

Plattformunabhängige App-Entwicklung für mobile Geräte

—

Grenzen und Möglichkeiten

- Bachelorarbeit -

Daniel Morgenstern

22. April 2014

Inhaltsverzeichnis

1	Einleitung	3
1.1	Ziel der Arbeit	3
1.2	Motivation	3
1.3	Aufbau der Arbeit	3
2	Theoretische Grundlagen	4
2.1	Apps für mobile Geräte	4
2.1.1	Mobile (native) App	4
2.1.2	Web-Anwendung	5
2.2	Entwicklung von Hybriden Apps	7
2.2.1	Grundlegende Technologien	7
2.2.2	Phonegap	7
2.2.3	jQueryMobile	8
2.2.4	KnockoutJS	8
	Literaturverzeichnis	9

1 Einleitung

1.1 Ziel der Arbeit

1.2 Motivation

1.3 Aufbau der Arbeit

2 Theoretische Grundlagen

2.1 Apps für mobile Geräte

2.1.1 Mobile (native) App

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 konkurrierende 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 muss der Entwickler die entsprechenden SDKs (Software Development Kit) der Plattform, für die entwickelt wird, in seiner Entwicklungsumgebung installiert haben. 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 (GUI-Toolkit) zu nutzen.

Als Programmiersprache wird für die Android-App-Entwicklung Java verwendet. Das heißt, grundsätzlich kann jeder, der das Java- und das Android-SDK auf seinem Rechner installiert hat, mit einer beliebigen geeigneten Entwicklungsumgebung wie *Eclipse*, *NetBeans IDE* oder *IntelliJ IDEA* Apps für Android entwickeln. Seit 2013 bietet Google darüberhinaus die auf IntelliJ IDE 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 vorgesehene 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

wie sieht es mit dem deployment, also der auslieferung in appstores etc aus?

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. Als Programmiersprache wird *ObjectiveC* verwendet, einer um objektorientierte Elemente erweiterte Variante von *C* (siehe Abbildung 2.1).

darf man
so was
hier an-
merken,
oder lieber
weg?

2.1.2 Web-Anwendung

Eine Web-App (oder dt. *Web-Anwendung*) ist eine Anwendungssoftware, 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.

vielleicht
auch
eher nach
Unterab-
schnitt 2.1.1

Zwar gab es, durch damals eher im Business-Bereich verortete Internet-Handys und Palmtops, auch vor den heute üblichen mobilen Touch-Geräten bereits mobile Internetseiten, die speziell für Darstellung auf kleinen Displays ausgerichtet waren, doch mit der massenhaften Verbreitung von mobilen internetfähigen Geräten und deren strikten anwendungsorientierten Bedien-Konzept boten viele herkömmliche Web-Dienste nun auch zusätzlich eine native App für verschiedene mobile Plattformen an. 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).

nachweis

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).

erklären!

2.1.3

Zumeist nutzen die verschiedenen Plattformen nicht nur unterschiedliche Dateiformate für ihre Anwendungen sondern auch unterschiedliche Programmiersprachen und Toolkits für die Programmierung. So verwendet beispielsweise Android die Programmiersprache Java in Kombination mit einem eigenen Android SDK, während Apps für Apples iOS mit ObjectiveC in der Firmeneigenen Entwicklungsumgebung Xcode (die auch nur auf dem firmeneigenen Betriebssystem Mac OSX läuft) geschrieben werden. Somit ist die Plattformunabhängigkeit mit den herkömmlichen Entwicklungsmethoden kaum zu erreichen, da die verschiedenen Produkte und Plattformen zueinander nicht kompatibel sind.

gehört der
nativ-web-
vergleich
jetzt eher
auch
hierher
oder in
Unterab-
schnitt 2.1.1
?

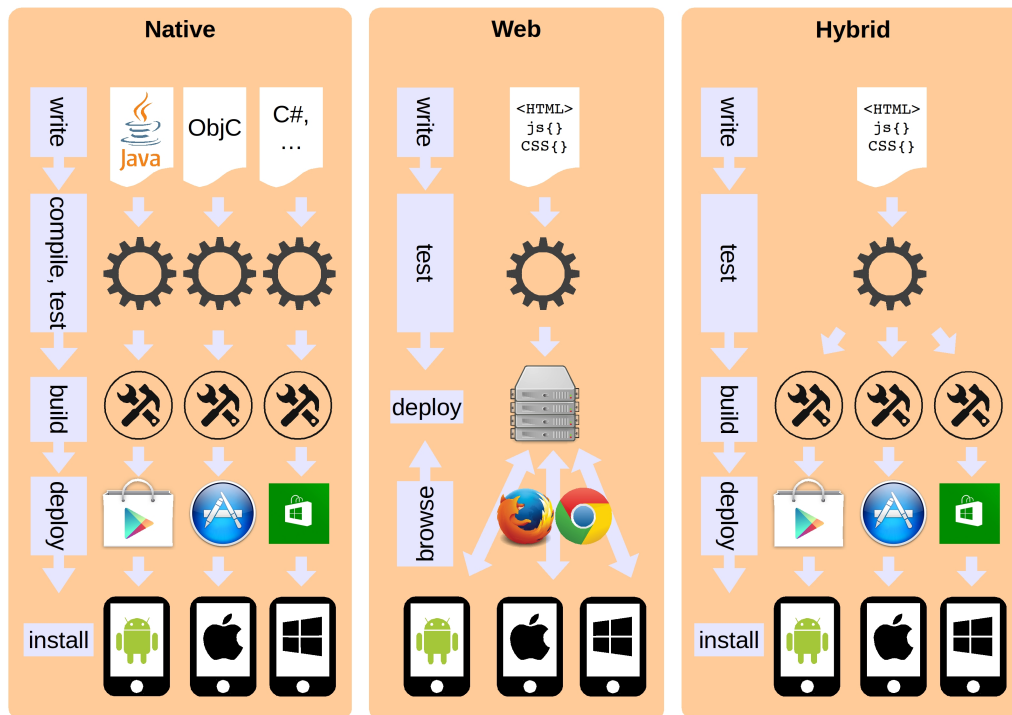


Abbildung 2.1: Entwicklungsstufen der verschiedenen Arten von Apps. Während bei der nativen App der gesamte Entwicklungszyklus einmal pro Plattform durchlaufen werden muss, verringert sich der Aufwand für die Web-App erheblich. Bei der Hybride-App muss die Anwendung zwar einmal für jede Plattform gebaut 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.

Im Planungsprozess einer Softwarelösung ist in bestimmten Fällen eine denkbare Alternative zur nativen App eine Webanwendung, die mithilfe des Browsers abrufbar ist, also konkret eine dynamische Website, wie man sie auch von der herkömmlichen Internet-Nutzung eines Desktop-Systems kennt.

Diese Variante hat zwar den Vorteil, dass die Anwendung plattformunabhängig ist, da jedes moderne (mobile) Betriebssystem einen Browser besitzt, der Entwicklungsaufwand also unabhängig von der Anzahl der Zielplattformen der gleiche bleibt, allerdings stößt der Entwickler schnell an seine Grenzen, wenn Hardware- oder betriebssystemnahe Anforderungen gestellt sind. So ist es beispielsweise mit Webtechnologien wie HTML oder Javascript nicht ohne weiteres möglich, auf die Kamera oder das GPS-Modul eines Gerätes oder das Adressbuch eines Betriebssystems zuzugreifen.

nachweis!

Ebenso kommt die Webanwendung im Normalfall nicht für Offline-Anwendungen in Frage, da die meisten mobilen Betriebssysteme nicht vorsehen, dass der Nutzer aus dem Dateisystem eine HTML-Seite öffnet. Dahingehend liegt ein Ansatz zur plattformunabhängigen App-Entwicklung auf der Hand: Die Hybrid-App, also die Verwendung von Webtechnologien mithilfe einer Schnittstelle zwischen der Webanwendung und der nativen Ebene des Betriebssystems.

nachweis!

2.2 Entwicklung von Hybriden Apps

2.2.1 Grundlegende Technologien

Konkret heißt das, die Software wird als Webanwendung mit den zugehörigen Technologien (HTML, Javascript, CSS) entwickelt und für die jeweiligen benötigten Plattformen in eine native App eingebettet, welche hier allerdings hauptsächlich aus einer Web-View, also einer abgespeckten Variante eines Web-Browsers zur Anzeige der Webanwendung, besteht. Da dieser Ansatz der am häufigsten von Cross-Platform-Frameworks genutzt ist, stellt dieser auch den Schwerpunkt der in dieser Arbeit explorierten Technologien dar und soll hier näher erläutert werden.

nachweis!

2.2.2 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 jewei-

ligen 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, [2] 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.

2.2.3 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.

2.2.4 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.

wie genau, weiß ich noch nicht, noch rausfinden!

oder zumindest muss deklariert werden, was verwendet werden soll. -> checken!

gehört vllt. eher woanders hin.

Konkreter, Beispiel?

gibt es bei javascript überhaupt eine laufzeit?

Konkreter, Beispiel, Grafik?

Literaturverzeichnis

- [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).
- [2] Adobe Systems Inc. Phonegap api documentation. 25. Februar 2014. URL: http://docs.phonegap.com/en/3.4.0/guide_cli_index.md.html#TheCommand-LineInterface (besucht am 20.03.2014).
- [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. November 2013. URL: <http://www.idc.com/getdoc.jsp?containerId=prUS24442013> (besucht am 26.03.2014).