Plattformunabhängige App-Entwicklung für mobile Geräte
—
Grenzen und Möglichkeiten

BACHELORARBEIT

im Studiengang
Medieninformatik

des Fachbereichs
Informatik und Medien
der Beuth Hochschule für Technik Berlin

Vorgelegt von
Daniel Morgenstern

im Wintersemester 2013/2014

Betreuende Lehrkraft:
Prof. Dr. Simone Strippgen

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Ziel der Arbeit

1.2 Motivation

1.3 Aufbau der Arbeit

Kapitel 2

Theoretische Grundlagen

2.1 Apps für mobile Geräte

2.1.1 Mobile (native) Apps

Unter mobilen Apps (Kurzform für engl. „Application“) versteht man im Allgemeinen Anwendungssoftware für Tablet-Comuter oder Smartphones. Im Laufe der letzten Jahre haben sich auf dem Markt für Mobilgeräte durch viele konkurierende Gerätehersteller eine Vielzahl von Smartphone- und Tablet-Betriebssystemen herausgebildet. Im Entwicklungsbereich wird in dem Zusammenhang auch von *Plattformen* gesprochen.

Zu den Plattformen mit dem höchsten Marktanteil zählen Googles Betriebssystem Android, iOS von Apple, Microsoft Windows Phone und Blackberry OS des gleichnamigen Smartphone-Herstellers Blackberry.[3] Die App-Entwicklung für diese mobilen Betriebssysteme erfolgt mehr oder weniger ähnlich und soll im Folgenden, um auf die beiden größten Vertreter einzugehen, anhand von Android beziehungsweise iOS näher beschrieben werden.

Grundsätzlich müssen auf der Entwicklungsumgebung die entsprechenden SDKs (Software Development Kit) der Plattform, für die entwickelt wird, installiert sein. Diese enthalten Softwarekomponenten, die zur Entwicklung der App notwendig sind, beispielsweise Klassen, die es einem erlauben, auf native Funktionalitäten des Betriebssystems wie zum Beispiel das Adressbuch, den Benachrichtigungsmechanismus oder auch auf Hardwarekomponenten wie die Kamera, den Bewegungssensor oder das GPS-Modul zuzugreifen sowie die entsprechenden plattformspezifischen Oberflächenkomponenten des jeweiligen GUI-Toolkits zu nutzen.

Als Programmiersprache für die Android-App-Entwicklung wird Java verwendet. Das heißt, als Voraussetzung für die Entwicklung von Android-Apps ist lediglich eine geeignete Entwicklungsumgebung wie *Eclipse*, *NetBeans IDE* oder *IntelliJ IDEA* sowie eine Installation des Java- und des Android-SDK nötig. Seit 2013 bietet Google darüberhinaus die auf IntelliJ IDEA basierende und eigens für die Android-Entwicklung angepasste Entwicklungsumgebung *Android Studio* an,[1] die bereits alle notwendigen Toolkits enthält. Nachdem der Code geschrieben ist, kann er kompiliert und zu einem lauffähigen Programm gebaut werden (siehe Abbildung 2.1). Anschließend kann die App in dem für die Zielplattform vorgesehenen Dateiformat ausgeliefert und auf dem Zielgerät installiert werden.

Auch Apple bietet mit *Xcode* eine firmeneigene IDE zur App-Entwicklung für sein mobiles Betriebssystem iOS an. Anders als Google geht der iPhone-Hersteller hier allerdings etwas restriktiver vor. So läuft die Entwicklungsumgebung Xcode, die man für die native iOS-Entwicklung benötigt, nur unter dem hauseigenen Betriebssystem Mac OSX und das wiederum nur auf den firmeneigenen Mac-Rechnern. So sichert sich Apple auch durch jeden Entwickler einen neuen Kunden.

Ansonsten verläuft der Entwicklungsprozess bei der iOS-Entwicklung im Prinzip ähnlich zur Android-Entwicklung (siehe Abbildung 2.1). Als Programmiersprache wird *ObjectiveC* verwendet, einer um objektorientierte Elemente erweiterte Variante der Programmiersprache *C*.

Möchte ein Auftraggeber einer Software also statt seinen Kunden nur eine App für ein Betriebssystem anzubieten, einen größeren Nutzerkreis erschließen, muss die zu entwicklende App für jede Zielplattform neu programmiert, getestet und gebaut werden, da jede mobile Plattform ihre eigenen Toolkits, Bibliotheken und Programmiersprachen verwendet, was die native App-Entwicklung für potenzielle Auftraggeber zu einem sehr kostenaufwändigen Projekt werden lassen kann. Andererseits bietet die native App-Entwicklung vollständige Unterstützung der betriebssystemeigenen Funktionalitäten wie den Zugriff auf Kamera, Adressbuch, Bewegungssensoren etc. der jeweiligen Plattform, sodass ein Softwareprojekt mit solchen besonders hardware- oder betriebssystemnahnen Anforderungen die Entwicklung einer nativen (plattformspezifischen) App notwendig erscheinen lassen kann.[1]

[1] Mehr dazu in Unterabschnitt 2.2.3

2.1.2 Web-Anwendungen

Eine Web-App (oder dt. *Web-Anwendung*) ist eine Andwendungssoftware, die auf einem Web-Server läuft und auf die der Nutzer mittels eines Browsers zugreifen kann; also eine dynamische Website, wie man sie auch schon vor dem Aufkommen von Smartphones und modernen Tablets kannte.

Die Grundlage des langjährigen Standards für die Enwicklung von Internetseiten bildet HTML (Hyper Text Markup Language), mit der deren Aussehen, Inhalt und Struktur textuell beschrieben werden kann. In Kombination mit CSS (Cascading Style Sheets) für das modulare Styling einer Website sowie Javascript, einer Skriptsprache zur DOM-Manipulation bietet die HTML-Spezifikation in ihrer neusten Version (HTML5) im Grunde alles, was für die Entwicklung einer modernen Benutzerschnittstelle am Computer notwendig ist. Die Fachlogik liegt, neben den Oberflächen-Komponenten in Form von HTML-, CSS- und Javascript-Dokumenten, auf einem Webserver und verarbeitet und reagiert auf Anfragen des Clients (Browser). Als Server-Technologie ist ein breites Spektrum an Programmiersprachen und Umgebungen einsetzbar (einige sind beispielsweise PHP, Java, ASP, CGI u.v.m).

Somit ist die Entwicklung einer Web-App (abgesehen von einigen browserspezifischen Eigenheiten) plattformunabhängig, da jedes moderne (mobile) Betriebssystem über einen Webbrowser verfügt. Zwar müssen Entwickler in bestimmten Details bei der Erstellung des Codes auf die teilweise unterschiedliche Unterstützung (bspw. von HTML-Elementen) durch die verschiedenen Browser achten, aber darüberhinaus wird der Entwicklungsaufwand für eine Web-App nicht von der Anzahl der Zielplattformen bestimmt, da von Client-Seite aus verschiedene Browser durch die Verbreitung und Beachtung von Web-Standards weitgehend einheitliche HTML-Dokumente lesen und interpretieren können und das Backend nicht auf Clients mit unterschiedlichen Plattformen, sondern auf Webservern liegt, deren Plattform bei der Entwicklung entweder schon bekannt oder nicht relevant ist (beispielsweise weil auch die Fachlogik plattformunabhängig mit PHP oder Java realisiert wurde).

Obwohl es, durch damals eher im Business-Bereich verortete Internet-Handys und Palmtops, auch vor den heute üblichen mobilen Touch-Geräten bereits mobile Internetseiten gab, die speziell für die Darstellung auf kleinen Displays ausgerichtet waren, boten mit der massenhaften Verbreitung

von mobilen, internetfähigen Geräten und deren (im Folgenden erläuterten) stark anwendungsorientierten Bedien-Konzept viele herkömmliche Web-Dienste nun auch zusätzlich eine native App für verschiedene mobile Plattformen an. So sind beispielsweise auch E-Mail-Dienste wie GMX[2], WEB.DE[6] oder GMAIL[5] seit dieser Entwicklung auch in Form einer eigenen App für Android und iOS vertreten, sodass der Nutzer, statt, wie von der Desktop-Computer-Nutzung gewohnt, einen anbieterunabhängigen Mail-Client zu konfigurieren, über den er seine E-Mails abruft, unter Umständen gleich die jeweilige App des E-Mail-Anbieters startet. Das heißt, der Nutzer folgt einem geänderten Bedienungsmuster seines Mobilgeräts gegenüber der herkömmlichen Computer-Nutzung: um zu einem bestimmten Ergebnis zu gelangen (bspw. *Nachrichten lesen*) also die Frage zu beantworten, *wie* er dahin gelangt (Einen Browser öffnen, zur gewünschten Seite navigieren: www.tagesschau.de), ist es für Anwender heutiger Mobilsysteme naheliegend, gleich die passende App zu starten (hier bspw. die Tagesschau-App).

Dafür gibt es verschiedene mögliche Gründe. Zum Einen muss im Gegensatz zu einer Website bei der mobilen App nicht die komplette Oberfläche (HTML-, CSS- und JavaScript-Dokumente sowie Grafiken) übertragen werden, sondern lediglich die Nutzdaten (also beispielsweise, um beim obigen Beispiel zu bleiben, die Nachrichten in Textform), was dem Nutzer ein höheres Maß an Performanz einbringt. Zum Anderen können trotz Vollbildmodus in bestimmten Fällen GUI-Elemente des Webbrowsers bei der Benutzung einer Web-Anwendung störend sein, so ist beispielsweise die Adresszeile am Rand nicht unbedingt erwünscht, wenn der Nutzer statt im Internet zu surfen dort eigentlich eine bestimmte Anwendung nutzen möchte. Ein anderes Beispiel für ein eventuell unerwünschtes Verhalten der Benutzerschnittstelle ist das der *Menü*-Taste bei Android-Geräten, die im Falle der Nutzung einer Web-Anwendung über den Browser nicht den Kontext der eigentlich benutzten Anwendung (hier also der Website) anzeigt, sondern lediglich den des Browsers.

Durch die Ausrichtung auf die Online-Nutzung einer Web-Anwendung ist diese im Gegensatz zu vielen nativen Apps, sofern sie nicht zwingend mit einem Server kommunizieren müssen, außerdem nicht ohne weiteres offline nutzbar.

Allgemein kann man sagen, dass der Zugriff auf native Funktionalitäten des Geräts respektive des Betriebssystems nicht oder nur gering unterstützt

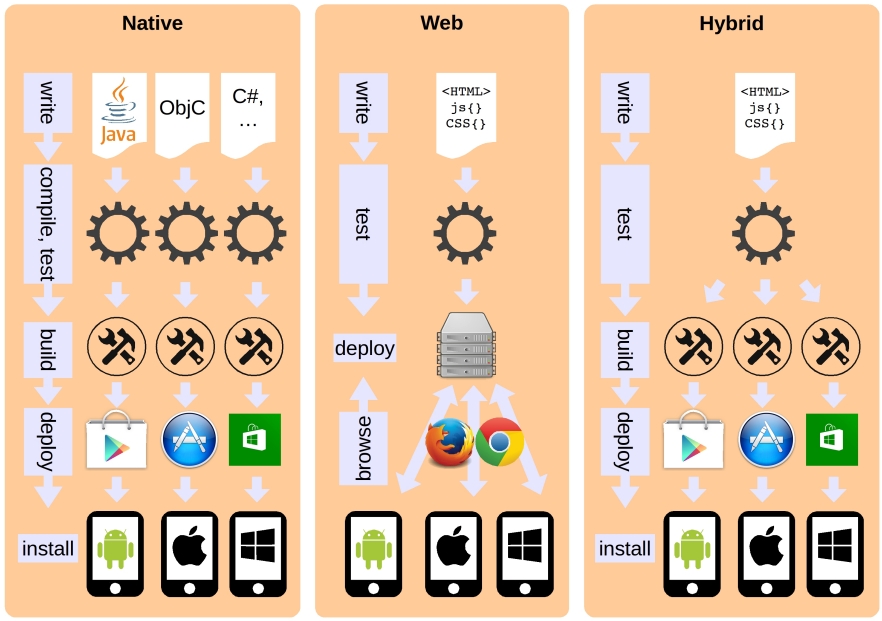

Abbildung 2.1: Entwicklungsstufen der verschiedenen Arten von Apps. Während bei der nativen App der gesamte Entwicklungszyklus einmal pro Plattform durchlaufen werden muss, verringert sich der Aufwand für die Web-App erheblich. Bei der Hybrid-App muss die Anwendung zwar einmal für jede Plattform gebaut und ausgeliefert werden um die Schnittstellen für die nativen Plattformen zu implementieren, aber der hauptsächliche Entwicklungsaufwand des Programmierens und testens fällt aufgrund des generischen Charakters nur einmal an.

wird, sodass der geringere Enwicklungsaufwand einer solchen Web-App (siehe Abbildung 2.1) unter Umständen zu Lasten des Funktionsumfangs und der Usibility der Anwendung geht.

2.1.3 Hybride Apps

2.2 Plattformunabhängige App-Entwicklung

2.2.1 Möglichkeiten des Erreichens von Plattformunabhängigkeit

2.2.2 Lösungen für die hybride App-Entwicklung

2.2.3 Entwicklung von hybriden Apps

Phonegap

Die entscheidende Lücke auf dem Weg zur Hybrid-App zwischen Webanwendung und der nativen App schließt das Framework *PhoneGap* von Adobe. Bei der Webanwendung taten sich in der Herleitung zwei grundlegende Problemfelder gegenüber der nativen App-Entwicklung auf. Zum Einen besteht eine Webanwendung aus mehreren Dokumenten, die auf einem Server liegen und nur über den Browser des Betriebssystems abgerufen werden, die Benutzung fühlt sich also für den Nutzer durch die ausbleibende Installation, die benötigte Internetverbindung und die optische Präsenz der Browser-Oberfläche nicht wie eine mobile App an. Zum Anderen können die Anforderungen an die Anwendung einen Zugriff auf native Features des Geräts oder seines Betriebssystems erfordern, der mittels herkömmlicher Webtechnologien nicht oder nur sehr eingeschränkt möglich sind.

Für letzteres bietet PhoneGap eine Javascript-Bibliothek, die auf das Cross-Platform-Framework *Cordova* von Apache aufbaut und den Zugriff auf native Features des jeweiligen Betriebssystems ermöglicht. Bei der konkreten Verwendung von nativen Features muss jedoch teilweise wieder auf die Unterstützung durch die jeweiligen Plattformen geachtet werden.

Darüber hinaus übernimmt das Online-Portal *PhoneGap Build* den Bauprozess der App, also das Überführen in ein installierfähiges App-Format. Während der Entwickler ohne diesen Build-Service die verschiedenen Toolkits aller Zielplattformen lokal verwalten müsste, um Cordova zu verwenden,[4] reicht es hier aus, die Web-Anwendung (beispielsweise per öffentlichem Git-Repository) auf das Portal hoch zu laden, den Build-Prozess im Browser abzuwarten und die fertigen Apps auf das gewünschten Zielgerät herunterzuladen und zu installieren.

Für Auftraggeber oder Entwickler, die ihren Code als sehr sensibel und

vertraulich ansehen, könnte diese Variante allerdings ein Problem darstellen sein, da der Quellcode dann stets auch auf den Servern von Adobe liegt.

JQueryMobile

Die Javascript-Bibliothek JQueryMobile baut auf JQuery auf und bietet ein Oberflächen-Toolkit für mobile Webseiten. Sie besteht zum Einen aus einer Javascript-Datei, die in die HTML-Seite eingebunden wird und zum Anderen aus einem CSS, das für das an mobile Touch-Geräte angepasste aussehen verantwortlich ist. So können einerseits UI-Elementen explizit Style-Klassen aus dem JQueryMobile-CSS zugewiesen werden, andererseits sorgt das JQuery-Javascript ohnehin bei allen verwendeten HTML-Elementen dafür, dass die Style-Klassen dynamisch vergeben werden und die UI-Komponenten somit ihr App-typisches Aussehen erhalten. Lediglich mit dem zusätzlichen Attribut "data-role"werden den UI-Elementen Eigenschaftentypen zugewiesen, über die die JQueryMobile-Bibliothek erkennt, wie dieses dargestellt werden soll.

Dieser Mechanismus erleichtert es dem Frontend-Entwickler erheblich, eine auf Touchscreens ausgerichtete Oberfläche zu entwerfen, da er im Grunde neben einigen zusätzlichen Attributen sämtliche herkömmlichen HTML-Elemente verwenden kann.

KnockoutJS

Auch bei *Knockout* handelt es sich um eine Javascript-Bibliothek, die per `<script>`-Tag der HTML-Seite hinzugefügt wird. Knockout übernimmt mit einem MVVM (Model-View-ViewModel) das *Data-Binding*, also die Verknüpfung zwischen Daten-Feldern im Programm und UI-Elementen. So lässt sich die UI (View) sehr klar von der Programmlogik (Model) trennen, was der Lesbarkeit, Erweiterbarkeit und Wartbarkeit der Software zugute kommt.

Statt also Javascript mit HTML-String-Schnipseln zu vermischen, indem das DOM direkt manipuliert und UI-Elemente zur Laufzeit programmatisch erweitert werden, definiert der Entwickler in einem separaten Javascript ein View-Model und bindet mit dem HTML-Attribut „data-bind“ eine bestimmte View-Eigenschaft an ein Datenfeld aus dem View-Model.

Konkreter, Beispiel, Grafik.

Quellenverzeichnis

Online-Quellen

[1] Xavier Ducrohet, Tor Norbye und Katherine Chou. *Android Studio: An IDE built for Android*. 15. Mai 2013. URL: http://android-developers.blogspot.in/2013/05/android-studio-ide-built-for-android.html (besucht am 26. 03. 2014) (siehe S. 3).

[2] *Kostenlose GMX Mail App - Ihr Postfach immer & überall dabei*. 1&1 Mail & Media GmbH. 31. März 2014. URL: http://www.gmx.net/produkte/mobile/mail-app (besucht am 29. 03. 2014) (siehe S. 5).

[3] Ramon Llamas, Ryan Reith und Michael Shirer. *Android Pushes Past 80 % Market Share While Windows Phone Shipments Leap 156.0 % Year Over Year in the Third Quarter, According to IDC*. IDC Corporate USA. 12. Nov. 2013. URL: http://www.idc.com/getdoc.jsp?containerId=prUS24442013 (besucht am 26. 03. 2014) (siehe S. 2).

[4] *PhoneGap API Documentation*. Adobe Systems Inc. 25. Feb. 2014. URL: http://docs.phonegap.com/en/3.4.0/guide_cli_index.md.html#The Command-Line Interface (besucht am 20. 03. 2014) (siehe S. 7).

[5] Rene Ritchie. *Google releases official Gmail app for iPhone, iPad*. 2. Nov. 2011. URL: http://www.imore.com/google-releases-official-gmail-app-iphone-ipad (besucht am 31. 03. 2014) (siehe S. 5).

[6] *WEB.DE App - Postfach Apps für Smartphone, iPhone und iPad*. 1&1 Mail & Media GmbH. 31. März 2014. URL: https://produkte.web.de/freemail_mobile_startseite (besucht am 31. 03. 2014) (siehe S. 5).