, sollte ein Entwicklungsteam seine technologischen Ansätze und Werkzeuge im Vorfeld sorgfältig aussuchen und sich Fragen stellen wie: „Für welche Ziel-Plattform soll die Anwendung entwickelt werden?“, „Soll die App lediglich eine Plattform unterstützen?“, „Soll die Lösung auf unterschiedlichen Geräten laufen?“, „Benötigt die Anwendung Zugriff auf Geräte-spezifische Hardware?“, „Wie soll die Verteilung der Applikation auf die Geräte erfolgen?“ und „Wie soll aus der mobilen App auf Unternehmensdaten und Geschäftsprozesse zugegriffen werden?“. Der Artikel befasst sich mit diesen sowie anderen Fragen und gibt damit kleine Entscheidungshilfen für die Auswahl der passenden Entwicklungsstrategie.

Mobile Welten

Auf dem Markt der mobilen Geräte und Betriebssysteme haben sich Apple mit iOS und Google mit Android endgültig durchgesetzt. Microsoft versucht mit dem gerade erschienenen Windows Phone 8 zu den beiden Marktführern aufschließen. Bei der Entscheidungsfindung zu einer Ziel-Plattform muss man bedenken, dass in jeder Umgebung vollkommen unterschiedliche Design-Konzepte, eigene Programmiersprachen, Werkzeuge und Hardware-Umgebungen eingesetzt werden. Dazu kommen verschiedene OS-Versionen mit unterschiedlichen Features und APIs sowie variierende Geräte-Eigenschaften.

Mobile Web-Apps sind eine vielversprechende Alternative zu nativen Anwendungen, um der Plattform-Fragmentierung gerecht zu werden. Hier kommen Web-Technologien wie HTML5, CSS und JavaScript zum Einsatz. HTML5- und JavaScript-Frameworks erlauben dem Entwickler den Zugriff auf verschiedene Geräte-Funktionen und helfen, das „Look & Feel“ der nativen Systemumgebung nachzuempfinden.

Für Web-Apps sprechen vor allem die Flexibilität in der Entwicklung, die einfache Distribution und ihre Plattform-Unabhängigkeit. Auch dass mit einer gemeinsamen Code-Basis verschiedene Plattformen abgedeckt und damit Kosten erheblich reduziert werden können, ist für viele Unternehmen ein wichtiges Argument. An ihre Grenzen stoßen HTML5-Web-Anwendungen allerdings, wenn es um die größtmögliche Performance der App und die optimale Ausnutzung der Geräte-Funktionen geht. Da es im B2B-Bereich jedoch meist weniger auf ultimative grafische Feinheiten ankommt, überwiegt für die Entscheider der Vorteil der Cross-Plattform-Entwicklung: Mit diesen Applikationen sind sie in der Lage, eine möglichst große Bandbreite an Geräte-Plattformen abzudecken und eine maximale Anzahl an Usern zu erreichen.

Mobile Arbeitsabläufe sind abhängig vom Arbeitsumfeld und müssen deshalb systematisiert und in enger Zusammenarbeit mit den Endanwendern hinsichtlich Benutzer-Führung und -Bedienung optimiert werden. Bei einer Inhouse-Entwicklung sollten die Projektleiter das vorhandene Wissen der Entwickler über Programmiersprachen und Entwicklungsumgebungen der Ziel-Plattform berücksichtigen, um Einarbeitungszeiten zu reduzieren. In der mobilen Anwendungsentwicklung gibt es drei verschiedene Entwicklungsansätze: native Apps, Web-Apps und hybride Apps (siehe Abbildung 1).

Beim Development von Web-Apps und hybriden Apps stehen dem Entwickler Web-Techniken wie HTML5, CSS3 und JavaScript zur Verfügung. Der Web-Ansatz bietet wie die native Entwicklung eine große Auswahl an Bibliotheken und Frameworks, die unterschiedliche Bereiche des Entwicklungsprozesses abdecken. Einige offerieren Unterstützung durch alle Anwendungs-Stacks wie Benutzeroberfläche, UI-Interaktion, Controller-Komponenten und Data Binding. Andere bedienen nur einen Teilbereich dieser Punkte.

Native Entwicklung

Bei der nativen Entwicklung arbeitet der Programmierer direkt mit einem Software Development Kit (SDK) in der spezifischen Programmiersprache und den Tools für die Ziel-Plattform. Eine native App zeichnet sich dadurch aus, dass sie lokal auf dem mobilen Gerät installiert ist und direkt vom Betriebssystem des Geräts ausgeführt wird. Die App läuft ausschließlich auf der jeweiligen Plattform und muss für jede Geräte-Plattform entwickelt und gepflegt werden. Das native SDK unterstützt optimal die jeweiligen Geräte-Eigen-

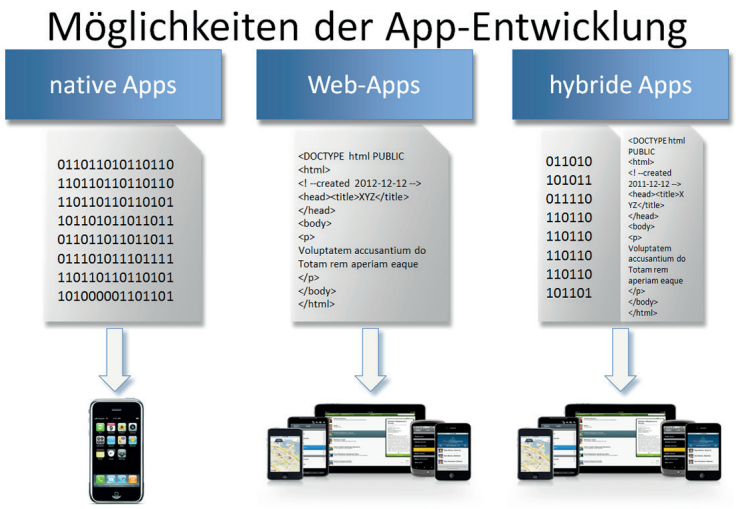

Abbildung 1: Mobile Entwicklungsansätze

schaften, kann auf spezifische Geräte-Funktionen zugreifen und bietet somit die beste Integration mit dem mobilen Device. Der Nachteil besteht in dem großen Entwicklungsaufwand für die Unterstützung mehrerer Plattformen.

Derzeit kann nur eine native App das volle Potenzial des jeweiligen Geräts hinsichtlich Usability, User Experience und Performance ausschöpfen. Wer also die Möglichkeiten der Zielgeräte optimal ausnutzen will, für den geht kein Weg an der nativen Entwicklung vorbei. Weitere Informationen zu Android- und iOS-SDK findet man unter [1] und [2].

Mobile Web-Apps

Mit diesem Ansatz erstellt man Web-Anwendungen, die sich vom „Look & Feel" her fast wie native Anwendungen anfühlen, aber in einer Browser-Umgebung ablaufen. Die Browser-Controls werden ausgeblendet, damit der Anwender das Gefühl bekommt, mit einer nativen App zu arbeiten. Mobile Web-Apps werden üblicherweise mit HTML5, JavaScript und CSS3 erstellt. In diesem Bereich kommen JavaScript-Frameworks wie jQuery Mobile und Sencha Touch zum Einsatz. jQuery Mobile basiert zum Beispiel auf dem weit verbreiteten JavaScript-Framework „jQuery" und stellt zusätzliche Funktionen für die mobile Entwicklung zur Verfügung, wie zum Beispiel Page-Transitions, Theme-Support und eine Widget-Bibliothek. Eine Liste der von jQuery Mobile unterstützten Plattformen findet man unter [3].

Mobile Web-Apps werden von einem Web-Server gehostet und laufen wie jede Web-Anwendung unter der Laufzeit-Umgebung des Servers. Einerseits profitiert der Entwickler von dem Vorteil, keinen restriktiven und langwierigen Approval-Prozess eines App-Store-Anbieters durchlaufen zu müssen. Andererseits hat er aber auch keinen Zugriff auf die Marketing-Mechanismen der App-Stores und muss sich um den Vertrieb seiner App selber kümmern.

Wie schon erwähnt, werden Web-Apps mit der noch jungen HTML5-Technologie entwickelt. Da sich die HTML5-Spezifikation in der Entwicklung befindet, ist sie natürlich noch vielen Änderungen unterworfen. Welche HTML5-Features zur Verfügung stehen und in welchem Umfang sie umgesetzt werden, hängt stark von den Browsern ab, die auf dem Gerät laufen. Die Browser für iOS und Android beispielsweise basieren auf der freien HTML-Rendering-Engine „WebKit", in der bereits viele HTML5-Spezifikationen zum Einsatz kommen.

Die integrierten WebKit-Releases der Browser-Hersteller unterscheiden sich in den Versionen und damit in Quantität und Qualität der umgesetzten HTML5-Features. Deshalb kann es Probleme geben, wenn zum Beispiel eine Browser-Engine ein bestimmtes HTML5-Feature bereits implementiert hat, diese Funktion auf anderen Plattformen aber noch fehlt.

Der Entwickler steht damit vor der Aufgabe, zur Laufzeit festzustellen, ob der aufrufende Browser das gewünschte HTML5-Feature unterstützt. Über den User-Agent-Header, die Browser-Version und die Betriebssystem-Variante lassen sich keine genauen Rückschlüsse auf die unterstützten HTML5-Features eines Client ziehen. Deutlich genauer und komfortabler geht es mit der JavaScript-Library „Modernizr". Die Library prüft beim Laden der Seite, welche HTML5- und CSS3-Features der Browser unterstützt. Neben der Feature-Detection bietet Modernizr noch weitere Features wie Resource-Loading und ein JavaScript-API, das man um eigene Tests erweitern kann. Modernizr ist ein mächtiges und nützliches Tool, das in keinem Werkzeugkasten eines HTML5-Entwicklers fehlen sollte [4].

Cross-Plattform mit hybriden Apps

Auch eine hybride App entwickelt der Programmierer als mobile Web-App für mehrere Plattformen. Mit dem Unterschied, dass er hier zusätzlich aus der Web-App mit speziellen Frameworks eine native Anwendung erzeugt, die man über einen App-Store vertreiben kann. Hybrid-Frameworks wie „PhoneGap" oder „Appcelerator Titanium" packen die HTML5-Web-App in eine verteilbare Anwendung für die jeweilige Ziel-Plattform. Über die JavaScript-APIs der Frameworks ist es zusätzlich möglich, Geräte-Funktionen wie „Kamera", „Kontakte" oder „GPS" einzubinden, auch wenn diese noch nicht per HTML5 unterstützt werden.

Bei der hybriden App ist – wie bei der nativen Entwicklung – für jede Ziel-Plattform das jeweilige Developer Kit auf der Entwickler-Maschine erforderlich, für Android-Geräte muss also das Android SDK und für iOS die Xcode IDE auf dem Entwicklungsrechner installiert sein. Da XCode ein Mac-OS-X-Betriebssystem voraussetzt, benötigt ein Programmierer in dem Fall einen Mac-Rechner mit dem entsprechenden Betriebssystem. Diese Umstände lassen sich mit einem Build auf der Infrastruktur eines externen Anbieters umgehen: Als Ergebnis des externen

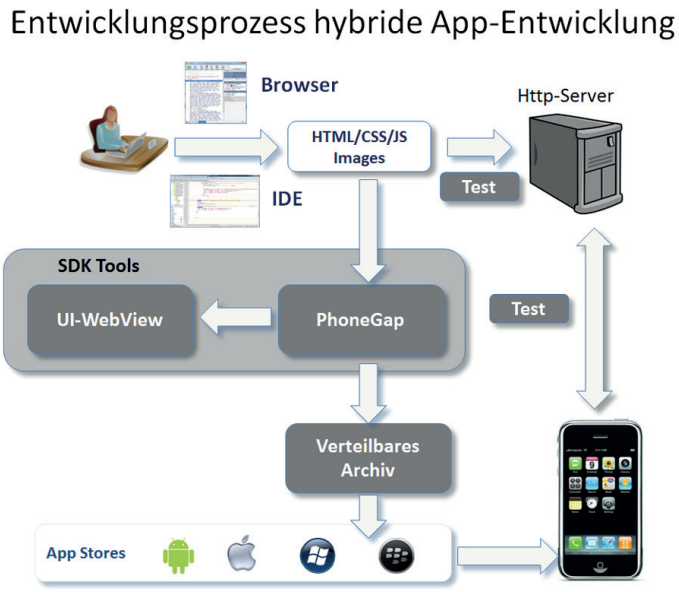

Abbildung 2: Hybrider Entwicklungsprozess

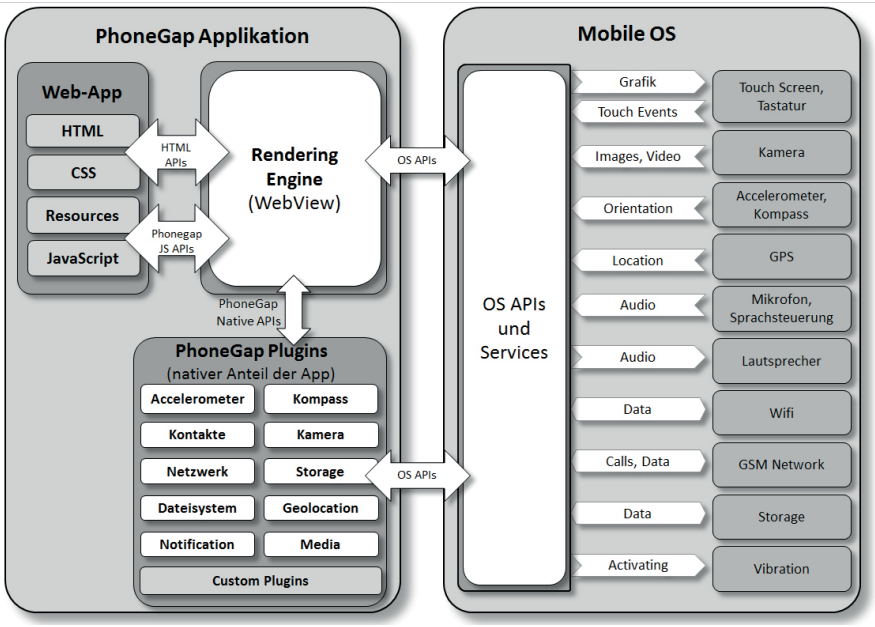

Abbildung 3: PhoneGap-Architektur

Build-Prozesses erhält man ein Artefakt für den App-Store. Diese Leistung wird zum Beispiel von Adobe mit dem kostenpflichtigen „PhoneGap Build" angeboten [5]. Abbildung 2 zeigt den schematischen Entwicklungsprozess einer hybriden App.

Lücken schließen mit PhoneGap

Das derzeit am weitesten verbreitete Hybrid-Framework ist das Open-Source-Framework „PhoneGap". Die Apache Software Foundation hostet PhoneGap mittlerweile unter dem Namen „Cordova". PhoneGap schließt die Lücke zwischen nativen Apps und Web-Apps, indem es aus einer Web-basierten Anwendung eine native Anwendung generiert. Dabei findet jedoch keine Generierung auf Source-Code-Ebene statt, sondern die Web-App wird in einen nativen Container, eine sogenannte „WebView", eingebettet. Die native Ziel-Plattform führt diese WebView aus. PhoneGap packt die Web-App also sozusagen in eine native Hülle, weshalb man von PhoneGap auch als „native Wrapper" spricht. Zusätzlich bietet PhoneGap eine umfangreiche JavaScript-Library mit Schnittstellen zu vielen Geräte-Funktionen, die noch nicht vom HTML5-Standard abgedeckt beziehungsweise von den Browsern noch nicht implementiert wurden, wie „Kamera", „Kontakte" oder „Kompass". PhoneGap unterstützt eine große Anzahl von Geräte-Plattformen. Eine Übersicht der unterstützten Plattformen und Geräte-Funktionen findet man unter [6]. Das Framework PhoneGap besteht im Wesentlichen aus zwei Komponenten (siehe Abbildung 3):

- Ein PhoneGap-JavaScript-API stellt native Funktionalitäten für die JavaScript-Engine des Browsers zur Verfügung
- Ein plattformspezifischer nativer Code, sogenannte „Plug-ins", wird vom PhoneGap-JavaScript-API aufgerufen

Eine PhoneGap-Applikation besteht aus der eigentlichen Web-Anwendung, die ein Entwickler mit Zutaten wie HTML, CSS, Bildern und JavaScript erstellt, aus der Rendering-Engine des Betriebssystems und dem nativen Anteil der Applikation. Der native Anteil der App wird mit PhoneGap-Plug-ins realisiert. Diese wurden in der Sprache der jeweiligen Plattform erstellt und kommunizieren mit den APIs und Services des Betriebssystems.

Die Web-Browser-View bildet das User-Interface einer PhoneGap-Applikation. Sie hat den kompletten Screen des Gerätes zur Verfügung. Der Kern der UI-WebView – die sogenannte „Rendering-Engine" – ist für die Darstellung des HTML-Mark-up zuständig. PhoneGap verwendet die WebView, die das native Betriebssystem als API zur Verfügung stellt. Auf den gängigsten Plattformen wie Android und iOS steht dafür das schon besprochene WebKit „Rendering-Engine" zur Verfügung.

PhoneGap basiert auf einem modularen Plug-in-Konzept. Wer mehr Funktionalitäten benötigt, als Pho-

neGap „out-of-the-box“ mitbringt, kann das Plug-in-Konzept von PhoneGap benutzen, um die Fähigkeiten des Kern-API zu erweitern [7].

Appcelerator Titanium

Das hybride Open-Source-Framework „Titanium“ von Appcelerator verfolgt einen ähnlichen Cross-Plattform-Ansatz wie „PhoneGap“, geht aber noch einen Schritt weiter, indem es die nativen UI-Controls verwendet, die für die jeweilige Ziel-Plattform verfügbar sind. Das sorgt für eine bessere Performance und User Experience gegenüber anderen Hybrid-Frameworks. Derzeit unterstützt das Programmiergerüst als Zielsystem iOS und Android.

Die Entwicklungs-Plattform besteht aus dem Software Developer Kit „Titanium“ und der auf Eclipse basierenden Entwicklungsumgebung „Titanium Studio“. Für die Entwicklung mit Titanium wird JavaScript mit der Erweiterung „CommonJS“ verwendet. Die UI-Elemente sind nativ, für jedes Element gibt es eine Bridge, die zwischen der nativen Umgebung und dem JavaScript-API vermittelt.

Ein umfangreiches JavaScript-API stellt Schnittstellen zu HTML5-Funktionen, für das Model-Binding und Persistence zur Verfügung. Das API bietet Zugriff auf viele Geräte-Funktionen wie „Kamera“, „GPS“, „Sensoren“, „Kalender“ und „Kontakte“. Zudem verfügt Titanium über eine UI-Komponenten-Bibliothek sowie über High-Level-Funktionen für den Zugriff auf Twitter, Foursquare, Facebook und YSL [8].

Oracle ADF Mobile

Auch Oracle hat mittlerweile mit ADF Mobile in der mobilen Anwendungsentwicklung Fuß gefasst. Das Oracle Application Framework (ADF) ist ein JavaEE-Framework für die Entwicklung kompletter Geschäftsanwendungen. Als Entwicklungsumgebung kommt JDeveloper zum Einsatz, der dem Entwickler die Wahl lässt zwischen einem Code-zentrierten oder einem grafischen, deklarativen Ansatz für die Entwicklung von JEE-Komponenten.

ADF Mobile erweitert das JEE-Framework um eine mobile, HTML5-basierte Komponente. Damit lassen sich wie bei anderen Frameworks native App-Pakete aus Web-Apps generieren, nur dass Oracle dabei einen etwas anderen Weg einschlägt: Die dem Framework zugrunde liegende, hybride Entwicklungs-Architektur vereint Web-Techniken wie HTML5 und JavaScript mit serverseitigen Java-Frameworks wie JavaServer Faces (JSF). Im Unterschied zu anderen hybriden Ansätzen integriert ADF eine leichtgewichtige Java Virtual Machine (JVM) in die App.

Die JVM ist die Runtime-Umgebung für die ADF-Business-Komponenten (Managed Beans) und die ADF-Model-Komponenten. Letztere stellen ein Data-Binding zwischen den Views und den Business-Komponenten zur Verfügung, ebenso die Zugriffslogik für SOAP- oder REST-basierte Web-Services.

Der Vorteil von Oracle ADF ist, dass erfahrene Java-Entwickler die Applikationslogik in Java entwickeln können und der erzeugte Bytecode auf iOS und Android-Plattformen lauffähig ist. Mit ADF Mobile ist es damit möglich, vorhandene Business-Komponenten auf mobilen Geräten bereitzustellen. Programmierer sparen mit dem Framework also eine Menge Zeit: Sie müssen die Applikationslogik nur einmal entwickeln und können sie in verschiedenen Kontexten weiterverwenden.

Die Views werden mithilfe spezieller JSF-Komponenten erstellt, sogenannten „ADF Mobile AMX Views“. Zur Laufzeit werden diese von der JavaScript-Engine transparent zu HTML5-Komponenten gerendert.

Auch mit Oracle ADF kann eine gemeinsame Code-Basis für verschiedene Plattformen gesetzt werden, derzeit sind Android und iOS unterstützt – weitere Plattformen wie Windows Phone und Blackberry sind in Arbeit. Intern basiert ADF Mobile auf den schon besprochenen Frameworks „jQuery Mobile“ und „PhoneGap“, sodass ein Entwickler über das PhoneGap-JavaScript-API auf gerätespezifische Funktionen zugreifen kann.

Für den Build und die Installation von Android-Apps braucht man auch hier ein SDK für Android und für iOS-Anwendungen einen Mac-OS-X-Rechner mit der Entwicklungsumgebung „Xcode“ von Apple. ADF Mobile bin-

	Programmiersprache	Vorteile	Nachteile
native App	Android: Java iOS: Objective-C Blackberry: Java, C++ Windows Phone 7/8: C#, C++	optimale Performance alle Gerätefunktionen nutzbar Marketing über App-Stores	Plattform-abhängig Verteilung nur über App-Store Testen und Verteilen benötigt kostenpflichtiges Entwickler-Zertifikat Approval-Prozess in den App-Stores
Web-App	HTML5 (JavaScript + HTML + CSS3)	Web-Entwickler nutzen bekannte Technologien Plattformunabhängigkeit einfache Verteilung	nicht alle Geräte-Funktionen verfügbar Performance ist vom Browser abhängig kein Marketing über App-Stores
hybride App	HTML5 (JavaScript + HTML + CSS3)	Gerätefunktionen fast vollständig verfügbar Plattformunabhängigkeit Marketing über App-Stores	Verteilung nur über App-Store (als App) Testen und Verteilen benötigt kostenpflichtiges Entwicklerzertifikat (als App) Approval-Prozess in den App-Stores (als App)

Tabelle 1

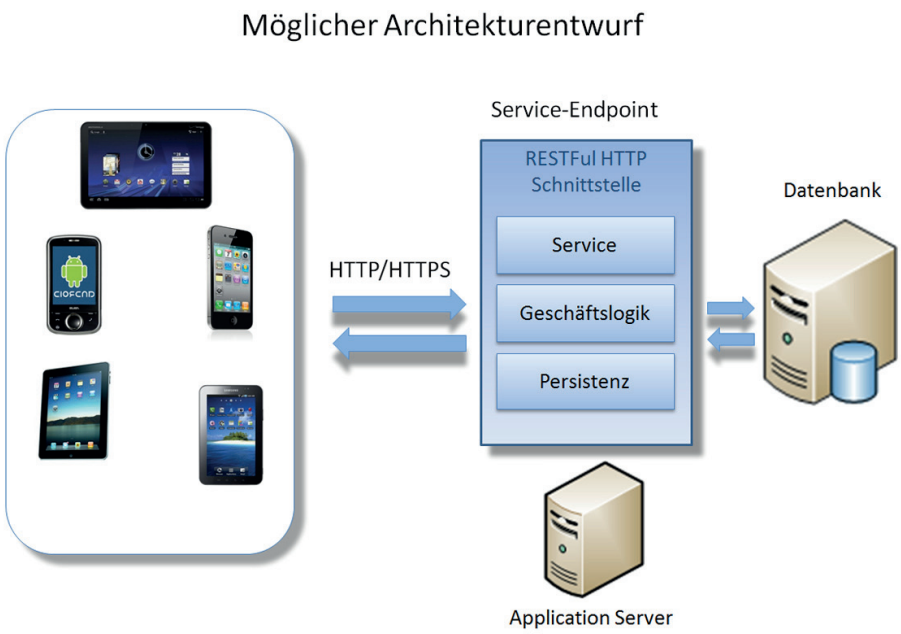

Abbildung 4: Architektur-Entwurf mit RESTFul Back-End

det die SDKs transparent im JDeveloper ein und bietet dem Entwickler ein Deployment direkt aus der IDE auf den iOS-Simulator beziehungsweise auf den Android-Emulator. Aus dem JDeveloper heraus können ebenfalls die entsprechenden IPA-/APK-Pakete für das Deployment auf einem mobilen Gerät erzeugt werden [9].

Testen von Apps

Das intensive und fortlaufende Testen einer App ist ein absolutes Muss während des gesamten Entwicklungsprozesses und damit das tägliche Brot eines mobilen Entwicklers. Für das manuelle Testen unter iOS und Android stellt das jeweilige SDK einen Emulator beziehungsweise Simulator für das Endgerät zur Verfügung. Die Tests im Emulator ersetzen allerdings keinen Test auf einem physikalischen Gerät, da sich die Zielgeräte unter realen Be-

Impressum

Herausgeber:
DOAG Deutsche ORACLE-Anwendergruppe e.V.
Tempelhofer Weg 64, 12347 Berlin
Tel.: 0700 11 36 24 38
www.doag.org

Verlag:
DOAG Dienstleistungen GmbH
Fried Saacke, Geschäftsführer
info@doag-dienstleistungen.de

Chefredakteur (ViSdP):
Wolfgang Taschner, redaktion@doag.org

Redaktion:
Fried Saacke, Carmen Al-Youssef, Mylène Diacquenod, Dr. Dietmar Neugebauer, Franz Hüll, Dr. Frank Schönthaler, Christian Trieb

Titel, Gestaltung und Satz:
Claudia Wagner, Fana-Lamielle Samatin
DOAG Dienstleistungen GmbH

Titelfoto: Fotolia

Anzeigen:
CrossMarketeam Doris Budwill
www.crossmarketeam.de
Mediadaten und Preise finden Sie unter:
www.doag.org/go/mediadaten

Druck:
adame Advertising and Media GmbH Berlin, www.adame.de

dingungen anders verhalten können als im Emulator. Es empfiehlt sich daher dringend, frühzeitig und regelmäßig Tests auf einem angeschlossenen Endgerät auszuführen.

Idealerweise hat der Entwickler Zugriff auf mehrere Geräte, die er zum Testen verwenden kann. Die manuellen Tests können mühsam und langwierig sein, deshalb ist es angebracht, automatisierte Tests kontinuierlich während des gesamten Entwicklungsprozesses auszuführen. Verschiedene Frameworks und Tools unterstützen bei der Erstellung und Ausführung der Tests.

Aufgrund der verschiedenen Laufzeit-Umgebungen und Browser ist es aufwändiger, eine Mobile-Web-App zu testen als eine native App. Wenn auf unterschiedlichen Browsern getestet werden muss, ergibt sich ein komplexes Test-Szenario, das manuelle Tests nur mit großem Aufwand zulässt. Auch hier bietet es sich an, automatisierte Tests durchzuführen. JavaScript-Code lässt sich mit Unit-Tests und entsprechenden Test-Frameworks wie „Jasmine“ [10] oder „Qunit“ testen [11].

Will man das UI einer Web-App automatisiert testen, bieten sich Frameworks an, mit denen der Entwickler den Browser fernsteuern kann. Sie erlauben, die Durchführung von In-Browser-Tests für verschiedene Browser zu automatisieren. Beispiele hierfür sind die Test-Frameworks „Selenium“ [12] und „JsTestDriver“ [13]. Mehr zu testgetriebener Entwicklung mit JavaScript in dem Buch [14].

Architektur-Betrachtungen

Vor der Entwicklung einer mobilen Unternehmensapplikation sollte man sich als Programmierer die Frage stellen, wie der App vorhandene Geschäftsprozesse und Unternehmensdaten zur Verfügung gestellt werden können. In allen Fällen empfiehlt es sich zunächst, vorhandene Back-End-Strukturen und -Schnittstellen ganzheitlich zu betrachten und eine Ist-Analyse der vorhandenen Architektur vorzunehmen. Findet man eine saubere Trennung von Geschäftslogik- und Präsentations-Schicht vor, ist man schon auf einem guten Weg. Denn zwischen diesen beiden Schichten kann man einen Service-Layer in Form einer „BusinessFacade“ einfügen. Die Service-Schicht stellt der Anwendung eine Back-End-seitige Geschäftslogik und Unternehmensdaten zur Verfügung. Dieser integrative Ansatz hat den Vorteil, dass aus einer nativen App, aus einer mobilen Web-App und auch aus einem vorhandenen Web-Portal auf die Service-Schicht zugegriffen werden kann. Abbildung 4 zeigt einen schematischen Architektur-Entwurf. Es hat sich als Best Practice erwiesen, eine mobile App nach dem MVC-Muster aufzubauen und entsprechend in die Gesamtarchitektur zu integrieren. Das Back-End fungiert dann quasi als Model in diesem Design-Ansatz.

Tabelle 1 gibt abschließend einen guten Überblick über die Vor- und Nachteile sowie die primären Programmiersprachen der Plattformen für die unterschiedlichen Entwicklungsansätze, die in diesem Beitrag beschrieben wurden.

Quellen und weitere Informationen

[1] Android Developer Tools: http://developer.android.com/tools/index.html
[2] iOS Dev Center: https://developer.apple.com/devcenter/ios/index.action
[3] jQuery Mobile Supported platforms: http:// http://jquerymobile.com/gbs/
[4] Modernizr: http://modernizr.com
[5] Adobe PhoneGap Build: http://build.phonegap.com
[6] PhoneGap Supported Features: http://phonegap.com/about/feature
[7] PhoneGap: http://phonegap.com
[8] Appcelerator Titanium: http://www.appcelerator.com
[9] Oracle ADF Mobile: http://www.oracle.com/technetwork/developer-tools/adf/overview/adf-mobile-096323.html
[10] Jasmine: http://pivotal.github.com/jasmine/
[11] Qunit: http://qunitjs.com
[12] Selenium: http://seleniumhq.org
[13] js-test-driver: http://code.google.com/p/js-test-driver
[14] Tobias Bosch, Stefan Scheidt, Torsten Winterberg: Mobile Web-Apps mit JavaScript: Leitfaden für die professionelle Entwicklung, entwickler.press, Frankfurt/Main, August 2012

Jörg Bredlau
joerg.bredlau@opitz-consulting.com