

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 Sommersemester 2014
Betreuende Lehrkraft:
Prof. Dr. Simone Strippgen

Inhaltsverzeichnis

Abbildungsverzeichnis	iii
1 Einleitung	1
1.1 Motivation und Aufgabenstellung	1
1.2 Ziel und Aufbau der Arbeit	1
I Theoretische Grundlagen	2
2 Apps für mobile Geräte	3
2.1 Mobile (native) Apps	3
2.2 Web-Anwendungen	5
2.3 Hybride Apps	8
3 Plattformunabhängige App-Entwicklung	10
3.1 Überblick: Lösungen und Ansätze	10
3.2 Entwicklung von hybriden Apps	13
3.2.1 Die <i>JavaScript</i> -Bibliothek jQuery	14
3.2.2 Data-Binding mit Knockout	15
3.2.3 JQuery Mobile: Mobile Web-Oberflächen	17
3.2.4 Phonegap / Cordova	20
3.2.4.1 Grundlegendes	20
3.2.4.2 Funktionsweise	23
3.2.4.3 Schnittstelle zur mobilen Plattform	30
3.2.4.4 PhoneGap Build	33
II Praktische Umsetzung	38
4 Konzeption	39
4.1 Vorüberlegungen	39
4.2 Beispiel-Anwendung	39

4.3 Ausgewählte Technologie und Architektur für die Implementierung	41
5 Implementierung der Geräte-Schnittstelle	43
5.1 Zugriff auf die Kontaktverwaltung des Geräts	43
5.1.1 Funktionsweise	43
5.1.2 Explorationsergebnisse	47
6 Fazit	53
Resümee / Ausblick	55
Resümee	55
Ausblick	56
Anhang	57
A Anwendungsfälle der Beispiel-Anwendung	58
A.1 Diagramme	58
A.2 Anwendungsfallbeschreibung	60
B Quellcode	68
C Glossar	83
Begriffe und Abkürzungen	83
Technologien	86
Quellenverzeichnis	90
Bildquellen	90
Software-Quellen und Dokumentationen	90
Online-Quellen	92

Abbildungsverzeichnis

2.1	Schaubild Hybrid Apps	9
3.1	MVVM-Pattern	16
3.2	Knockout-Beispiel im Browser	18
3.3	jQuery Mobile-Beispiel	20
3.4	Beispiel ohne <code>class</code> -Attribut	20
3.5	Beispiel ohne jQuery Mobile	21
3.6	Cordova Plattform- und Feature-Unterstützung	23
3.7	Cordova-Dateistruktur	24
3.8	Cordova-Beispiel-App im Browser	28
3.9	Cordova-Beispiel-App im <i>Android</i> -Emulator	28
3.10	Erstellung einer neuen App auf PhoneGap Build	34
3.11	PhoneGap Build-Pakete	35
3.12	PhoneGap Build-Workflow	36
3.13	PhoneGap Build-Oberfläche	37
3.14	Tablet-Screenshot mit installierter PhoneGap Build-App	37
4.1	Beispiel-Anwendung im iOS-Simulator	40
5.1	Die installierte App in der App-Übersicht.	48
5.2	Natives Adressbuch mit Beispiel-Kontakten	49
5.3	Kontakt-Auswahl in der Anwendung	50
5.4	Übergabe der Kontakte an die Anwendung	51
A.1	Akteure für die Spezifikation der Beispiel-Anwendung	58
A.2	Anwendungsfalldiagramm der Beispielanwendung	59

Kapitel 1

Einleitung

1.1 Motivation und Aufgabenstellung

1.2 Ziel und Aufbau der Arbeit

Teil I

Theoretische Grundlagen

Kapitel 2

Apps für mobile Geräte

2.1 Mobile (native) Apps

Unter mobilen *Apps* (*Kurzform für engl. „Application“*) versteht man im Allgemeinen Anwendungssoftware für Tablet-Computer 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 10* des gleichnamigen Smartphone-Herstellers *Blackberry* [29]. 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* 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 nö-

tig. Seit 2013 bietet Google darüber hinaus die auf IntelliJ IDEA basierende und eigens für die Android-Entwicklung angepasste Entwicklungsumgebung *Android Studio* an [24], die bereits alle notwendigen Toolkits enthält. Nachdem der Code geschrieben ist, kann er kompiliert und zu einem lauffähigen Programm *gebaut* (engl. „build“) werden (siehe Abbildung 2.1). Anschließend kann die App in dem für die Zielplattform vorgesehenen Dateiformat ausgeliefert und auf dem Zielgerät installiert werden.

Auch der Software- und Computer-Hersteller Apple bietet mit *Xcode* eine firmeneigene Entwicklungsumgebung 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 OS X* und das wiederum nur auf den firmeneigenen Mac-Rechnern [3]. Darüber hinaus ist für iOS-Entwickler die Teilnahme am *iOS Developer Program* erforderlich, um Apps für Apple-Geräte auszuliefern und auf Geräten installieren zu können, wofür der Konzern einen jährlichen Mitgliedsbeitrag von 99 \$ im Jahr verlangt [25]. So sichert sich Apple, nicht nur durch die kostenpflichtige Mitgliedschaft im iOS Developer Program, sondern allein schon durch deren exklusive Platformunterstützung ihres eigenen Mobilbetriebssystems, auch mit jedem Entwickler einen neuen Kunden.¹

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 entwickelnde 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

¹ Diese Restriktion fällt allerdings auch in der unten beschriebenen plattformunabhängigen App-Entwicklung nicht unbedingt weg (siehe Abschnitt 3.2).

ein Softwareprojekt mit solchen besonders hardware- oder betriebssystemnahen Anforderungen die Entwicklung einer nativen (plattformspezifischen) App notwendig erscheinen lassen kann.²

2.2 Web-Anwendungen

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.

Die Grundlage für die Entwicklung von Internetseiten bildet der langjährige Standard *HTML*, mit dem deren Aussehen, Inhalt und Struktur textuell beschrieben werden kann. In Kombination mit *CSS* für die modulare Gestaltung 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 grafischen Benutzerschnittstelle 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.³ Als Server-Technologie ist ein breites Spektrum an Programmiersprachen und Umgebungen einsetzbar.⁴

Somit bietet die Entwicklung einer Web-App (abgesehen von einigen Browser-spezifischen Eigenheiten) bereits eine gewisse Plattformunabhängigkeit, da jedes moderne Betriebssystem über einen Webbrower verfügt. Zwar müssen Entwickler in bestimmten Details bei der Erstellung des Codes auf die teilweise unterschiedliche Unterstützung (beispielsweise von *HTML*-Elementen) durch die verschiedenen Browser achten, aber darüber hinaus wird der Entwicklungsaufwand für eine Web-App nicht von der Anzahl der Zielpлатzformen 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 Webserversn liegt, deren Plattform bei der Entwicklung entweder schon bekannt oder nicht relevant ist.⁵

² Mehr dazu in Abschnitt 3.2

³ Die Rolle des Clients übernimmt hier also der Browser.

⁴ Einige sind beispielsweise *PHP*, *Java*, *ASP* u. v. a. m.

⁵ 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-Konzepten viele herkömmliche Internet-Dienste nun auch zusätzlich eine native App für verschiedene mobile Plattformen an. So sind beispielsweise auch E-Mail-Dienste wie *GMX*, *Web.de* oder *Gmail* seit der Verbreitung von Smartphones und Tablets 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 [28, 33, 38]. Das heißt, der Nutzer folgt einem geänderten Bedienungsmuster seines Mobilgeräts gegenüber der herkömmlichen Computer-Nutzung: um zu einem bestimmten Ergebnis zu gelangen (beispielsweise *Nachrichten lesen*) also die Frage zu beantworten, *wie* er dahin gelangt (Einen Browser öffnen und zur gewünschten Seite navigieren: www.tagesschau.de), ist es für Anwender heutiger Mobilsysteme naheliegend, gleich die passende App zu starten (Hier z. B. die Tagesschau-App).

Dafür gibt es verschiedene mögliche Gründe. Zum Einen muss im Gegensatz zu einer Website bei der mobilen App nicht die komplette Oberfläche⁶ übertragen werden, sondern lediglich die Nutzdaten,⁷ 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 anzeigt,⁸ sondern lediglich den des Browsers.

In bestimmten Fällen kann eine nützliche Funktion einer App die Offline-Nutzung sein, wenn beispielsweise durch die abgedeckten Anwendungsfälle keine Verbindung oder Synchronisation mit einem Server nötig ist. Beispiele hierfür könnten, um nur einige zu nennen, ein Taschenrechner, kleine Spiele,

⁶ HTML-, CSS- und JavaScript-Dokumente sowie Grafiken

⁷ Also beispielsweise, um beim obigen Beispiel zu bleiben, die Nachrichten in Textform.

⁸ hier also der Website

oder eine Bildverarbeitungs-App sein. Für diese Offline-Nutzung einer App zeichnet die Web-Anwendung ein geteiltes Bild: Zwar wurden in den letzten Jahren mehrere Methoden entwickelt, eine Web-Anwendung auch offline nutzen zu können, doch durch ihre Ausrichtung auf die Nutzung via Internet stellt die Implementierung dieser Funktionalität für Entwickler einen Zusatzaufwand dar.

Einige Möglichkeiten, eine Web-Anwendung ohne Internetverbindung nutzbar zu machen, sind beispielsweise die aus der HTML5-Spezifikation hervorgehenden Technologien *WebStorage*, ein Mechanismus zum lokalen Speichern von größeren Datenmengen in Form von Schlüssel-Wert-Paaren [15] sowie *WebSQL* bzw. *IndexedDB*, beides auf Web-Anwendungen optimierte Datenbanken-Spezifikationen, die vom *W3C* herausgegeben werden [14, 17]. Allerdings bestehen auch bei diesen Mechanismen teilweise Einschränkungen durch die Browervielfalt beziehungsweise deren Versionen. So wird IndexedDB beispielsweise nicht von *Safari* oder iOS unterstützt, *Google Chrome* muss für die Nutzung mindestens in Version 23 oder höher vorliegen, *Mozilla Firefox* in 10 oder höher. Auch bei WebSQL zeichnet sich ein ähnlich diffuses Bild ab: Während Chrome die Technologie ab der Version 4 und iOS ab Version 3.2 unterstützt, ist für Nutzer der Browser Firefox und *Microsoft Internet Explorer* die Technik gar nicht verfügbar. Lediglich WebStorage wird weitgehend von allen gängigen Browern unterstützt [31]. Weiterhin wird die Offline-Funktionalität gegenüber der nativen App dadurch eingeschränkt, dass der Nutzer diese ohne weiteres Zutun des Entwicklers nur dann nutzen kann, wenn die entsprechende Internet-Seite im Offline-Zustand des Geräts bereits im Browser geöffnet ist, da diese nicht lokal auf dem Gerät, sondern auf einem Webserver gespeichert ist. Für vollständigen Offline-Zugriff müsste der Entwickler die komplette Website so paketieren, dass der Nutzer sie – wie eine native App – von seinem Gerät aus starten und nutzen kann.⁹

Allgemein kann man sagen, dass der Zugriff auf native Funktionalitäten des Geräts respektive des Betriebssystems nicht oder nur gering unterstützt wird, sodass der geringere Entwicklungsaufwand einer solchen Web-App (siehe Abbildung 2.1) unter Umständen zu Lasten des Funktionsumfangs und der Usability der Anwendung geht.

⁹ siehe Abschnitt 2.3 und 3.2.

2.3 Hybride Apps

Die *Hybrid-App* verbindet Eigenschaften der Nutzung einer nativen App mit den Vorteilen der Web-Entwicklung mithilfe von Web-Technologien und entsprechenden *Frameworks* und löst damit beispielsweise das Problem der mangelnden Offline-Fähigkeit einer Web-App sowie deren geringe Unterstützung von plattformspezifischen oder hardware-nahen Funktionalitäten. Da jedes moderne mobile Betriebssystem für Entwickler auch die Möglichkeit bietet, eine *WebView* in die zu entwickelnde App einzubinden, also eine GUI-Komponente, in die HTML-Inhalte hinein geladen werden können, liegt der Ansatz für hybride Apps auf der Hand: Auf Entwicklungsebene wird die Anwendung als Web-App entwickelt, gleichzeitig mithilfe von entsprechenden *APIs (Programmierschnittstellen)* und Frameworks zur Anbindung an die native Ebene der Zielplattform in eine App für die jeweiligen Plattformen integriert, sodass auf Benutzerseite die Nutzung einer Web-Anwendung, die in puncto Funktionsumfang, Usability und Look-And-Feel einer nativen mobilen App sehr nahe kommt, möglich wird.

Dieses Vorgehen bietet unter anderem für Web-Entwickler den Vorteil, ihre bisherigen Programmierkenntnisse im Web-Bereich im Wesentlichen auch für die Entwicklung von hybriden Apps nutzen zu können. So wird in der Regel der grundlegende Teil des Codes für das *Frontend*, wie bei der Web-Entwicklung, mit HTML in Kombination mit CSS und JavaScript geschrieben und getestet. Da allerdings auf dem mobilen Gerät für das Backend, also die Verarbeitungsinstanzen, nicht, wie bei einer herkömmlichen Web-Anwendung, ein Server mit einer entsprechenden Server-Technologie wie PHP oder ASP läuft, wird auch dieser Teil der App bei der hybriden Entwicklung meist mit JavaScript bewerkstelligt. Anschließend muss die Anwendung für die verschiedenen Zielplattformen gebaut werden, um in das jeweilige Container-Format für Apps der verschiedenen Plattformen eingebunden werden zu können und den Zugriff auf die plattformspezifischen Toolkits durch die Cross-Platform-APIs zu ermöglichen (Abbildung 2.1). Hierfür kann es erforderlich sein, dass auf der Entwicklungsplattform die jeweiligen SDKs installiert sind, was gegenüber der Web-Entwicklung einen administrativen Mehraufwand darstellt. Eine andere Variante ist die Auslagerung des Bauprozesses auf einen externen Build-Server, beispielsweise mithilfe eines externen Web-Service eines Drittanbieters, was den Vorteil hat, die SDKs für die Zielplattformen nicht auf jedem Entwicklungsrechner verwalten zu müssen. Allerdings hat der Entwickler durch die Herausgabe des Codes an einen solchen Dienstleister nicht mehr

die vollständige Kontrolle über den Code, sodass die Variante der Auslagerung des Build-Prozesses gerade für Closed-Source-Projekte unter Umständen nicht in Frage kommt. Des Weiteren kann der Betreiber des Build-Services unter Umständen Restriktionen bezüglich der Plattformunterstützung erteilen, wodurch eventuell eine geringere Anzahl von Zielplattformen unterstützt wird, als von Entwicklerseite gewünscht oder erforderlich.¹⁰

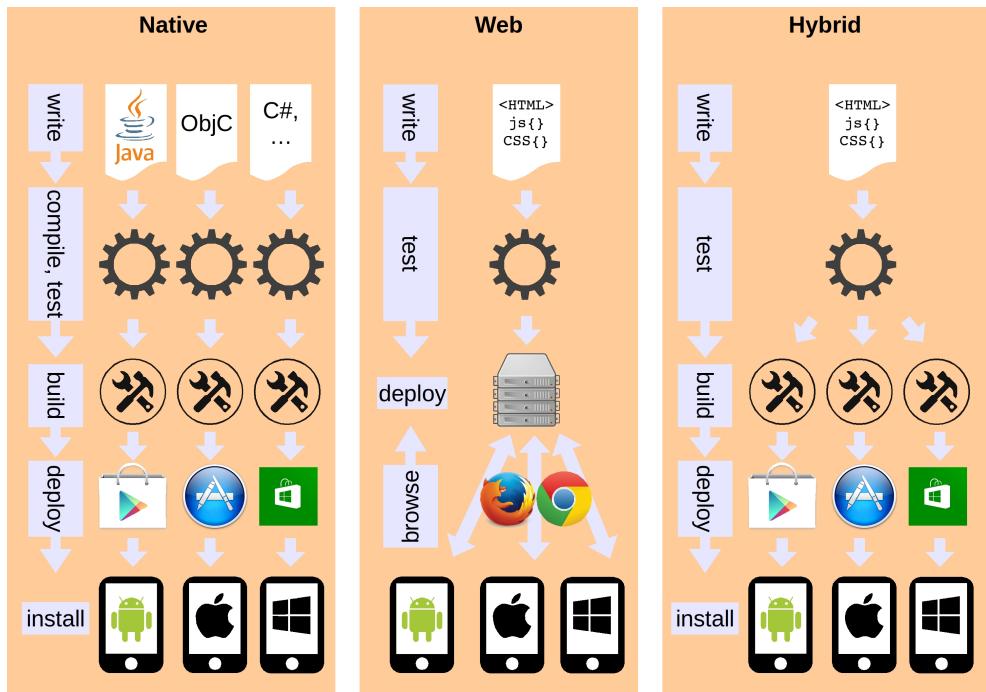


Abbildung 2.1: Entwicklungsstufen der verschiedenen Arten von Apps. Während bei der nativen App der gesamte Entwicklungszyklus einmal pro Plattform durchlaufen werden muss, verringert sich der Aufwand für die Web-App erheblich. Bei der Hybrid-App muss die Anwendung zwar einmal für jede Plattform gebaut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

Quelle: Eigene Grafik.

¹⁰ Beispielsweise unterstützt *PhoneGap Build* in der neusten Version 3 nur noch die drei großen Mobilplattformen Android, iOS, und Windows Phone.

Kapitel 3

Plattformunabhängige App-Entwicklung

3.1 Überblick: Lösungen und Ansätze für die plattformunabhängige App-Entwicklung

Es gibt sehr viele Frameworks und Lösungen zum Erreichen einer plattformunabhängigen App-Entwicklung. Die Unterschiedlichen Ansätze begründen sich in unterschiedlichen Anforderungen an die zu entwickelnde App, unterschiedliche vorhandene Ressourcen und Infrastruktur der Entwickler und App-Betreiber sowie dem Ziel, bestehendes Know-How der Entwickler einbringen zu können. Im Folgenden soll ein Überblick über verschiedene Ansätze geben und konkrete Beispiele genannt werden. Dabei wird die im vorherigen Abschnitt vorgenommenen Gruppierung in native Apps, Web-Apps und hybride-Apps beibehalten.

Wie in Abschnitt 2.2 erwähnt, sind Web-Apps durch die Client-Server-Konzeption, in Verbindung mit Web-Standards, die jeder moderne Browser unterstützt, in gewisser Weise per se plattformunabhängig. Die fehlende Offline-Nutzbarkeit, der eingeschränkte Zugriff auf Hardwarekomponenten wie Gerätetasten, Kamera, GPS-Modul, Mikrofon,¹ der fehlende Zugriff auf Systemdaten wie das Adressbuch, Speicherkarten und das nicht native Systemdesign sind dabei offenkundige Nachteile der Web-Apps, die sie für manche Anwendungsbereiche nicht in Frage kommen lassen.

¹ Diese ist teilweise im noch nicht von allen Browsern voll umgesetzten HTML5-Standard begründet.

Wie in Abschnitt 2.3 beschrieben, stellen Hybrid-Apps einen naheliegenden Ansatz zur plattformunabhängigen App-Entwicklung dar. Durch den Einsatz von Standard-Web-Technologien wie HTML, CSS und JavaScript in Verbindung mit einer in einer nativen Container-App erzeugten WebView, lassen sich schnell viele Plattformen erreichen. Hierzu gibt es Frameworks, die die nativen Container-Apps erstellen und zusätzlich APIs für Hardware-Zugriffe und Geräte- und Systemfunktionen, die nicht durch HTML abgedeckt werden, bereit stellen. Ein häufig genutztes Beispiel hierfür ist das in Unterabschnitt 3.2.4 beschriebene *PhoneGap / Cordova*.

Eine Problematik, die sich bei allen Ansätzen zur plattformunabhängigen App-Entwicklung stellt, sind die in Abschnitt 2.1 erwähnten unterschiedlichen Designs und Designrichtlinien sowie unterschiedlichen Bedienkonzepte. Soll die hybride App also dem *Look-And-Feel* einer nativen sehr nahekommen, so muss zu einem gewissen Grat die Benutzeroberfläche für jede Plattform bspw. mithilfe von Stylesheets separat angepasst werden.

Durch den konsequenten Ansatz von HTML, Design-Elementen über Stylesheets auszulagern, hat der Entwickler hier die Möglichkeit, seine Benutzeroberfläche für jedes System über das laden unterschiedlicher Stylesheets anzupassen. Hierzu bietet beispielsweise *Intel* mit seinem *App Framework* eine JavaScript-Oberflächenbibliothek, der entsprechende Stylesheets für jede große Plattform zugrunde liegen. So wird diese Arbeit dem Entwickler abgenommen und die App soll sich so gut in das native Design einfügen. Es sei allerdings erwähnt, dass weiterhin keine GUI-Elemente nativ gerendert werden, sondern lediglich versucht wird, durch das Anpassen von Farbe, Form und Schrift einen nativen Look zu erzeugen.

Wie in Abschnitt 2.1 beschrieben, kommen für das Entwickeln nativer Apps für jede Plattform andere Programmiersprachen, SDKs und GUI-Toolkits zum Einsatz. Um eine native App plattformunabhängig entwickeln zu können, sind daher Schnittstellen erforderlich, mit denen auf die Funktionen der nativen SDKs zugegriffen werden und eine native App kompiliert werden kann. Im Gegensatz zu hybriden Apps sollen also für die Benutzeroberfläche native Elemente verwendet werden, und keine eigene HTML Oberfläche geschaffen werden. Für die Umsetzung der Funktionslogik gibt es, grob gesprochen, zwei Möglichkeiten. Entweder wird ausgehend von einem Code nativer Bitcode für die jeweilige Plattform kompiliert, sogenanntes *Cross-Compiling*, oder eine Skriptsprache in einer integrierten Runtime-Environment interpretiert bzw. in einer virtuellen Maschine ausgeführt.

Ein Beispiel eines Frameworks, bei dem der gemeinsame Code in einer integrierten Runtime-Environment ausgeführt wird, ist *Titanium* der Firma *Appcelerator*. Programmiert wird in der Titanium-SDK komplett mit JavaScript. Es wird allerdings nicht, wie bei hybriden Apps, eine WebView mit einer Benutzeroberfläche in HTML5 erstellt, sondern der JavaScript Code wird zur Laufzeit in nativen Code übersetzt und dieser ausgeführt. Mit dem *MVC*-Framework *Alloy* steht dabei eine Benutzerschnittstellenabstraktion zur Verfügung, in der die Benutzeroberfläche in JavaScript geschrieben werden kann, und zur Laufzeit die nativen Methoden der jeweiligen Plattform aufgerufen werden.

Ein Cross-Compiling Beispiel ist das von *Xamarin* gesponserte Projekt *Mono*. Mono ist eine *Open-Source-Cross-Plattform-Implementierung* der Programmiersprache *C#*. Mit den Mono nutzenden SDKs *Xamarin.iOS* (*Mono-Touch*) und *Xamarin.Android* können so native Apps für iOS und Android gebaut werden. Für Windows Phone wird standardmäßig in C# programmiert, so dass sich viele Bestandteile des Codes für alle drei Plattformen nutzen lassen. Xamarin bildet so ein Komplettpaket, dass einen gemeinsamen Code für die Funktionslogik, sowie durch eine entsprechende API, für die Benutzeroberfläche ermöglicht. Dabei wird durch den Kompiliervorgang auf die jeweiligen nativen Benutzeroberflächenelemente zurückgegriffen.

Im Bereich Cross-Compiling gibt es etwas allgemeiner Programmiersprachen, die sich in ausführbare Dateien vieler Plattformen, sowie andere Programmiersprachen übersetzen lassen. Ein Beispiel hierfür ist die Open-Source-Multiplattform-Programmiersprache Haxe, deren Code sich in JavaScript .js-Dateien, *Flashs* .swf-Dateien, *Neko VM-Bytecode*, PHP-Dateien, C++, C# und Java-Dateien übersetzen lässt. Zusammen mit den Bibliotheken *Lime* und *OpenFL* kann auf die gesamte Flash-API zugegriffen, und durch integrierte Unterstützung für *SWF*-Dateien die Möglichkeit der Nutzung der *Flash-Editing-Environment* für alle gängigen Plattformen ermöglicht werden. Das Projekt richtet sich hauptsächlich an Spieleentwickler, die so die relativ einfach zu erstellenden Flash-Spiele auf alle Plattformen bringen können.

Ist bereits eine bestehende native App für eine Plattform vorhanden, die für andere Plattformen verfügbar gemacht werden soll, liegt der Ansatz nahe, zu versuchen, den bestehenden Quellcode für eine andere Plattform zu kompilieren. Ein solches Projekt ist *XMLVM*, was durch eine Reihe von Umwandlungsprozessen native iOS-*Builts* aus bestehendem Java-Android-Code erstellt.

Eine weitere Möglichkeit, ein Programm plattformunabhängig zu nutzen, ist, mittels einer Client-Server Konzeption, ähnlich den Web-Apps. Dieser Ansatz setzt natürlich eine Server-Infrastruktur voraus, ist aber beispielsweise für Firmen, die bereits eine Serveranwendung haben, die auf Mobilgeräten nutzbar gemacht werden soll, interessant. Hier soll *Tabris* vorgestellt werden.

Tabris ist ein Framework, mit dem mobile Apps komplett in Java geschrieben werden können und dennoch native Bedienelemente von Android, iOS und HTML5 beinhalten. Die eigentliche Anwendung wird dazu auf einem Server ausgeführt, auf dem die Funktionslogik läuft und alle Daten gespeichert werden. Die Apps auf den Mobilgeräten fungieren dann als Clients, die so die Benutzeroberfläche der Anwendung darstellen. So steht für die Software die volle *JavaEE (Java Plattform Enterprise Edition)* zur Verfügung.

Die Clients greifen auf die Server-Anwendung über eine URL zu, über die sie die Benutzeroberfläche als *JSON* Repräsentation empfangen. Aus dieser wird dann durch den Client mit nativen Komponenten die Benutzeroberfläche gerendert. Diese sind *Cocoa Touch Widgets* in iOS, Java-basierte Widgets in Android und HTML5 in Webbrowsern. Der Anwendungsentwickler muss sich um diesen Mechanismus allerdings nicht weiter kümmern. Das Programm wird als herkömmliche Java-Applikation entwickelt, wobei für die Benutzeroberfläche das *Tabris User Interface Framework (Tabris UI)* zum Einsatz kommt.

Tabris UI baut auf *SWT*, *JFace* und *OSGi* auf und ermöglicht es, im Gegensatz zu reinem *SWT*, auf viele native Bedien- und Navigationselemente zuzugreifen. Solche sind zum Beispiel die *ActionBar* von Android, die *View-Controllers* von Apple (welche für die Verwaltung angezeigten Inhalte in iOS zuständig sind) oder das Gesten-gesteuerte Blättern zwischen Seiten durch eine Wisch-Geste auf dem Display (engl. „swipe“). Weiter erhält der Entwickler über entsprechende APIs einfachen Zugriff auf viele Geräte-Funktionen und Interaktionen mit anderen Systemfunktionen wie beispielsweise Geolokalisierung oder Verknüpfung des Programms mit den Kommunikationsschnittstellen (Telefonieren, SMS, E-Mail, etc.).

3.2 Entwicklung von hybriden Apps

Wie in 2.3 beschrieben, bildet die hybride App-Entwicklung die Schnittmenge aus der nativen App-Entwicklung und der Web-Entwicklung mithilfe von Web-Technologien und zusätzlichen Frameworks und Bibliotheken. Hier soll

mit Cordova / PhoneGap die konkrete Nutzung eines dieser Frameworks und weitere verwendete Technologien wie *Knockout* oder *jQuery Mobile* erläutert werden.

3.2.1 Die JavaScript-Bibliothek jQuery

Bereits für die Entwicklung von reinen Web-Anwendungen stellen neben den grundlegenden Web-Technologien HTML, CSS und JavaScript Erweiterungen wie die JavaScript-Bibliothek *jQuery* nützliche Hilfsmittel dar, die viele Funktionen gegenüber der Verwendung von „reinem“ JavaScript deutlich vereinfacht. So ist beispielsweise der Programmieraufwand für den Zugriff auf Elemente einer HTML-Seite durch jQuery wesentlich geringer als ohne die Bibliothek. Besonders deutlich wird dies an den unten aufgeführten Code-Beispielen, in denen ein Button exemplarisch die Funktionalität übernehmen soll, alle Absätze einer HTML-Seite auszublenden. Während bei herkömmlichem JavaScript für die Selektion aller Elemente die Funktion `getElementsByName()` aufgerufen werden muss (siehe Listing 3.2), ist bei jQuery der Zugriff auf alle Elemente eines *Tags* per Dollar(\$)-Notation deutlich verkürzt (siehe Listing 3.1). Auch die nächste Anweisung zur Ausführung einer Operation für alle ausgewählten Elemente (hier: `hide()`, also *ausblenden*) fällt bei jQuery wesentlich kürzer aus, indem die Funktion noch in der selben Zeile wie der vorherigen Selektor aufgerufen werden kann (`$("p").hide();`). Bei der reinen JavaScript-Variante ist nach der Selektion aller Absätze zunächst einmal ein Array ausgewählt, sodass, um auf den einzelnen Elementen Operationen ausführen zu können, durch alle Elemente des Arrays iteriert und die Funktion für jedes Element aufgerufen werden muss (siehe Listing 3.2, Zeilen 5 - 7).

Listing 3.1: jQuery-Beispiel-Code zum Ausblenden aller Absätze [26].

```

1 <html>
2   <head>
3     <script src="lib/jquery-1.10.2.min.js"></script>
4     <script>
5       $(document).ready(function() {
6         $("button").click(function() {
7           $("p").hide();
8         });
9       });
10    </script>
11  </head>
12  <body>
13    <h2>This is a heading</h2>
14    <p>This is a paragraph.</p>
15    <p>This is another paragraph.</p>
16    <button>Click me</button>
17  </body>
18 </html>
```

Listing 3.2: Die gleiche Funktionalität wie in Listing 3.1 mit reinem JavaScript.

```

1 <html>
2   <head>
3     <script>
4       function hideParagraphs() {
5         var paragraphs = document.getElementsByTagName("p");
6         for (i in paragraphs) {
7           paragraphs[i].style.display = "none";
8         }
9       }
10      </script>
11    </head>
12    <body>
13      <h2>This is a heading</h2>
14      <p>This is a paragraph.</p>
15      <p>This is another paragraph.</p>
16      <button onclick="hideParagraphs()">Click me</button>
17    </body>
18 </html>
```

3.2.2 Data-Binding mit Knockout

Neben jQuery bietet das Knockout-Framework ein weiteres nützliches Hilfsmittel für die Entwicklung von Web-Anwendungen, das ebenso wie jQuery und jQuery Mobile aus einer JavaScript-Bibliothek besteht und die Verbindung der HTML-Oberfläche mit der Programmlogik der Anwendung mittels *Data-Binding*, also der dynamischen Anbindung von *UI*-Komponenten zu Da-

tenfeldern auf Programmebene, erheblich vereinfacht. Das Data-Binding wird bei Knockout durch das *Model-View-ViewModel-Pattern (MVVM)* realisiert, das Entwicklern eine Trennung zwischen Benutzeroberfläche und UI-Logik ermöglicht. Diese Aufteilung dient unter anderem der Übersichtlichkeit des Codes und kann beispielsweise die Aufteilung der Entwicklung von Beninterschnittstellen erleichtern, indem die UI- und die Geschäftslogik von Softwareentwicklern übernommen werden kann, während Designer den Schwerpunkt auf die Gestaltung der Oberfläche legen können [30].

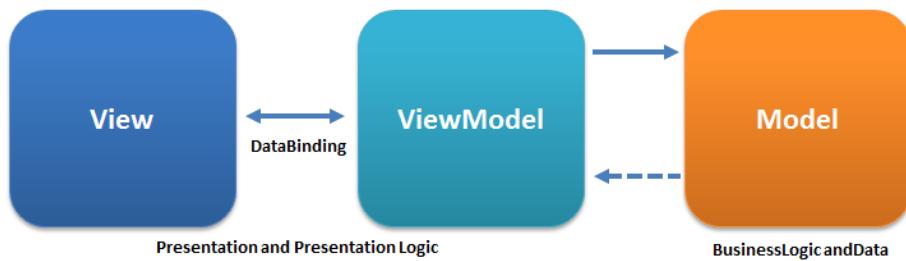


Abbildung 3.1: Schematische Darstellung des Entwurfsmusters MVVM: Die View ist über das Data-Binding mit dem ViewModel verbunden, indem die UI-Logik implementiert ist und das mit der Geschäftslogik (Model) interagiert.

Quelle: Wikimedia Commons [2].

Das MVVM-Pattern stellt eine Gliederung der Software in drei Grundlegende Komponenten dar: Die *View* repräsentiert die Präsentationsschicht, also die Benutzeroberfläche, im Falle der Web-Anwendung also die HTML-Seite, deren Elemente per Data-Binding an Eigenschaften des *ViewModels* gebunden werden können. Das *Model* steht für die Geschäftslogik und beinhaltet das Datenmodell und die Funktionen, die vom ViewModel angefragt werden können, um beispielsweise Benutzereingaben zu validieren oder Daten für die Anzeige in der Oberfläche zu erhalten (siehe Abbildung 3.1).

Ein wesentlicher Mechanismus für die Aktualisierung der Oberfläche bei einer Änderung des ViewModels von Knockout ist die Verwendung von *Observables*, also Objekten oder Datenfeldern, welche bei Änderungen ihres Inhalts eine Nachricht aussenden, sodass andere Objekte automatisch auf die Zustandsänderung des Observables reagieren können, beispielsweise, um die Anzeige auf der Oberfläche zu aktualisieren.

Im Code-Beispiel unten (Listing 3.3) wird ein einfaches ViewModel mit drei Eigenschaften erstellt: `firstName`, `lastName`, und `fullName`, wobei die ersten beiden *Observables* darstellen und letztere aus den anderen beiden Feldern gene-

riert wird (Zeilen 10 - 13). Durch den Aufruf der Knockout-Funktion `observable` () ist es nicht notwendig, bei einer Änderung der Daten an der Oberfläche, die Änderung der Anzeige der Daten (Zeile 27) explizit anzustoßen (Beispielsweise per EventListener auf einer UI-Komponente). Stattdessen übernimmt das Knockout-Framework die Durchreichung aller Änderungen im ViewModel, so dass bei einer Benutzereingabe in eines der `<input>`-Felder eine Änderung der Daten im ViewModel registriert wird und automatisch alle damit verbundenen Views aktualisiert werden (siehe Abbildung 3.2).²

Listing 3.3: Einfaches Anwendungsbeispiel für die Verwendung der JavaScript-Bibliothek Knockout.

```

1 <html>
2   <head>
3     <script src="lib/jquery-1.10.2.min.js"></script>
4     <script src="lib/knockout-3.1.0.js"></script>
5     <script>
6       // jQuery-Standard: Erst ausführen, wenn Dokument geladen.
7       $(document).ready(function() {
8         // ViewModel:
9         var ViewModel = function(first, last) {
10           this.firstName = ko.observable(first);
11           this.lastName = ko.observable(last);
12           this.fullName = ko.computed(function() {
13             return this.firstName() + " " + this.lastName();
14           }, this);
15         };
16         ko.applyBindings(new ViewModel("Planet", "Earth"));
17       });
18     </script>
19   </head>
20   <body>
21     <p>First name:<br/>
22       <input data-bind="value: firstName" />
23     </p>
24     <p>Last name:<br/>
25       <input data-bind="value: lastName" />
26     </p>
27     <h2>Hello, <span data-bind="text: fullName" style="font-size: 2em;"> </span>!</h2>
28   </body>
29 </html>

```

3.2.3 JQuery Mobile: Mobile Web-Oberflächen

Um das Erscheinungsbild und Verhalten von Webseiten an eine bessere Nutzung für mobile Geräte anzupassen, bietet sich der Einsatz eines entsprechenden GUI-Toolkits an. Die von der *jQuery Foundation* entwickelte GUI-

² Hier das `<h2>`-Element, das mit der ViewModel-Eigenschaft `fullName` verknüpft ist.

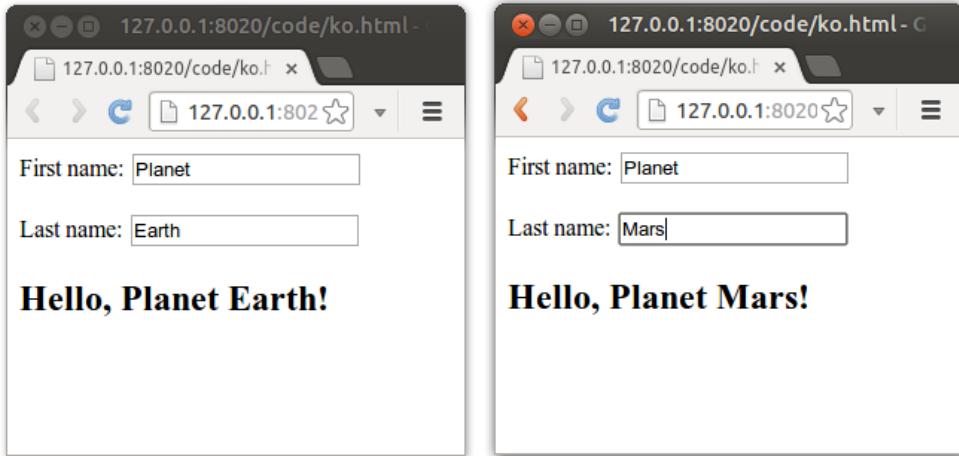


Abbildung 3.2: Knockout-Beispiel aus Listing 3.3 im Browser. Links im Bild: Anzeige bei Initialisierung der Oberfläche, rechts: Benutzereingabe ins Eingabefeld: „Mars“. Änderungen in der UI (Hier: im Eingabefeld) werden sofort im ViewModel registriert und automatisch an alle verknüpften Anzeigen weitergereicht (Hier an das fettgedruckte Begrüßungselement).

Quelle: Eigener Screenshot.

Bibliothek jQuery Mobile bietet hier Möglichkeiten für Entwickler von mobilen Webseiten, ihre Dokumente an verschiedene Eigenschaften anzupassen, die gemeinhin unter dem Begriff *Look-And-Feel* zusammengefasst werden, wie dem äußereren Erscheinungsbild, der Fähigkeit, mit Touch-Gesten umzugehen, der Anpassung an die geringere Display-Größe sowie von vielen mobilen Apps gewohnten Animationen, und somit erwartungskonform zu gestalten.

JQuery Mobile besteht mit einer JavaScript-Bibliothek und einem zusätzlichen Stylesheet in CSS, aus zwei Dokumenten, deren Einbindung in die HTML-Seite analog zu der von jQuery per Verlinkung als `<script>`- beziehungsweise `<link>`-Tag funktioniert. Da jQuery Mobile auf jQuery aufbaut, muss auch der Link zum jQuery-Script gesetzt sein, um auf die benötigten Funktionen zugreifen zu können (siehe Listing 3.4, Zeilen 3 - 5).

Listing 3.4: Einbindung von jQuery Mobile in eine HTML-Seite [27].

```
1 <html>
2   <head>
3     <link rel="stylesheet" href="lib/jquery.mobile-1.4.2.min.css">
4     <script src="lib/jquery-1.10.2.min.js"></script>
5     <script src="lib/jquery.mobile-1.4.2.min.js"></script>
6   </head>
7   <body>
8     <div data-role="page">
9       <div data-role="header">
10         <h1>Welcome To My Homepage</h1>
11       </div>
12       <div data-role="main" class="ui-content">
13         <p>I Am Now A Mobile Developer!!</p>
14       </div>
15       <div data-role="footer">
16         <h1>Footer Text</h1>
17       </div>
18     </div>
19   </body>
20 </html>
```

Die Definition von GUI-Elementen geschieht hierbei über das HTML-Attribut `data-role`, dem vordefinierte Werte wie `page`, `header`, `footer`, `button` u. v. a. m. zugewiesen werden können, anhand derer das jQuery Mobile-Framework den HTML-Elementen die jeweilige Style-Definition aus dem Stylesheet zuweisen kann (siehe Listing 3.4, Zeilen 8, 9, 12 und 15). Somit wird dem Entwickler ermöglicht, ohne zusätzlichen Entwicklungsaufwand für die Programmierung von GUI-Komponenten oder Erstellung von Style-Definitionen Webseiten mit zeitgemäßem und adäquaten Look-And-Feel für mobile Geräte anzupassen (siehe Abbildung 3.3 und 3.5).

Neben der Zuweisung von Rollen über das `data-role`-Attribut, durch die das Framework den GUI-Komponenten automatisch entsprechende Style-Definitionen zuweist, können weiterhin über das `class`-Attribut direkt Style-Klassen aus dem jQuery-Stylesheet verwendet werden. Beispielsweise sorgt im obigen Beispiel der Zusatz `class="ui-content"` (Listing 3.4, Zeile 12) für ein besseres Layout des `<div>`-Inhalts, wie der Vergleich ohne das `class`-Attribut in Abbildung 3.4 zeigt.

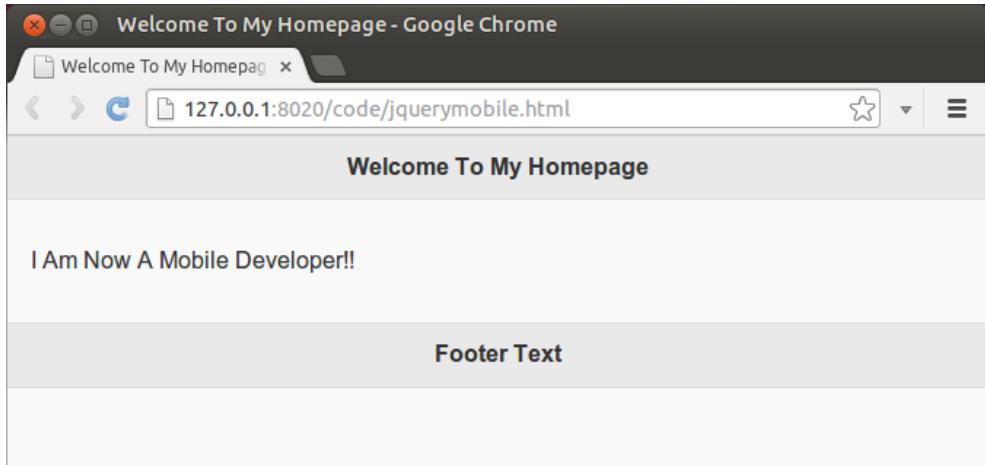


Abbildung 3.3: jQuery Mobile-Beispiel aus Listing 3.4 im Browser.

Quelle: Eigener Screenshot.

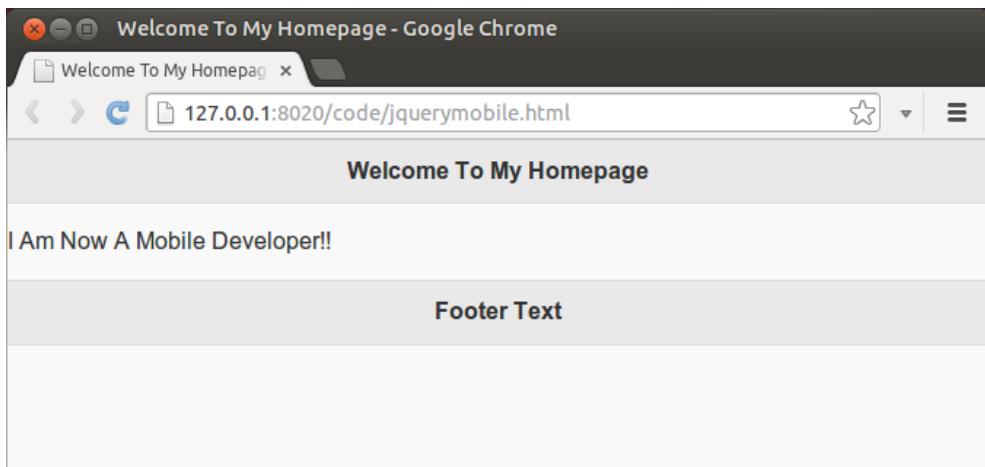


Abbildung 3.4: Beispiel-Oberfläche wie in Abbildung 3.3, ohne das class-Attribut in Listing 3.3, Zeile 12.

Quelle: Eigener Screenshot.

3.2.4 Phonegap / Cordova

3.2.4.1 Grundlegendes

PhoneGap ist ein Open-Source-Framework von *Adobe Systems* zur Erstellung von hybriden Apps und bildet damit die Grundlage für den hier explorierten Ansatz zur plattformunabhängigen App-Entwicklung. Die Software wurde ursprünglich unter dem Namen PhoneGap von der Firma *Nitobi* entwickelt, die 2011 von Adobe aufgekauft wurde [21]. Später wurde die Code-Basis der *Apa-*

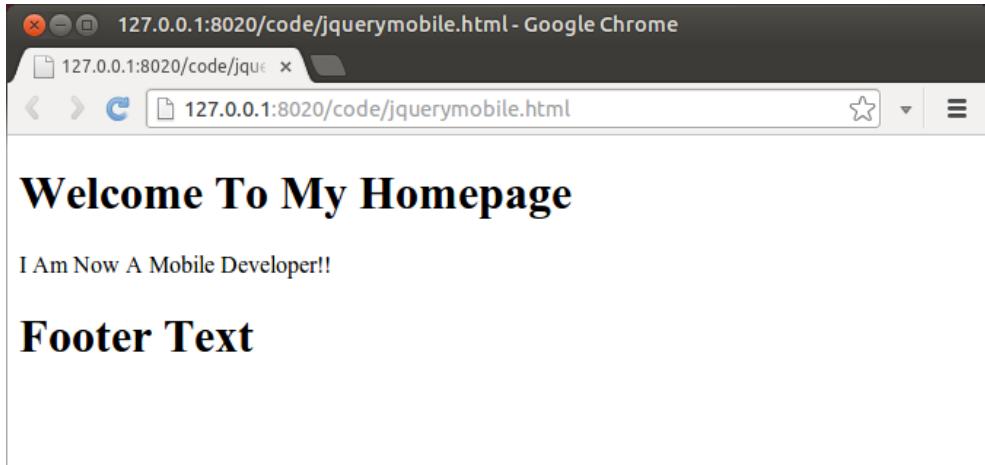


Abbildung 3.5: Beispiel-Oberfläche wie in Abbildung 3.3, aber ohne die jQuery Mobile-Bibliotheken.

Quelle: Eigener Screenshot.

che Software Foundation übergeben und dort in Cordova umbenannt, wodurch das Framework in den Quellen stellenweise unter beiden Namen erscheint. Adobe PhoneGap baut also als dessen Distribution auf Cordova auf, wobei derzeit der einzige wesentliche Unterschied im Namen des Pakets besteht (Stand: 19. März 2012), nach eigenen Angaben aber durchaus weitere Tools mit Bezug auf andere Adobe-Dienste in die PhoneGap-Distribution einfließen können [32].

Da das Apache-Framework als Open-Source-Basis auch die allgemeine Grundlage für weitere Cordova-Distributionen bildet und so auch die Community-Anlaufstelle zur Mitwirkung am Cordova-Projekt darstellt [32], wird im Folgenden weitgehend der Begriff „Cordova“ verwendet, die meisten Inhalte treffen aber ebenso auf die Adobe-Version PhoneGap zu.

Wie in Abschnitt 2.3 beschrieben, wird bei der hybriden App-Entwicklung eine Web-App programmiert, die dann in einen nativen App-Container „verpackt“ werden und somit auf dem jeweiligen Mobilgerät als mobile App ausgeführt werden kann. Das Cordova-Framework besteht darüber hinaus im Wesentlichen aus einer API in Form einer JavaScript-Bibliothek, die es dem Entwickler ermöglicht, auf native Funktionalitäten des mobilen Betriebssystems zuzugreifen sowie einem mitgelieferten *Kommandozeilen-Werkzeug* (*engl. Command-Line Interface, CLI*), das für die Erstellung, Erweiterung und An-

passung der Anwendung, zur Bewerkstelligung des Build-Prozesses für die verschiedenen Plattformen sowie auch der Ausführung in einem Emulator oder auf einem Mobilgerät zum Testen der Anwendung dient [6].

In der Cordova-Dokumentation werden zwei grundlegende Entwicklungs-szenarien beschrieben, für die das Framework verwendet werden kann. Mit der Version 3.0 wurde das CLI Teil des Software-Paktes, das viele Arbeitsschritte automatisiert ausführt und damit den Cordova-Entwicklungsprozess vereinfacht. Der Hauptfokus dieses Werkzeugs liegt im für die hybride App-Entwicklung grundlegenden Entwicklungsworflows, dem *Web-Entwicklungsansatz*. Dieser Ansatz bietet die breiteste Plattformunterstützung bei möglichst geringem Mehraufwand für unterschiedliche Plattformen und bildet damit die Grundlage für den hier hauptsächlich fokussierten Ansatz [7].

Soll nur eine bestimmte Zielplattform bedient werden, kann auch nach dem *Nativen-Plattformansatz* entwickelt werden. Dabei wird das CLI in erster Linie für die Erstellung des Grundgerüsts der App verwendet, dessen Web-Oberfläche und nativer Kern dann mithilfe einer entsprechenden Entwicklungsumgebung in Kombination mit einem SDK der jeweiligen Plattform weiter verarbeitet und kompiliert werden können. Dieser Ansatz kann beispielsweise sinnvoll sein, wenn Entwickler mobiler Apps ihre Kenntnisse im Web-Bereich für die mobile App-Entwicklung nutzen möchten und sehr plattformspezifische Eigenschaften angepasst werden sollen, ist aber aufgrund des Mangels an entsprechenden Tools nicht oder nur gering für die Entwicklung plattformunabhängiger Apps geeignet [7].

Cordova stellt für die Verwendung von nativen Funktionalitäten des mobilen Betriebssystems mit seiner JavaScript-Plattform-API eine Verbindung von der Web-Anwendungsebene zu den jeweiligen SDKs der Zielplattformen her. Somit müssen, um auf plattformspezifische Features zugreifen zu können, auf der Entwicklungsplattform alle SDKs der gewünschten Zielplattformen installiert sein. Da die SDKs jedoch teilweise nur von bestimmten Desktop-Betriebssystemen unterstützt werden, muss das CLI unter Umständen auf mehreren Rechnern ausgeführt werden, die jeweils nur bestimmte mobile Plattformen bedienen (siehe Abbildung 3.6).

	amazon-fireos	android	blackberry10	Firefox OS	ios	Ubuntu	wp7 (Windows Phone 7)	wp8 (Windows Phone 8)	win8 (Windows 8)	tizen
cordova CLI	✓ Mac, Windows, Linux	✓ Mac, Windows, Linux	✓ Mac, Windows	✓ Mac, Windows, Linux	✓ Mac	✓ Ubuntu	✓ Windows	✓ Windows	✓	✗
Embedded WebView	✓ (see details)	✓ (see details)	✗	✗	✓ (see details)	✓	✗	✗	✗	✗
Plug-In Interface	✓ (see details)	✓ (see details)	✓ (see details)	✗	✓ (see details)	✓	✓ (see details)	✓ (see details)	✓	✗
Platform APIs										
Accelerometer	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Camera	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Capture	✓	✓	✓	✗	✓	✓	✓	✓	✗	✗
Compass	✓	✓	✓	✗	✓ (3GS+)	✓	✓	✓	✓	✓
Connection	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
Contacts	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗
Device	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Events	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
File	✓	✓	✓	✗	✓	✓	✓	✓	✓	✗
Geolocation	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Globalization	✓	✓	✗	✗	✓	✓	✓	✓	✗	✗
InAppBrowser	✓	✓	✓	✗	✓	✓	✓	✓	uses Iframe	✗
Media	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
Notification	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
Splashscreen	✓	✓	✓	✗	✓	✓	✓	✓	✓	✗
Storage	✓	✓	✓	✗	✓	✓	✓ localStorage & indexedDB	✓ localStorage & indexedDB	✓ localStorage & indexedDB	✓

Abbildung 3.6: Plattform- und Feature-Unterstützung des Cordova-Frameworks:

Bis auf wenige („kleinere“) mobile Betriebssysteme bietet Cordova auch über die am weitesten verbreiteten Plattformen wie Android, iOS und Windows Phone hinaus die volle Feature-Unterstützung für *Amazon FireOS*, *Ubuntu Phone* und eine fast vollständige für *Blackberry 10*.

Für Entwickler von hybriden Apps dürfte hier auch vor allem die erste Zeile *cordova CLI* interessant sein, da durch die Kompatibilität der jeweiligen Plattform-SDKs nur bestimmte Kombinationen von Entwicklungs- und Zielplattform möglich sind.

Quelle: Screenshot aus der Cordova-Dokumentation [8].

3.2.4.2 Funktionsweise

Nachdem das Cordova-Framework auf dem Entwicklungsrechner installiert ist, kann mit dem CLI ein neues App-Projekt angelegt werden. Dazu muss auf der Kommandozeile in das Entwicklungsverzeichnis für das zu erstellende Projekt navigiert und das Cordova-Tool mit dem `create`-Befehl aufgerufen werden (siehe Listing 3.5).

Listing 3.5: Befehl zum Erstellen einer Cordova-App.

```
1 $ cordova create hello-beuth de.beuth-hochschule.hello HelloBeuth
```

Das erste Argument `hello-beuth` spezifiziert dabei den Namen des Ordners für das zu erstellende App-Projekt, der mit dem `create`-Befehl angelegt wird. Die anderen beiden Parameter sind optional und geben mit der *ID* in der rückwärts geschriebenen Domain-Bezeichnung und dem Namen („HelloBeuth“), der später für die App angezeigt wird, weitere Informationen für die App an, die in einer *XML*-Datei im neu angelegten Projekt-Ordner gespeichert werden [7].

Mit der Ausführung dieses Skripts erstellt das Tool in einem neuen Ordner ein Grundgerüst für die App, das die nötige Struktur für den weiteren Entwicklungsprozess enthält (siehe Abbildung 3.7). Auf oberster Ebene liegt die Konfigurationsdatei `config.xml`, in der grundlegende Eigenschaften sowie Informationen für den Build-Prozess hinterlegt werden können (siehe Listing 3.6). Daneben werden unter anderem (zu diesem Zeitpunkt noch leere) Ordner für *Plugins* und plattformspezifischen Code sowie ein `www`-Ordner angelegt, der das Grundgerüst für den Web-Teil der Hybrid-App beinhaltet. Neben der Datei `index.html`, die die Beschreibung der Web-Oberfläche für die Anwendung darstellt, werden nach üblichen Konventionen in der Webentwicklung weiterhin die Unterordner `js`, `css` und `img` angelegt, worin die zur App gehörigen JavaScript- und CSS-Dateien bzw. Bilder gespeichert werden [7].

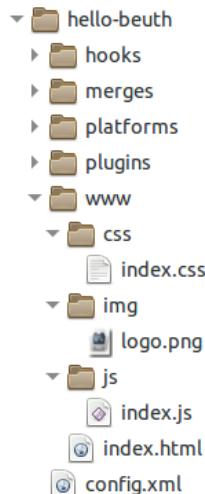


Abbildung 3.7: Dateistruktur einer mit Cordova erstellten App. Das Grundgerüst für die Hybrid-App wird automatisch angelegt und entspricht den üblichen Web-Entwicklungskonventionen.

Quelle: Eigener Screenshot.

Listing 3.6: Die Konfigurationsdatei config.xml, in der allgemeine Informationen über die Hybrid-App gespeichert werden. In Zeile 2 und 4 tauchen die in Listing 3.5 angegebenen optionalen Parameter *Id* und *Name* wieder auf.

```
1 <?xml version='1.0' encoding='utf-8'?>
2 <widget id="de.beuth-hochschule.hello" version="0.0.1"
3   xmlns="http://www.w3.org/ns/widgets" xmlns:cdv="http://
        cordova.apache.org/ns/1.0">
4   <name>HelloBeuth</name>
5   <description>
6     A sample Apache Cordova application that responds to the deviceready
7     event.
8   </description>
9   <author email="dev@cordova.apache.org" href="http://cordova.io">
10    Apache Cordova Team
11   </author>
12   <content src="index.html" />
13   <access origin="*" />
14 </widget>
```

Die Initialisierung der App erfolgt über den deviceready-Eventhandler, der standardmäßig von www/js/index.js referenziert wird (siehe Listing 3.7, Zeile 21 und Listing 3.8, Zeilen 11 u. 12).

Das in Zeile 18 referenzierte Script `cordova.js` stellt die o. g. wesentliche API zur nativen Betriebssystemebene dar, ist hier im www-Ordner jedoch noch nicht vorhanden, sondern wird erst nach Ausführung des `build`-Befehls in seiner jeweiligen plattformspezifischen Ausführung in das entsprechende Unterverzeichnis im Ordner `platforms` eingefügt.

Listing 3.7: Startseite der von Cordova erzeugten App, die auf das deviceready-Event reagiert.

```
1 <html>
2   <head>
3     <meta charset="utf-8" />
4     <meta name="format-detection" content="telephone=no" />
5     <!-- WARNING: for iOS 7, remove the width=device-width and
       height=device-height attributes. See https://issues.apache.org/jira/
       browse/CB-4323 -->
6     <meta name="viewport" content="user-scalable=no, initial-scale=1,
      maximum-scale=1, minimum-scale=1, width=device-width,
      height=device-height, target-densitydpi=device-dpi" />
7     <link rel="stylesheet" type="text/css" href="css/index.css" />
8     <title>Hello World</title>
9   </head>
10  <body>
11    <div class="app">
12      <h1>Hello Beuth</h1>
13      <div id="deviceready" class="blink">
14        <p class="event listening">Connecting to Device</p>
15        <p class="event received">Device is Ready</p>
16      </div>
17    </div>
18    <script type="text/javascript" src="cordova.js"></script>
19    <script type="text/javascript" src="js/index.js"></script>
20    <script type="text/javascript">
21      app.initialize();
22    </script>
23  </body>
24 </html>
```

Listing 3.8: Standardmäßig von Cordova angelegte JavaScript-Datei index.js

```

1 var app = {
2   // Application Constructor
3   initialize : function() {
4     this.bindEvents();
5   },
6   // Bind Event Listeners
7   //
8   // Bind any events that are required on startup. Common events are:
9   // 'load', 'deviceready', 'offline', and 'online'.
10  bindEvents : function() {
11    document.addEventListener('deviceready', this.onDeviceReady, false);
12  },
13  // deviceready Event Handler
14  //
15  // The scope of 'this' is the event. In order to call the
16  // 'receivedEvent'
17  // function, we must explicitly call 'app.receivedEvent(...);'
18  onDeviceReady : function() {
19    app.receivedEvent('deviceready');
20  },
21  // Update DOM on a Received Event
22  receivedEvent : function(id) {
23    var parentElement = document.getElementById(id);
24    var listeningElement = parentElement.querySelector('.listening');
25    var receivedElement = parentElement.querySelector('.received');

26    listeningElement.setAttribute('style', 'display:none;');
27    receivedElement.setAttribute('style', 'display:block;');
28
29    console.log('Received Event: ' + id);
30  }
31 };

```

Um die Oberfläche der App anzuzeigen, lässt sie sich, da sie im Wesentlichen aus einer HTML-Seite besteht, in einem herkömmlichen Browser öffnen (siehe Abbildung 3.8). Da jedoch das bei der hybriden App darunterliegende mobile Betriebssystem hier nicht zur Verfügung steht, ist hier auch keine Anbindung an dessen Funktionalitäten möglich. Dafür muss die Anwendung entweder direkt auf einem mobilen Gerät, dessen Plattform die App und das Cordova-API unterstützen, oder mithilfe eines Emulators ausgeführt werden (siehe Abbildung 3.9) [7].

Bevor die Hybrid-App mit Cordova zum Laufen gebracht werden kann, muss sie, wie die meisten Computerprogramme, in ein ausführbares Format überführt, also *gebaut* werden. Im Falle der Hybrid-App bedeutet das, dass diejenigen plattformspezifischen Komponenten der App hinzugefügt werden,

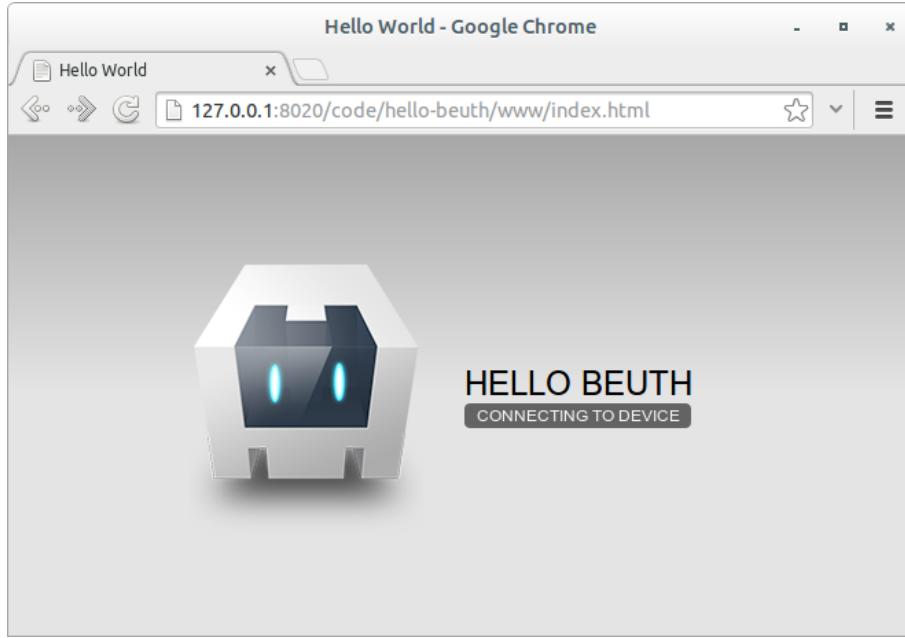


Abbildung 3.8: Die Startseite der Beispiel-App aus Listing 3.7 lässt sich auch im Desktop-Browser öffnen und anzeigen, allerdings kann hier kein `deviceready`-Event empfangen werden.

Quelle: Eigener Screenshot.

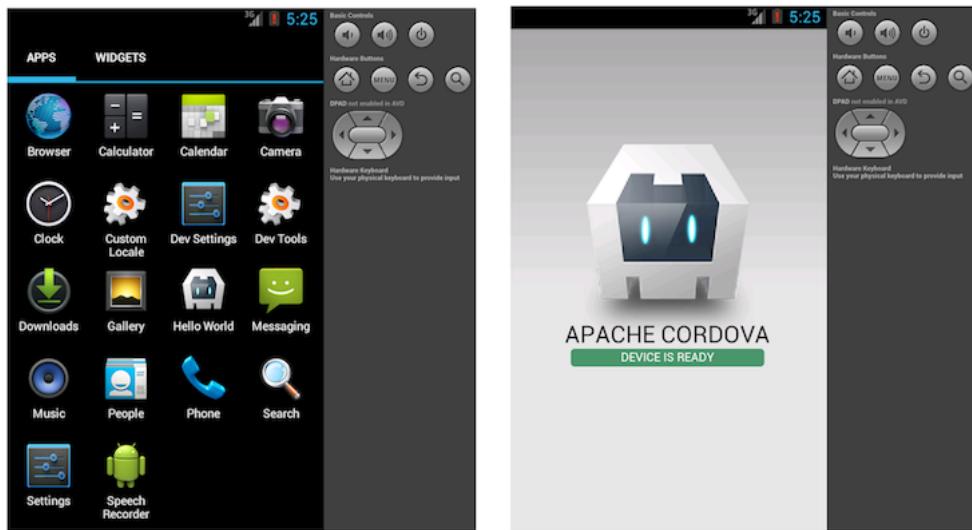


Abbildung 3.9: Anzeige in der App-Übersicht (links) und Ausführung der Cordova-Beispiel-App in einem Android-Emulator (rechts). Das grüne Label zeigt den Empfang des `deviceready`-Events an.

Quelle: Cordova-Dokumentation [1]

die nötig sind, um die Verbindung zwischen der plattformunabhängigen Web-Schicht und der nativen Schicht des jeweiligen Zielbetriebssystems herzustellen.

Damit das Cordova-Framework die entsprechenden Schnittstellen in die Anwendung einfügen kann, muss vor dem Build-Prozess ein Satz an Zielplattformen angegeben werden. Voraussetzung hierfür ist, dass die SDKs der jeweiligen Zielplattformen zu der verwendeten Entwicklungsplattform kompatibel und installiert sind [7].

Über den Cordova-Befehl `platform` lassen sich mit den Optionen `add` und `remove` Plattformen zur Projektkonfiguration der Anwendung hinzufügen bzw. entfernen (siehe Listing 3.9). Die Ausführung dieser Befehle wirkt sich auf den Inhalt des `platforms`-Ordners innerhalb der Projektstruktur aus [7].

Listing 3.9: Cordova-Skript um Plattformen zum Projekt hinzuzufügen bzw. zu entfernen. Zeile 1 fügt die Plattform Android hinzu, Zeile 2 entfernt diese wieder.

```
1 $ cordova platform add android
2 $ cordova platform remove android
```

Der `list`-Befehl dient dazu, eine Liste aller Plattformen des Projekts auszugeben. Wie bei den meisten anderen Befehlen auch, kann synonym auch eine Kurzschreibweise (`ls`) verwendet werden (siehe Listing 3.10).

Listing 3.10: Cordova-Skript zum Auflisten aller Plattformen des Projekts.

```
1 $ cordova platforms list
```

Anschließend kann mit dem `build`-Befehl die App gebaut werden:

Listing 3.11: Bauen der Cordova-App für Android.

```
1 $ cordova build android
```

Der `build`-Befehl stellt dabei eine Kurzform für die beiden Befehle `prepare` und `compile` dar (siehe Listing 3.12). Diese Aufteilung kann beispielsweise sinnvoll sein, um erst den `prepare`-Befehl auszuführen und den plattformspezifischen Code, den Cordova in `platforms/android` generiert, anschließend mit einer entsprechenden Entwicklungsumgebung (wie beispielsweise Android Studio) und deren nativem SDK anzupassen und zu kompilieren.³

³ vgl. Nativ-Entwicklungsansatz, siehe Unterunterabschnitt 3.2.4.1.

Listing 3.12: Ausführlichere Schreibweise für den `build`-Befehl aus Listing 3.11.

```
1 $ cordova prepare android
2 $ cordova compile android
```

Ist der Build-Prozess abgeschlossen, kann die fertige App auch mit dem Cordova-CLI getestet werden. Die Anwendung kann dafür entweder an einen Emulator, der in vielen Fällen mit dem jeweiligen SDK mitgeliefert wird, übergeben (siehe Listing 3.13) oder direkt auf einem an den Entwicklungsrechner angeschlossenen Mobilgerät ausgeführt werden (siehe Listing 3.14). Für letzteres müssen unter Umständen noch entsprechende Einstellungen auf dem Gerät vorgenommen werden [7].

Listing 3.13: Übergibt die App an den Emulator des Android-SDKs.

```
1 $ cordova emulate android
```

Listing 3.14: Führt die App auf einem angeschlossenen Mobilgerät aus.

```
1 $ cordova run android
```

3.2.4.3 Schnittstelle zur mobilen Plattform

Um den eigentlichen Mehrwert des Cordova-Frameworks zu nutzen, also auf native Features der Zielplattformen zuzugreifen, sind verschiedene Plugins nötig, die ebenfalls mit dem CLI verwaltet werden können. Plugins bestehen aus einem zusätzlichen Stück Code, der die Kommunikation zu den nativen Komponenten herstellt. Cordova bietet von Haus aus einen Basissatz von ca. 17 Kern-Plugins an,⁴ es können aber auch eigene entwickelt werden.⁵ Für die Erstellung und Veröffentlichung eigener Plugins stellt Apache in seiner Cordova-Dokumentation eine Entwicklungsanleitung bereit [5].

⁴ In den Quellen finden sich teilweise unterschiedliche Informationen über die Anzahl der Basis-Plugins. In der offiziellen Cordova-Dokumentation werden folgende aufgelistet: *Battery Status, Camera, Contacts, Device, Device Motion (Accelerometer), Device Orientation (Compass), Dialogs, FileSystem, File Transfer, Geolocation, Globalization, InApp-Browser, Media, Media Capture, Network Information (Connection), Splashscreen und Vibration* [4]. Auf der Cordova-Homepage finden sich zudem noch die beiden Einträge *Console* und *Statusbar* [37].

⁵ So bietet z.B. Adobe mit seiner PhoneGap-Variante noch einige weitere Plugins wie beispielsweise den Zugriff auf einen angeschlossenen Barcode-Scanner an, die über die Grundausstattung von Cordova hinausgehen [7].

Einrichten von Plugins: Für die Installation und Dokumentation von Plugins bietet Apache mit der *Apache Cordova Plugin Registry* ein Online-Portal an, in der sich zur Zeit 229 Plugins für das Cordova-Framework finden, darunter auch die o. g. Kern-Plugins von Apache [4]. Der überwiegende übrige Teil stammt von verschiedenen Entwicklern und Drittanbietern, die ihre nach dem *Plugin Development Guide* entwickelten Plugins diesem Portal hinzufügen können [5], und bieten häufig spezielle Unterstützung für bestimmte mobile Plattformen an [10].

Das Cordova-CLI bietet auch die Möglichkeit, alle verfügbaren Plugins zu durchsuchen, hinzuzufügen, aufzulisten und zu entfernen (siehe Listing 3.15, 3.16, 3.17 und 3.18).

Listing 3.15: Suchen von verfügbaren Plugins.

```
1 $ cordova plugin search contacts
```

Listing 3.16: Plugin zur App hinzufügen.

```
1 $ cordova plugin add org.apache.cordova.contacts
```

Listing 3.17: Analog zum `list`-Befehl für Plattformen (siehe Listing 3.10) können auch Plugins des aktuellen Projekts aufgelistet werden.

```
1 $ cordova plugin ls
```

Listing 3.18: Plugin entfernen.

```
1 $ cordova plugin rm org.apache.cordova.contacts
```

Verwendung von Plugins Cordova-Plugins können einen Satz von Methoden und Objekttypen mitbringen, mit deren Hilfe Entwickler die Funktionen der jeweiligen Plugins nutzen können. So beinhaltet beispielsweise das `contacts`-Plugin für den Zugriff auf die native Adressverwaltung des Geräts neben dem zentralen `Contact`-Objekt, das eine Instanz eines Eintrags im Adressbuch des Geräts repräsentiert, die Typen `ContactName`, `ContactField`, `ContactAddress`, `ContactOrganization`, die einige der Attribute des `Contact`-Objekts darstellen sowie das Konfigurationsobjekt `ContactFindOptions`, das bestimmte Optionen für den Zugriff auf das Adressbuch kapselt und einen `ContactError`, welcher bei Auftreten eines Fehlers, beispielsweise bei der Suche nach Kontakten, behandelt werden kann [13].

Darüber hinaus können hier die beiden Methoden `navigator.contacts.create` und `navigator.contacts.find` verwendet werden, um neue Kontaktobjekte zu erstellen bzw. zu suchen (siehe Abschnitt 5.1). Das Präfix `navigator` identifiziert ein elementares Objekt des JavaScript-Kerns, das in der Webentwicklung Informationen über den verwendeten Webbrower bereitstellt, wie beispielsweise den Namen oder dessen Version sowie Informationen über aktivierte Browser-plugins [34]. Das Cordova-Framework ersetzt mittels seiner zentralen, plattformspezifischen JavaScript-Datei `cordova.js` das JavaScript-Objekt `navigator` durch einen eigenen `CordovaNavigator` [9], dem dann durch die verschiedenen Plugins neue Eigenschaften und Methoden zugewiesen werden können über die der Zugriff auf verschiedene Plugins bewerkstelligt werden kann, wie in diesem Beispiel das `contacts`-Objekt, welches die beiden Plugin-Methoden `create` und `find` beinhaltet. Der o. g. JavaScript-Objekttyp `Contact` ist in der *Cordova Contacts API* definiert und beinhaltet die Felder `id`, `displayName`, `name`, `nickname`, `phoneNumbers`, `code`, `addresses`, `ims`, `organizations`, `birthday`, `note`, `photos`, `categories` und `urls` sowie die Methoden `clone` zum duplizieren von Kontaktinstanzen, `remove`, um Kontakte aus dem Adressbuch zu entfernen und `save`, mit der sich neu erstellte Kontakte in der Adressverwaltung des Geräts abspeichern lassen [18].

Je nach Aufgabengebiet unterscheiden sich die verschiedenen Plugins in ihrer Verwendung. Während bei einigen, wie z. B. dem Cordova-Plugin, Schnittstellenmethoden über das `navigator`-Objekt bereitgestellt werden (siehe Listing 3.19), bieten andere wie bspw. das *Battery-Status*-Plugin die Behandlung verschiedener Events an,⁶ die über das Hinzufügen eines *Event-Handlers* zum `window`-Objekt⁷ verarbeitet werden können (siehe Listing 3.20) und wieder andere, wie z. B. das *Device*-Plugin stellen ein bestimmtes Objekt bereit, über das in diesem Fall Informationen abgefragt werden können [11–13].⁸

Listing 3.19: Beispiel für die Verwendung des *Contacts*-Plugins für die Erstellung eines neuen `Contact`-Objekts [13].

```
1 var myContact = navigator.contacts.create({  
2   "displayName" : "Test User"  
3 });
```

⁶ Im Beispiel des *Battery-Status*-Plugins geben diese Auskunft über eine Statusänderung des Ladezustands oder den Anschluss an ans Stromnetz [11].

⁷ Ähnlich wie das `navigator`-Objekt ist auch das `window`-Objekt Bestandteil der JavaScript-Spezifikation, das Eigenschaften und Methoden zur Information und Steuerung des Browser-Fensters bietet [35].

⁸ In diesem Beispiel über das verwendete Gerät (siehe Listing 3.21).

Listing 3.20: Beispiel für die Verwendung des *Battery-Status*-Plugins [11].

```

1 window.addEventListener("batterystatus", onBatteryStatus, false);
2
3 function onBatteryStatus(info) {
4     // Handle the online event
5     console.log("Level: " + info.level + " isPlugged: " + info.isPlugged);
6 }
```

Listing 3.21: Beispiel für die Verwendung des *Device*-Plugins. Bei Empfang des *DeviceReady*-Events wird die Modell-Bezeichnung des Geräts in der Konsole ausgegeben [12].

```

1 document.addEventListener("deviceready", onDeviceReady, false);
2 function onDeviceReady() {
3     console.log(device.model);
4 }
```

3.2.4.4 PhoneGap Build

Mit seinem Online-Portal PhoneGap Build bietet Adobe einen webbasierten Build-Service für PhoneGap-Apps an, der den Bau-Prozess, im Falle der hybriden Apps also die eigentliche Anbindung an plattformspezifische Toolkits auf nativer Ebene sowie das Einbetten der Web-App in eine native WebView, auslagert und damit deutlich vereinfacht.

Wie in Abbildung 2.1 dargestellt, muss dieser Entwicklungsschritt im Gegensatz zu den vorherigen⁹ trotz (oder gerade wegen) des plattformunabhängigen Charakters mehrfach (also einmal für jede Zielplattform) ausgeführt werden, um die vorher entwickelte Web-App in die Umgebung einer nativen App einzubetten (siehe Abschnitt 2.3). Dabei kann unter anderem besonderer Mehraufwand auf der administrativen Ebene entstehen, da, wie im Unterabschnitt 3.2.4 beschrieben, nicht jede Entwicklungsplattform zu den gewünschten Zielplattformen kompatibel ist, und somit für die Verwaltung der verschiedenen SDKs und das Bauen der jeweiligen hybriden Apps der Einsatz mehrerer Entwicklungsplattformen nötig sein kann, sofern ein breites Spektrum an Zielplattformen angestrebt wird.

Um also nicht auf gar mehreren Entwicklungsrechnern sämtliche Toolkits aller erforderlichen Zielplattformen verwalten zu müssen, können PhoneGap-Entwickler hier ihren Quellcode hochladen und die App für die verschiedenen Zielplattformen im jeweiligen Format zusammenbauen bauen lassen. Allerdings bietet auch PhoneGap Build nicht für sämtliche mobilen Plattformen,

⁹ Also der Entwicklung des Codes und der Oberfläche sowie Tests.

die von Cordova / PhoneGap unterstützt werden, seinen Service an. Während bis Version 2.9 noch Apps für iOS, Android, Windows Phone, Blackberry 10, *webOS* und *Symbian* in dem Portal gebaut werden konnten, sind ab der Version 3.0 nur die drei größten Betriebssysteme iOS, Android und Windows Phone verfügbar [19].

Um den Build-Prozess über den Cloud-Build-Service einzuleiten, erstellt der Entwickler eine Web-App in seiner gewohnten Entwicklungsumgebung in Form von HTML-, JavaScript- und CSS-Dokumenten, die er anschließend auf PhoneGap Build hochlädt. Hierfür muss zunächst über die Web-Oberfläche des Portals eine neue App angelegt werden.

Abhängig vom jeweiligen Bezahlpaket haben PhoneGap-Entwickler die Möglichkeit, ihre Anwendung als private oder öffentliche App hochzuladen. Während private Apps entweder als Zip-Paket hochgeladen oder als Link zu einem *Git*-Repository¹⁰ in PhoneGap Build eingestellt werden können (siehe Abbildung 3.10), müssen öffentliche (also Open-Source-) Apps in Form eines GitHub-Repositorys zugänglich gemacht werden.

„We only allow open-source apps to be built from public Github repos“ [22]

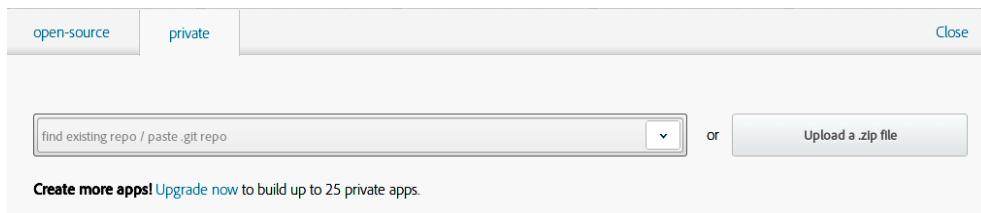


Abbildung 3.10: Dialog für die Erstellung einer neuen privaten App auf PhoneGap Build. Links das Eingabefeld zum Eintragen eines Git-Repository-Links und rechts der Button zum Hochladen von Zip-Archiven.

Quelle: Eigener Screenshot.

Neben der Bezahlvariante gibt es auch ein kostenloses Paket, das eine private App beinhaltet, bei der kostenpflichtigen Variante sind bis zu 25 private App enthalten. Open-Source-Apps können bei allen Paketen unbegrenzt angelegt werden (siehe Abbildung 3.11).

¹⁰ Beispielsweise wie die öffentlichen Apps auf *GitHub* gehosted oder einem eigenen Git-Repository

Choose your plan		
	Free Plan	Paid Plan
open source apps		∞ unlimited must be pulled from a public Github repo
collaborators		∞ unlimited invite people to your app as either developers or testers
private apps ?	1	25
	completely free	starting at \$9.99/mo
		sign in with your Adobe ID

Abbildung 3.11: Übersicht über die verschiedenen Bezahlpakete von PhoneGap Build: Ab 9,99 € im Monat können Entwickler bis zu 25 private Apps anlegen, in der kostenlosen Variante nur eine private, aber unbegrenzt öffentliche.

Quelle: Eigener Screenshot [23].

Nach dem Hochladen des Codes auf PhoneGap Build, wird dort automatisch der Build-Prozess für die Hybrid-Apps initiiert. Dabei sorgt PhoneGap Build dafür, dass für jede Plattform die entsprechende PhoneGap-JavaScript-Bibliothek injiziert wird, welche die API zur nativen Betriebssystemebene enthält (siehe Unterabschnitt 3.2.4) [20]. Ist der Build-Prozess abgeschlossen, kann die App im jeweiligen Format für die verschiedenen Plattformen als Direkt-Link oder per *QR-Code* heruntergeladen werden (siehe Abbildung 3.12 und 3.13).

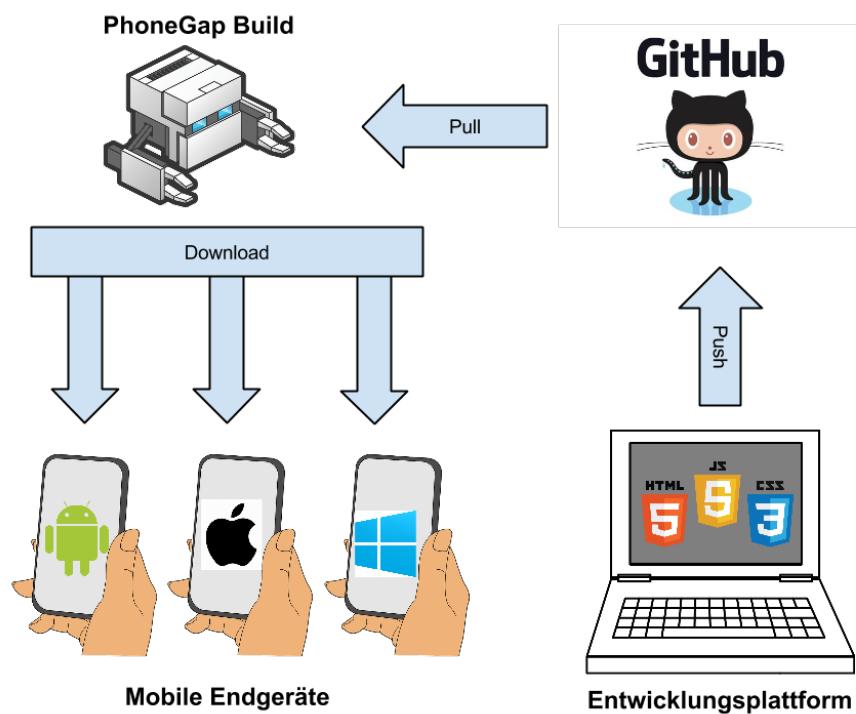


Abbildung 3.12: Schematische Darstellung eines möglichen Entwicklungsworkflows mit PhoneGap Build: Der Code wird als Web-Anwendung auf GitHub hochgeladen, bei PhoneGap Build über das GitHub-Repository aktualisiert und für die jeweiligen Plattformen gebaut, sodass die plattformspezifischen Apps heruntergeladen und auf den Zielgeräten installiert werden können.

Quelle: Eigene Grafik.

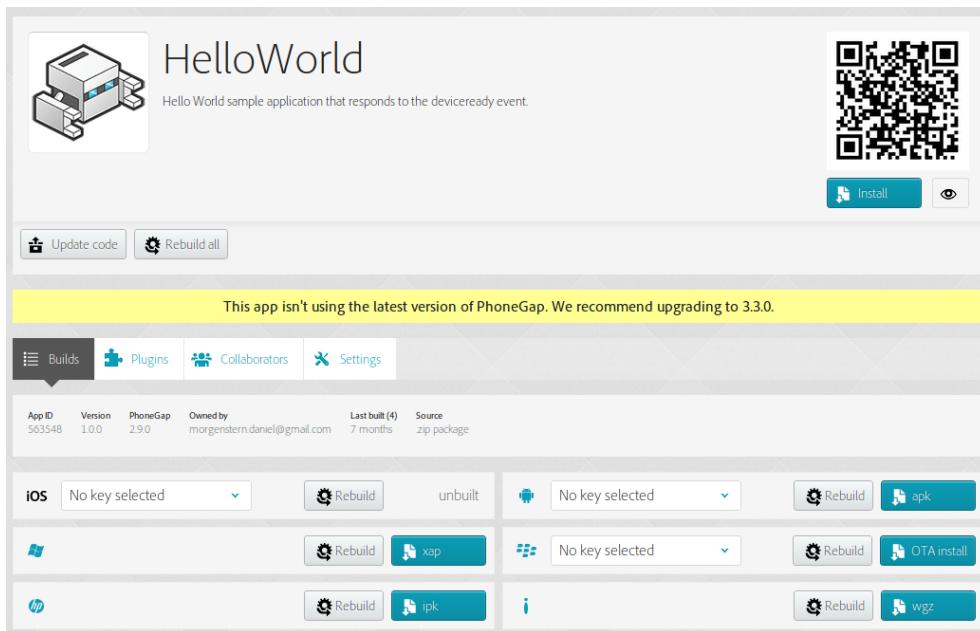


Abbildung 3.13: Oberfläche des PhoneGap Build-Portals: Detailansicht für eine Beispiel-App. Hier können verschiedene Einstellungen vorgenommen werden, sowie der Code aus einem Repository aktualisiert und die fertig gebaute App per Download-Button oder QR-Code heruntergeladen werden.

Quelle: Eigener Screenshot.



Abbildung 3.14: Die Beispiel-App („HelloWorld“) aus Abbildung 3.13 aus PhoneGap Build auf einem Android-Gerät installiert.

Quelle: Eigener Screenshot.

Teil II

Praktische Umsetzung

Kapitel 4

Konzeption

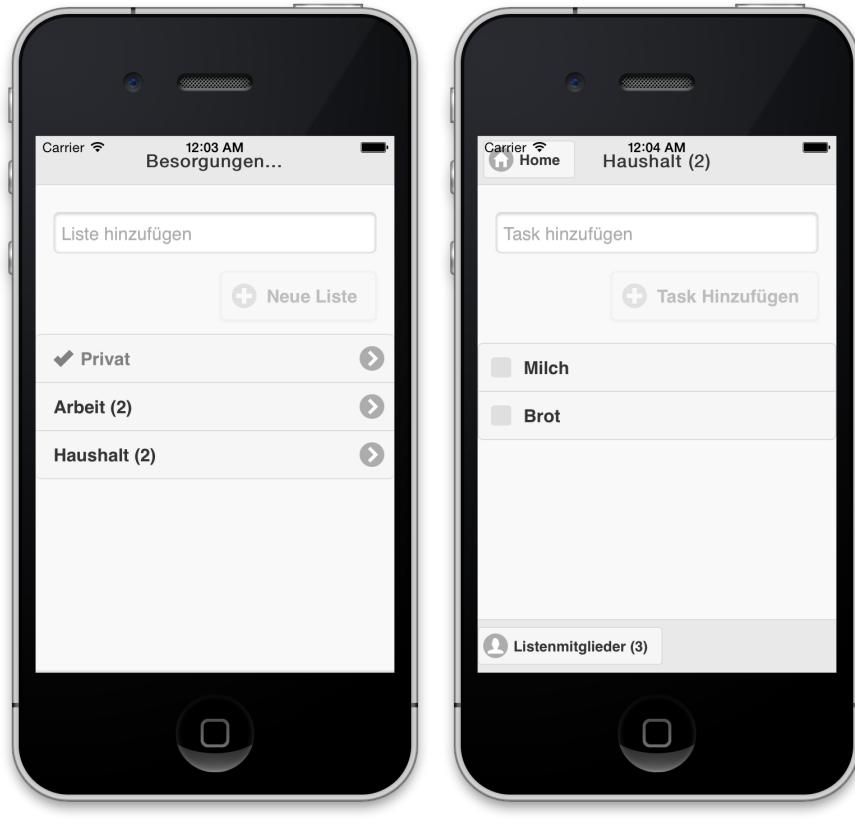
4.1 Vorüberlegungen

4.2 Beispiel-Anwendung

Für die praktische Erprobung und Untersuchung einer Technologie zur plattformunabhängigen App-Entwicklung sollte hier mit Cordova ansatzweise eine Beispiel-Anwendung in Form einer plattformunabhängigen mobilen App entwickelt werden. Für die Auswahl eines geeigneten Anwendungsgebiets war ein Kriterium die Maßgabe, möglichst solche Anforderungen der Anwendung zu identifizieren, die eine gewisse plattformspezifische Anforderung haben, also („plattformkritische“) Features benötigen, die sich beispielsweise nicht ohne weiteres mit einer reinen Web-Anwendung umsetzen lassen, um den eigentlichen Mehrwert eines entsprechenden Cross-Plattform-Frameworks nutzen zu können und aufzuzeigen.

Dabei soll der Fokus nicht speziell auf der Umsetzung eines möglichst breiten Funktionsumfangs der Anwendung selbst oder deren Usability liegen, sondern vielmehr anhand einer beispielhaften Implementierung weniger grundlegender plattformkritischer Features die Verwendung der Cross-Plattform-Technologie erläutert werden, um Möglichkeiten und Grenzen aus Sicht des Entwicklers aufzuzeigen.

Die Beispiel-App trägt den Titel „Besorgungen“, soll also für den Anwender ein Hilfswerkzeug für die Erledigung alltäglicher Aufgaben darstellen. Letztendlich handelt es sich um eine etwas komplexere ToDo-Liste, in der sogenannte „Tasks“ (also Aufgaben, Besorgungen) in Listen kategorisiert und verwaltet werden können. Diese Listen können vom Nutzer selbst angelegt, gelöscht und verändert werden. Um von der Schnittstelle zum Geräte-Adressbuch gebraucht



(a) Startbildschirm der Beispielanwendung

(b) Listenansicht der Beispielanwendung

Abbildung 4.1: Startbildschirm und Listenansicht der Beispiel-Anwendung im *iOS-Simulator*.

Quelle: Eigener Screenshot.

zu machen, können Listen mit im Adressbuch vorhandenen Kontakten geteilt werden, d. h. es sollen Kontakte aus dem Adressbuch geladen und einer Liste als neues Listenmitglied hinzugefügt werden.

Weiterhin soll ein grundlegendes Prinzip darin bestehen, dass Tasks vielerlei Eigenschaften zugewiesen bekommen können. So zum Beispiel Fotos, die von der Kamera des Geräts aufgenommen werden sollen, Ortsangaben, um GPS-Daten zu empfangen und zu verarbeiten sowie Datumsangaben für die Interaktion mit dem nativen Kalender. Außerdem soll die Anwendung Benachrichtigungen über das Erreichen eines angegebenen Datums oder Ortes per nativem Benachrichtigungsmechanismus des jeweiligen Betriebssystems aussenden.

den können.¹ Der Entwicklungsstand des hier beschriebenen Beispiel-Codes beschränkt sich jedoch auf die Umsetzung der o. g. Funktionalität, Listen zu teilen, also neue Listenmitglieder anhand der Daten aus der Kontaktverwaltung des Geräts zu Listenobjekten hinzuzufügen.

4.3 Ausgewählte Technologie und Architektur für die Implementierung

Da Cordova eins der meist genutzten Frameworks zur plattformunabhängigen App-Entwicklung darstellt, das kostenfrei, Open-Source und gut dokumentiert ist und zudem den Vorteil bietet, vorhandene Kenntnisse der Web-Entwicklung für die Erstellung von mobilen Apps nutzbar zu machen, wurde dieses hier für die Implementierung der in Abschnitt 4.2 beschriebenen Beispiel-Anwendung und zur näheren Beleuchtung ausgewählt. Weiterhin fand die auf jQuery aufbauende Oberflächen-Bibliothek für mobile Web-Oberflächen jQuery Mobile sowie die Data-Binding-Bibliothek Knockout für die Umsetzung der hybriden App hier Verwendung (Abschnitt 3.2 *Entwicklung von hybriden Apps*).

Wie in Unterabschnitt 3.2.2 beschrieben, stellt Knockout ein recht mächtiges und hilfreiches Mittel für die einfache und gut strukturierte Programmierung unter JavaScript dar. Der dadurch empfohlene und erleichterte Einsatz des MVVM-Patterns und die damit einhergehende klare und leicht wartbare Struktur bildet die Grundlage für die Architektur der hier beispielhaft implementierten Anwendung.

Grundlegend besteht die Anwendung aus den drei in Abbildung 3.1 abgebildeten Teilen *View* in Form einer HTML-Seite, dem JavaScript-Objekt *Model*, das den Zugriff auf das Gerät bewerkstellt und dessen Daten bereitstellt und dem *ViewModel*, welches die Eigenschaften und anzuseigenden Daten der View beinhaltet. Das JavaScript-Framework Knockout hält dabei durch das Data-Binding die Anzeige der Daten auf der View mit den Daten im ViewModel synchron (siehe Unterabschnitt 3.2.2).

Die Anbindung vom Model an das ViewModel wurde hier über das *Observer-Pattern* realisiert, mit dem eine lose Kopplung zwischen einer Komponente und deren *Observers* (engl. „*Beobachter*“) erzielt werden kann. Dabei registriert sich das ViewModel bei dessen Initialisierung beim Model per Aufruf der Methode `addEventListener(eventType, eventHandler)` (siehe Listing B.8) und

¹ Eine detailliertere Spezifikation der Anwendungsfälle für die geplante Beispiel-App findet sich in Anhang A auf Seite 58.

übergibt dabei als Parameter „`eventType`“ einen *String*, der den Typ des Events angibt, auf dessen Aussendung sich das ViewModel registriert sowie eine Funktion als `eventHandler`, welche ausgeführt werden soll, sobald das Event empfangen wird (siehe Listing B.9). Dazu hat das Model einen Verweis auf eine `ObserverMap`, welche die registrierten Event-Handler der Observer in Listen speichert, die dem jeweiligen Ereignistyp, der bei der Registrierung angegeben wird, zugeordnet werden (siehe Listing B.5).

Statt also bei erfolgreicher Ausführung einer Operation die Daten direkt per Methodenaufruf an das ViewModel zu übergeben, wird hier lediglich die Methode `notifyObservers()` der `ObserverMap` aufgerufen, wodurch die registrierten Observer über das Auftreten des Ereignisses benachrichtigt werden. Das Model muss dabei keinerlei Kenntnis vom ViewModel haben, lediglich im ViewModel ist bekannt, welche Methoden des Models aufgerufen werden können, bzw. welche Events dieses aussendet.

Durch diese lose Kopplung kann einerseits die Entwicklung erleichtert werden, da bei einer möglichen Aufteilung der Implementierung das Entwicklerteam, das für die Implementierung des Models zuständig ist, keine Kenntnis über die Struktur des ViewModels oder gar der View haben muss und darüber hinaus könnten auch weitere Observer auf Benachrichtigungen des Models registriert werden, beispielsweise, um weitere Aktionen nach Auftreten eines Events durchzuführen.

Neben diesen architektonischen Aspekten wurde hier das Observer-Pattern für die Kopplung zwischen Model und ViewModel gewählt, da bei Anforderung der Daten durch das ViewModel noch nicht bekannt ist, wann diese Daten genau zurückgeliefert werden, da das Cordova-Framework nach Aufruf einer Plugin-API-Methode² ausgeführt wird (siehe Listing 5.1). Diese erst vom Gerät laden muss und so das Zurückliefern der Daten per Rückgabewert zu einem `null`-Wert führen kann, da bei Aufruf der Methode `findContacts()` die Daten unter Umständen noch nicht verfügbar sind (siehe Listing 5.1).

² Beispielweise `navigator.contacts.find()` zum Anfordern der Kontaktdaten aus dem Adressbuch

Kapitel 5

Implementierung der Geräte-Schnittstelle

5.1 Zugriff auf die Kontaktverwaltung des Geräts

5.1.1 Funktionsweise

Im Anwendungsfall *Liste Teilen* der in Abschnitt 4.2 beschriebenen Beispielanwendung soll die Anforderung erfüllt werden, aus der Anwendung heraus auf das native Adressbuch des Geräts zuzugreifen, um die bestehenden Kontakte zu laden und anzuzeigen, sodass diese vom Nutzer ausgewählt werden, an die Anwendung übergeben und als neues Listenmitglied in ein Listenobjekt eingetragen werden können.

Nachdem das Cordova-Plugin *Contacts*, wie in Unterabschnitt 3.2.4 *Phongap / Cordova* beschrieben, zur Anwendung hinzugefügt wurde, lässt sich damit der Zugriff auf die native Kontaktverwaltung des jeweiligen Betriebssystems bewerkstelligen. Dieses bietet beispielsweise die Möglichkeit, nach bestimmten Kontakten im Adressbuch zu suchen, neue Kontakte zu erstellen und dem Adressbuch hinzuzufügen, sowie bestehende Kontakte zu entfernen oder zu duplizieren [13].

Um Kontakte im Adressbuch zu suchen, muss im Wesentlichen die Methode `navigator.contacts.find(fields, onSuccess, onError, options)` ausgeführt werden, wobei `fields` die zu durchsuchenden Datenfelder repräsentiert, `onSuccess` die Funktion angibt, die bei erfolgreicher Ausführung der `find`-Methode ausgeführt werden soll, `onError` den *Error-Handler* bei Auftreten eines Fehlers beim Suchen der Kontakte und `options` zusätzliche Optionen wie Suchfilter oder ein `multiple-Flag`, das angibt, ob mehrere Kontakte zurückgegeben werden sollen

(siehe Listing 5.1, Zeile 26). Die `find`-Methode wird *asynchron* ausgeführt, das heißt, dass der Aufrufer nicht wissen kann, wann das Ergebnis zurückgeliefert wird, also nicht als Rückgabewert zurückgegeben werden kann, sondern bei erfolgreicher Suche an den o. g. `onSuccess-Handler` übergeben wird und dort weiter verarbeitet werden kann [13].

Da in diesem Beispiel (Listing 5.1) keine bestimmten, sondern *alle* Kontakte angezeigt werden sollen, wird der `find`-Methode kein spezieller `filter` übergeben (Zeile 23). Ebenso ist aus dem selben Grund hier keine Einschränkung der durchsuchten Felder nötig, sodass durch der `fields`-Parameter den Wert `"*"` hat, und so alle Felder des `Contact`-Objekts durchsucht werden (Zeile 25). Bei erfolgreicher Ausführung der `find`-Methode werden die zurückgegebenen Kontaktdaten als Parameter in Form eines *Arrays*, das die entsprechenden JavaScript-Objekte vom Typ `Contact` beinhaltet, an die `onSuccess`-Methode übergeben und vom Model an dessen Observer (in diesem Fall das ViewModel) versendet (Zeile 13).

Der Bezeichner `events.FOUND_CONTACTS` stellt eine String-Konstante dar, die in einem globalen Konstanten-Objekt definiert wurde und den Typ des Events identifiziert (siehe Listing B.6).

Listing 5.1: Verwendung des *Contacts*-Plugins von Cordova.

```
1 function findContacts() {
2
3     if (!model.deviceReady) {
4         var errorMsg = 'Gerät ist nicht bereit!';
5         console.error(errorMsg);
6         alert(errorMsg);
7         return;
8     }
9
10    var onSuccess = function(contacts) {
11        var successMsg = contacts.length + ' Kontakte gefunden.';
12        console.log(successMsg);
13        model.observerMap.notifyObservers(events.FOUND_CONTACTS, contacts);
14    };
15
16    var onError = function(contactError) {
17        var errorMsg = 'Fehler beim Laden der Kontakte!';
18        console.error(errorMsg);
19        alert(errorMsg);
20    };
21
22    var options = new ContactFindOptions();
23    options.filter = "";
24    options.multiple = true;
25    var fields = [ "*" ];
26    navigator.contacts.find(fields, onSuccess, onError, options);
27
28 }
```

Die in Listing 5.1 per Observer-Pattern versendeten Daten werden in der entsprechenden Handler-Methode an die `contacts`-Eigenschaft des ViewModels übergeben (Listing 5.2, Zeilen 16 - 18), deren Elemente (also die Kontakt-Objekte) anschließend durch das Data-Binding in der Oberfläche angezeigt werden (siehe unten).

Listing 5.2: Wesentlicher Ausschnitt des ViewModels.

```
1 function ErrandsView(lists) {
2     var self = this;
3
4     self.lists = ko.observableArray(lists);
5     self.newListName = ko.observable("");
6     self.contacts = ko.observableArray([ util.getDummyContact() ]);
7
8     // [...] Der Übersichtlichkeit halber gekürzt.
9
10    self.getContacts = function(event, ui) {
11        console.debug("Fordere Kontaktdateien an...");
12        model.findContacts();
13    };
14
15    self.bindEvents = function() {
16        model.addEventListener(events.FOUND_CONTACTS, function(contacts) {
17            self.contacts(contacts);
18        });
19    };
20
21    console.log("Init View...");
22    self.bindEvents();
23 }
```

Die Methode `findContacts()`, die den Aufruf zum Laden der Kontakte an das Model delegiert, wird hier per Event-Binding an die Kontakt-Komponente der Oberfläche gebunden, sodass diese jedes mal aufgerufen wird, wenn die Kontaktliste auf- oder zugeklappt wird, um die Daten aus dem Model anzufordern (siehe Listing 5.3, Zeile 3). Durch die Bindung der Eigenschaft `contacts` in Form eines *Observable Arrays* des ViewModels (Listing 5.2, Zeile 6) an die *Listview* der Oberfläche werden in dieser Liste alle Kontakte des Adressbuchs angezeigt (siehe Listing 5.3, Zeile 7).

Listing 5.3: UI-Komponente zur Darstellung der Kontaktliste und Auswahl einzelner Kontakte.

```

1 <div data-role="collapsible" data-expanded-icon="carat-d"
      data-collapsed-icon="plus" data-theme="b"
2   data-bind="jqmRefreshList: $root.contacts,
3   event: { 'collapsibleexpand': $root.getContacts }">
4   <h4>Mitglieder hinzufügen</h4>
5   <ul data-role="listview" data-theme="b"
6     data-bind="foreach: $root.contacts,
7       jqmRefreshList: $root.contacts">
8     <li>
9       <a href="#" data-bind="text: name.formatted,
10          click: $parent.addMember,
11          css: $parent.memberStatus($data),
12          style: {
13            'background-color':
14              $parent.isMember($data) ? '#aed6f' : '#333'
15            }">
16       </a>
17     </li>
18   </ul>
19 </div>
20

```

5.1.2 Explorationsergebnisse

Einige Unterschiede ergeben sich bei der Verwendung des *Contacts*-Plugins zwischen den verschiedenen Zielplattformen. So können beispielsweise nicht alle Methoden der Contacts API auf allen mobilen Plattformen gleichermaßen ausgeführt werden. Die hier verwendete `find()`-Methode bietet mit der Unterstützung für Android, Blackberry 10, *FoxOS*, iOS, Windows Phone und *Microsoft Windows 8* eine relativ breite Kompatibilität für alle größeren gängigen und mobilen Betriebssysteme. Allerdings auch werden einige Datentypen *nicht* von allen Plattformen unterstützt, so zum Beispiel das `ContactOrganization`-Objekt¹, das lediglich bei der Entwicklung für Android, Blackberry 10, Firefox OS, iOS und Windows Phone zur Verfügung steht, damit aber immer noch alle großen Plattformen unterstützt [13].

Doch auch bei den unterstützten Plattformen sind einige Besonderheiten zu beachten, wenn es um die Verwendung einzelner Attribute der verschiedenen Objekttypen geht. So ist beispielsweise das Feld `displayName` des `Contact`-Objekts unter iOS nicht verfügbar, sodass in diesem Fall auf das `name-` oder `nickname`-Feld zurückgegriffen werden muss. Ähnliches gilt auch für weitere Ob-

¹ siehe Unterunterabschnitt 3.2.4.1 *Grundlegendes*, Seite 20.



(a) Die installierte App in der Android-Apps-Übersicht.

(b) Die installierte App in der iOS-Apps-Übersicht.

Abbildung 5.1: Nach der Ausführung des `cordova emulate`-Befehls wird die zu testende Cordova-App in der Übersicht des Betriebssystems im jeweiligen Emulator angezeigt.

Quelle: Eigener Screenshot.

jekte oder deren Methoden und Felder, die unter bestimmten Plattformen nicht oder nur teilweise unter Berücksichtigung bestimmter Besonderheiten funktionieren [13].

So wird beispielsweise `find`-Methode grundsätzlich mit Android, BlackBerry 10, Firefox OS, iOS, Windows Phone und Windows 8 von den wichtigsten mobilen Plattformen unterstützt, doch bei letzterer besteht die Einschränkung, dass aus der hybriden App heraus nicht ohne weiteres auf die Kontaktdateien zugegriffen werden kann, wie es bei den anderen Plattformen der Fall ist (siehe oben), sondern eine Nutzerinteraktion nötig ist, um Kontakte aus dem Adressbuch auszuwählen und weiter zu verarbeiten, sodass bei Anfrage

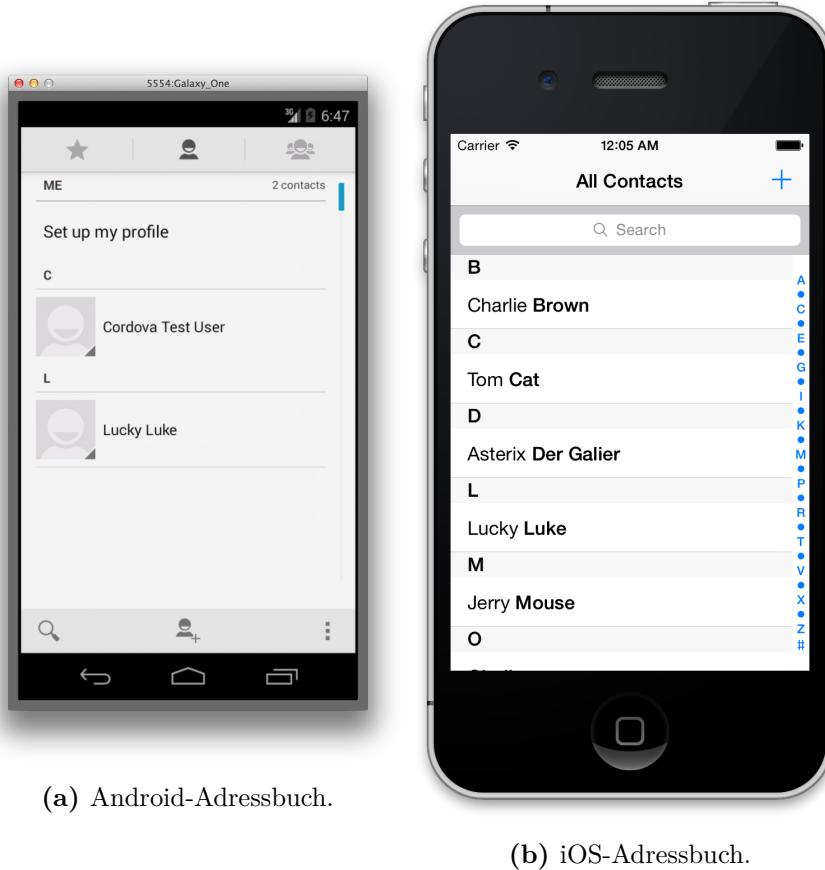


Abbildung 5.2: Anzeige des nativen Adressbuchs mit Beispiel-Kontakten.

Quelle: Eigener Screenshot.

von Kontaktdaten die App für das native Windows 8-Adressbuch geöffnet wird und der Nutzer die gewünschten Kontakte selbst von Hand auswählen muss. Darüber hinaus bieten Windows 8-Kontakte lediglich einen Lesezugriff, sodass hier aus der Kontaktdatenbank geladene Objekte nicht verändert, gelöscht oder dupliziert werden können [13].

Eine Kategorisierung von Kontakten über das `categories`-Attribut wird bei Firefox OS sowie iOS nicht unterstützt, sodass beim Versuch, diesen Wert anzufragen, `null` zurückgegeben wird. In Bezug auf die Unterstützung des `Contact`-Objekts ergeben sich die meisten Einschränkungen für die Plattform Windows Phone. Hier finden sich in der Contacts API-Dokumentation beispielsweise Informationen über ein teilweise leicht inkonsistentes Verhalten beim Erstellen bzw. Suchen von Kontakten. So unterscheidet sich beispielsweise der Wert des `displayName`-Attributs, das bei Erstellung eines `Contact`-Objekts angegeben wird

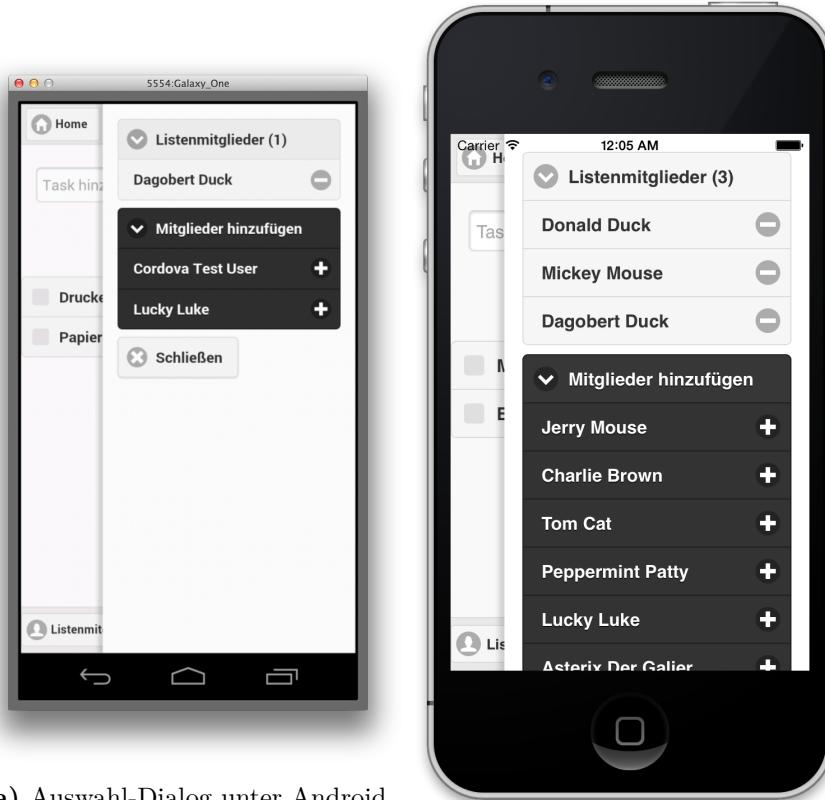


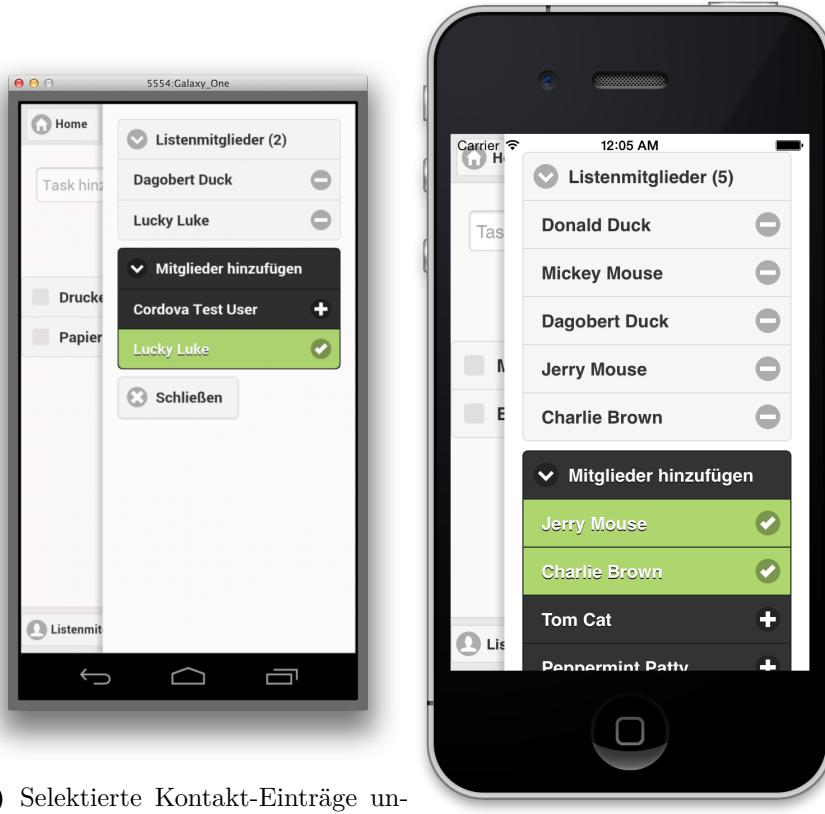
Abbildung 5.3: Kontakt-Auswahl-Dialog für das Hinzufügen von Kontakten aus dem Adressbuch. Die Anwendung lädt die Kontakte aus dem Adressbuch des Geräts in die Auswahl-Box der Anwendung (siehe Abbildung 5.2).

Quelle: Eigener Screenshot.

von dem, das bei der Suche zurückgeliefert wird. Ebenso können bei der Erstellung eines `Contact`-Objekts mehrere *URLs* angegeben werden, während bei den Objekten des Suchergebnisses nach Ausführung der `find`-Methode lediglich *eine* verfügbar ist.

Einige Listenfelder des `Contact`-Objekts besitzen ein Attribut mit dem Namen `pref`, dem ein boolescher Wert zugewiesen kann, der angibt, ob das jeweilige Objekt innerhalb einer Liste² das vom Nutzer präferierte darstellt. Dieses `pref`-Attribut wird ebenfalls unter einigen Plattformen nicht unterstützt, so z. B. für `phoneNumbers` und `emails` unter Windows Phone sowie beim `addresses`-

² So beispielsweise bei den Feldern `emails`, `phoneNumbers` oder `addresses`, denen jeweils mehrere E-Mail-Adressen, Telefonnummern bzw. Adressen zugewiesen werden können.



(a) Selektierte Kontakt-Einträge unter Android.

(b) Selektierte Kontakt-Einträge unter iOS.

Abbildung 5.4: Übergabe der Kontakte an die Anwendung: Wird ein Kontakt ausgewählt, wird dieser dem Observable-Array `contacts` des ViewModels (siehe Listing 5.2) hinzugefügt und als *ausgewählt* in der Oberfläche markiert.

Quelle: Eigener Screenshot.

Feld unter Android 2.X, Blackberry 10, iOS und Windows 8. Die `Contact`-Felder `note`, `ims`, `birthdays` und `categories` werden für Windows Phone gar nicht unterstützt.

Das o. g. `Contact`-Feld `name` hat den Datentyp `ContactName` besteht und besteht aus den weiteren String-Attributen `formatted`, `familyName`, `code`, `middleName`, `honorablePrefix` und `code`. Grundlegend wird dieses Objekt zwar von allen elementaren Plattformen unterstützt, doch auch hier gibt es einige Einschränkungen. Das `formatted`-Attribut wird unter Android, Blackberry 10, Firefox OS und

iOS nur teilweise unterstützt und bietet für Android, iOS sowie Firefox OS lediglich einen Lesezugriff. Unter Windows 8 stellt dieses das einzige Attribut des `ContactName`-Objekts dar, alle anderen werden nicht unterstützt.

Während bei der nativen App-Entwicklung für iOS oder Android UI-Komponenten für die Arbeit mit Kontakten bereitstehen, liefert die Contacts API hier lediglich Mechanismen für den Zugriff auf die Kontaktdata, jedoch nicht die entsprechenden UI-Komponenten, da diese vom jeweiligen verwendeten GUI-Toolkit abhängen. Die hier verwendete Oberflächen-Bibliothek jQuery-Mobile beinhaltet lediglich allgemeine *Widgets* wie Listen, Buttons, Tabellen etc., sodass die Erstellung einer UI-Komponente für die Auswahl von Kontakten dem Entwickler überlassen bleibt.

In Verbindung mit der Data-Binding-Bibliothek Knockout kann eine solche Komponente jedoch relativ einfach erstellt werden, indem die Felder in der Oberfläche an Eigenschaften des ViewModels gebunden werden, welches durch Benachrichtigungen des Models, das den Zugriff auf die native Ebene des Betriebssystems abwickelt, die Daten der Anwendung und des Geräts anzeigen kann (siehe Unterabschnitt 3.2.2).

Grundsätzlich bietet die Contacts API trotz einiger Einschränkungen (siehe oben) ein nützliches Werkzeug mit einem relativ breiten Funktionsumfang für den grundlegenden Zugriff auf die Kontaktverwaltung der meisten großen mobilen Betriebssysteme. Der für dieses Feature definierte Anwendungsfall konnte mit den Mitteln der Cordova-API und weiteren Technologien der Webentwicklung wie Knockout oder jQuery Mobile gut für die hier verwendeten Plattformen iOS und Android umgesetzt werden. Trotz des grundsätzlich plattformunabhängigen Ansatzes sollte allerdings bei der Entwicklung mithilfe dieser API die Unterstützung der verwendeten Features und Attribute und deren Besonderheiten für die angestrebten Zielplattformen in der Contacts API-Dokumentation überprüft werden. Somit ist eine Kenntnis über die angestrebten Zielplattformen für die Verwendung eines solchen Plugins beinahe unabdingbar und die Entwicklung in gewisser Weise wieder ein Stück weit abhängiger von den verwendeten Plattformen.

Kapitel 6

Fazit

Resümee / Ausblick

Resümee

Ausblick

Anhang

Anhang A

Anwendungsfälle der Beispiel-Anwendung

1

A.1 Diagramme

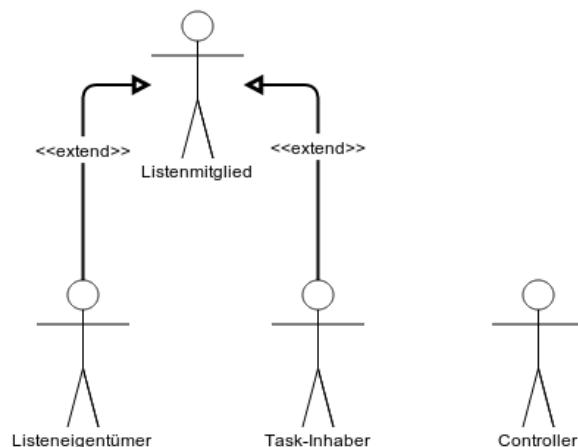


Abbildung A.1: Beziehung zwischen den Akteuren für die Spezifikation der Beispiel-Anwendung (siehe Abbildung A.2).

Quelle: Eigene Grafik.

¹ Die hier beschriebenen Anwendungsfälle und deren Spezifikation sind aus einer frühen Konzeptionsphase eines App-Prototypen, der primär für die Exploration eines Frameworks zur plattformunabhängigen App-Entwicklung und weniger für die tatsächliche Funktionstüchtigkeit konzipiert ist (siehe Abschnitt 4.2 *Beispiel-Anwendung*).

Daher deckt sich diese Beschreibung und der Umfang der beschriebenen Funktionalität zum großen Teil nicht mit dem Zustand der in Teil II beschriebenen Implementierung.

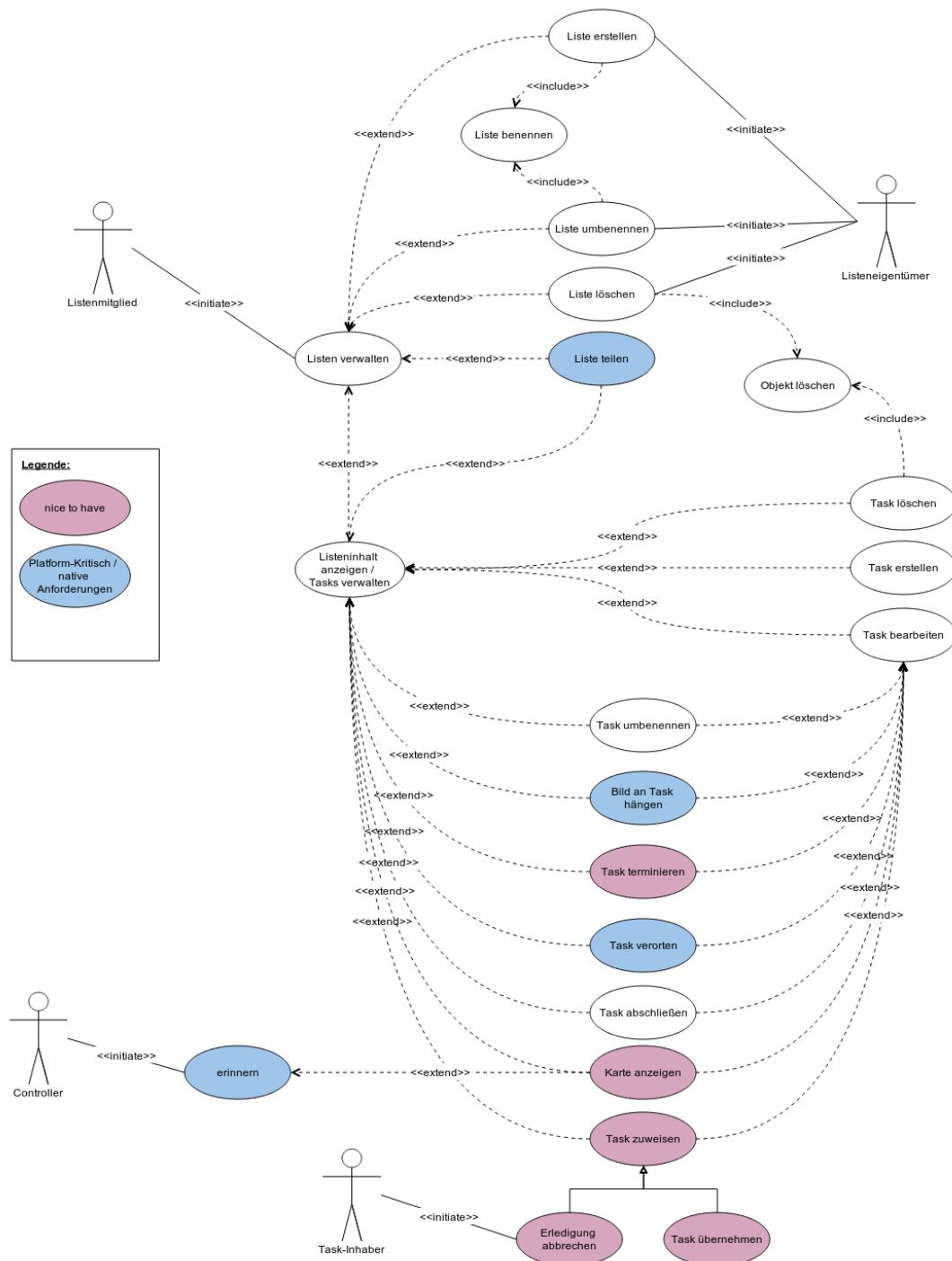


Abbildung A.2: Anwendungsfalldiagramm der Beispielanwendung.

Quelle: Eigene Grafik.

A.2 Anwendungsfallbeschreibung

Listen verwalten

Akteure

initiiert vom Listenmitglied

Anfangsbedingung

Es existieren Listen für den angemeldeten Benutzer. (Das System legt für jeden Benutzer mindestens eine Standardliste an.)

Beschreibung

1. Das System zeigt eine Übersicht der gespeicherten Listen, derer der aktive Benutzer Mitglied ist.
2. Für alle Listen hat er hier die Möglichkeit,
 - (a) sich den Listeninhalt anzeigen zu lassen
(extend *Tasks verwalten / Listeninhalt anzeigen*) und
 - (b) Listen zu teilen (extend *Liste teilen*).
3. Für alle Listen, deren er darüber hinaus Eigentümer ist, hat er die Möglichkeit, diese
 - (a) zu löschen (extend *Liste löschen*),
 - (b) umzubenennen (extend *Liste umbenennen*) und
 - (c) zu erstellen (extend *Liste erstellen*).
4. Das System prüft den verfügbaren Bildschirmspace und initiiert, sofern der Platz dafür ausreicht, automatisch den Anwendungsfall *Tasks verwalten / Listeninhalt anzeigen* anzeigen für die erste Standardliste.

Liste erstellen

Akteure

Initiiert vom Listeneigentümer

Beschreibung

1. Der Nutzer signalisiert dem System, dass er eine neue Liste erstellen möchte.
2. Weiter mit *Liste benennen*.

Liste umbenennen

Akteure

Initiiert vom Listeneigentümer

Beschreibung

1. Der Nutzer signalisiert dem System, dass er eine Liste umbenennen möchte.
2. Weiter mit *Liste benennen*.

Liste benennen**Akteure**

Initiiert vom Listeneigentümer

Beschreibung

1. Das System zeigt ein Eingabefeld, in dem der neue Listenname eingegeben werden kann.
2. Der Nutzer gibt den Namen für die neue Liste ein und bestätigt die Eingabe.
3. Das System speichert die Liste mit dem eingegebenen Namen ab (je nach Kontext eine neu angelegte oder die alte mit neuem Namen).
4. Die Ansicht wechselt wieder zur Listenansicht, in der jetzt auch die neu angelegte Liste erscheint.

Liste löschen**Akteure**

Initiiert vom Listeneigentümer

Beschreibung

1. Der Nutzer signalisiert dem System, dass er eine Liste löschen möchte.
2. Weiter mit *Objekt löschen*.

Objekt löschen

Teil-Anwendungsfall.

Beschreibung

1. Das System öffnet einen Dialog, in dem der Nutzer gefragt wird, ob er sicher ist, dass er das ausgewählte Objekt löschen möchte.
2. Der Nutzer bestätigt den Dialog zum Löschen.
3. Das System löscht das gewählte Objekt aus dem System und wechselt wieder zur vorherigen Ansicht.

Alternative Verläufe

2. Der Nutzer bricht den Vorgang ab.

Liste teilen

Akteure

Initiiert vom Listenmitglied

Beschreibung

1. Der Nutzer signalisiert dem System, dass er andere Nutzer zu einer Liste einladen möchte.
2. Das System wechselt zum Adressbuch des Betriebssystems.
3. Der Nutzer wählt einen oder mehrere Kontakte aus seinem Adressbuch aus und bestätigt die Eingabe mit „Fertig“.
4. Das System wechselt wieder zur Besorgungen-App und zeigt einen Dialog, in dem die ausgewählten Kontakte noch einmal aufgelistet werden.
5. Der Nutzer bestätigt die Eingabe mit „Liste teilen“.
6. Das System speichert die Information der neuen Listenmitglieder ab.
7. Das System versendet Benachrichtigungen an die eingeladenen Nutzer, dass eine Liste mit ihnen geteilt wurde.

Tasks verwalten / Listeninhalt anzeigen

Akteure

initiiert vom Listenmitglied

Beschreibung

1. Der Nutzer signalisiert dem System mit der Auswahl einer Liste, dass er deren Inhalt anzeigen oder verwalten möchte.
2. Das System zeigt neben den vorher schon angezeigten Listen nun auch den Inhalt der selektierten Liste, also die darin enthaltenen Tasks an.
3. Von hier aus hat der Nutzer folgende Interaktionsmöglichkeiten (extend):
 - (a) *Liste teilen*
 - (b) *Task erstellen*
 - (c) *Task zuweisen*
 - (d) *Task bearbeiten*

Außerdem werden dem Nutzer je nach verfügbarem Bildschirmplatz auch für alle Tasks Shortcuts zu allen Extend-Anwendungsfälle von *Task bearbeiten* angezeigt.

Task löschen

Akteure

Initiiert vom Listenmitglied

Beschreibung

1. Der Nutzer signalisiert dem System, dass er einen Task löschen möchte.
2. Weiter mit *Objekt löschen*.

Task erstellen**Akteure**

Initiiert vom Listenmitglied

Beschreibung

1. Der Nutzer signalisiert dem System, dass er einen Task erstellen möchte.
2. Das System öffnet ein Eingabefeld, in das der Name für den neuen Task eingetragen werden kann.
3. Der Nutzer trägt den Namen für den Task ein und bestätigt die Eingabe.
4. Das System speichert den neuen Task in die ausgewählte Liste ab.

Task umbenennen**Akteure**

Initiiert vom Listenmitglied

Beschreibung

1. Der Nutzer signalisiert dem System, dass er einen Task umbenennen möchte.
2. Das System öffnet ein Eingabefeld, in das der neue Name für den Task eingetragen werden kann.
3. Der Nutzer trägt den Namen für den Task ein und bestätigt die Eingabe.
4. Das System ändert den Namen des Tasks und speichert ihn ab.

Task bearbeiten**Akteure**

initiiert vom Listenmitglied

Beschreibung

1. Der Nutzer signalisiert dem System, dass er einen Task bearbeiten möchte.
2. Das System öffnet den Bearbeiten-Dialog für den ausgewählten Task.
Der Nutzer hat folgende Interaktionsmöglichkeiten:
 - (a) *Bild an Task hängen*
 - (b) *Task terminieren*

- (c) *Task verorten*
- (d) *Task zuweisen*
- (e) *Task abschließen*
- (f) *Erledigung abbrechen (Task zurückgeben)*
- (g) *Task umbenennen*

Bild an Task hängen

Akteure

Initiiert vom Listenmitglied

Beschreibung

1. Der Nutzer signalisiert dem System, dass er ein Bild an einen Task hängen möchte.
2. (In der Prototyp-Version deaktiviert) Das System öffnet einen Dialog, in dem gefragt wird, ob ein Bild mit der Kamera aufgenommen oder ein Bild aus der Galerie ausgewählt werden soll.
3. Der Nutzer wählt die Option „Von der Kamera“.
4. Das System wechselt zur Kamera-Anwendung des Betriebssystems.
5. Der Nutzer nimmt ein Foto auf und bestätigt die Eingabe mit „Fertig“.
6. Das System wechselt wieder zur Besorgungen-App und zeigt in einem Dialog das aufgenommene Foto.
7. Der Nutzer bestätigt die Eingabe mit „Anhängen“.
8. Das System speichert das Foto zum ausgewählten Task ab.

Task terminieren

Akteure

Initiiert vom Listenmitglied

Beschreibung

1. Der Nutzer signalisiert dem System, dass ein Task an einem bestimmten Datum erledigt werden soll.
2. Das System zeigt ein Datumsauswahlfeld an.
3. Der Nutzer gibt das entsprechende Datum in das Feld ein.
4. Das System fragt in einem Dialog, ob der Task auch in den Kalender des Betriebssystems eingetragen werden soll.
5. Der Nutzer bestätigt den Dialog.

6. Das System speichert den Task als ganztägigen Termin in den Kalender des Betriebssystems ab und gibt eine entsprechende Meldung über das erfolgreiche Speichern zurück.

Task verorten

Akteure

Initiiert vom Listenmitglied

Beschreibung

1. Der Nutzer signalisiert dem System, dass ein Task an einem bestimmten Ort erledigt werden soll.
2. Das System zeigt ein Eingabefeld, um den Ort einzugeben.
3. Der Nutzer trägt den Ort in das Eingabefeld ein (entweder gleich als Freitextsuche oder vorerst nur als reine GPS-Koordinaten).
4. Das System speichert den angegebenen Ort zum betreffenden Task ab.

Task zuweisen

Akteure

Initiiert vom Listenmitglied

Beschreibung

1. Der Nutzer signalisiert dem System, dass er einen Benutzer für einen Task zuständig zuweisen möchte.
2. Das System zeigt eine Liste der Listenmitglieder der aktuellen Besorgungsliste.
3. Der Nutzer wählt einen Nutzer aus dieser Liste aus.
4. Das System speichert den ausgewählten Nutzer als neuen Task-Inhaber ab und zeigt mit einem kleinen Icon am Task an, wem dieser Task zugewiesen ist.

Erledigung abbrechen (Task zurückgeben)

Akteure

Initiiert vom Task-Inhaber

Anfangsbedingung

Der Task wurde einem Nutzer zugewiesen, der dadurch zum Task-Inhaber wird.

Beschreibung

1. Verlauf wie bei *Task zuweisen*.

2. Statt 3.: Der Nutzer wählt aus der Liste den Default-Wert „(kein)“ aus.

Task übernehmen

Akteure

Initiiert vom Listenmitglied

Beschreibung

1. Verlauf wie bei *Task zuweisen*.
2. Statt 3.: Der Nutzer wählt aus der Liste sich selbst aus.

Task abschließen

Akteure

Initiiert vom Listenmitglied

Beschreibung

1. Der Nutzer signalisiert dem System, dass ein Task erledigt ist.
2. Das System speichert den Task als erledigt ab und stellt dieses visuell dar (Bspw. durchgestrichen, auf Erledigt-Liste verschoben).

Erinnern

Akteure

Initiiert vom Controller; Listenmitglied ist beteiligt.

Beschreibung

1. Das System erkennt durch den permanenten GPS-Abgleich der zugeordneten Listenmitglieder mit den verorteten Tasks, dass sich ein Nutzer gerade in der Nähe des Ortes eines Tasks befindet.
2. Das System sendet eine Benachrichtigung an den nativen Benachrichtigungsmechanismus des Betriebssystems, aus der hervorgeht, dass der Nutzer sich gerade in der Nähe eines Ortes eines zu erledigenden Tasks befindet.

Karte anzeigen

Akteure

Initiiert vom Listenmitglied

Anfangsbedingung

Es wurde ein Ort für den betreffenden Task hinterlegt (*Task verorten*).

Beschreibung

1. Der Nutzer signalisiert dem System, dass er den Ort eines bestimmten Tasks auf der Karte ansehen möchte.
2. Das System wechselt zur nativen Karteanwendung des Betriebssystems und zeigt (neben dem aktuellen Standort) den Ort des Tasks mit einer Markierung auf der Karte an.

Anhang B

Quellcode

Listing B.1: HTML-Oberflächenbeschreibung der hier implementierten Beispiel-Anwendung.

```
1 <!DOCTYPE html>
2 <!--
3 Licensed to the Apache Software Foundation (ASF) under one
4 or more contributor license agreements. See the NOTICE file
5 distributed with this work for additional information
6 regarding copyright ownership. The ASF licenses this file
7 to you under the Apache License, Version 2.0 (the
8 "License"); you may not use this file except in compliance
9 with the License. You may obtain a copy of the License at
10
11 http://www.apache.org/licenses/LICENSE-2.0
12
13 Unless required by applicable law or agreed to in writing,
14 software distributed under the License is distributed on an
15 "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
16 KIND, either express or implied. See the License for the
17 specific language governing permissions and limitations
18 under the License.
19 -->
20 <html>
21   <head>
22     <meta charset="utf-8" />
23     <meta name="format-detection" content="telephone=no" />
24     <!-- WARNING: for iOS 7, remove the width=device-width and height=device-height
25         attributes. See https://issues.apache.org/jira/browse/CB-4323 -->
26     <meta name="viewport" content="user-scalable=no, initial-scale=1, maximum-scale=1
27       , minimum-scale=1, target-densitydpi=device-dpi" />
28
29     <link rel="stylesheet" href="lib/jquery.mobile/1.4.2/jquery.mobile-1.4.2.css" />
30     <link rel="stylesheet" href="css/index.css" />
31     <link rel="stylesheet" href="css/style.css" />
32     <script type="text/javascript" src="cordova.js"></script>
33
34     <!-- Reihenfolge der Scripts ist relevant! -->
35     <script src="lib/jquery/jquery-2.1.1.min.js"></script><!-- Kein Support für IE <
36         9 TODO: Kompatibilität mit WebViews checken! -->
37     <script src="lib/jquery.mobile/1.4.2/jquery.mobile-1.4.2.js"></script>
38     <script src="lib/knockout/knockout-3.1.0.debug.js"></script>
39
40     <!-- Weinre Debugger: -->
```

```
38 <script src="http://localhost:8080/target/target-script-min.js#anonymous">
39   </script>
40
40   <title>Besorgungen</title>
41 </head>
42 <body>
43
44   <!--
45     MAIN PAGE: LISTS
46   -->
47 <div data-role="page" id="home">
48
49   <header data-role="header">
50     <h1 data-bind="text: heading"></h1>
51   </header>
52
53   <!--
54     UI-CONTENT
55   -->
56 <div data-role="main" class="ui-content">
57   <div class="insert-new">
58     <form data-bind="submit: addList">
59       <input data-bind="value: newListName, valueUpdate: 'afterkeydown'" type="text" placeholder="Liste hinzufügen" />
60       <button data-role="button" data-bind="enable: newListName().length>0" type="submit" class="add-button"> Neue Liste </button>
61     </form>
62   </div>
63   <br />
64   <ul data-role="listview" id="lists" data-bind="foreach: lists,
jqmRefreshList: lists">
65     <li data-bind="visible: $root.lists().length > 0">
66       <a href="#" data-transition="slide" data-bind="html: displayName, attr: {
href: page, class : allDone() ? 'all-done ui-btn ui-btn-icon-right
ui-icon-carat-r' : 'ui-btn ui-btn-icon-right ui-icon-carat-r'}, style: { color:
allDone() ? 'gray' : 'inherit' }"></a>
67     </li>
68   </ul>
69   <br />
70 </div>
71   <!-- END UI-CONTENT -->
72
73 <footer data-role="footer" data-position="fixed">
74   <!-- Nur Test-Buttons für Speicher-Funktion / Funktionstüchtigkeit noch nicht
geklärt. -->
75   <!--      <button data-bind="click: store" class="ui-btn ui-btn-inline">
76     Save
77   </button>
78   <button data-bind="click: restore" class="ui-btn ui-btn-inline">
79     Restore
80   </button> -->
81 </footer>
82
83 </div>
84
85   <!--
86     SUBPAGE: TASKS
87   -->
88   <!-- ko foreach: lists -->
89 <div data-role="page" data-bind="attr: { id: id } ">
```

```

91      <header data-role="header">
92          <a href="#home" data-role="button" class="ui-btn ui-btn-left ui-icon-home
93              ui-btn-icon-left">Home</a>
94          <h1 data-bind="text: displayName"></h1>
95      </header>
96
97      <!-- UI-CONTENT -->
98      <div data-role="main" class="ui-content">
99          <div class="insert-new">
100             <form data-bind="submit: addTask">
101                 <input data-bind="value: newTaskName, valueUpdate: 'afterkeydown'" type="text"
102                     placeholder="Task hinzufügen" />
103                 <button data-bind="enable: newTaskName().length > 0" data-role="button"
104                     type="submit" class="add-button"> Task Hinzufügen </button>
105             </form>
106             <br />
107             <fieldset data-role="controlgroup">
108                 <div data-bind="foreach: tasks, jqmRefreshList: tasks">
109                     <label data-bind="attr: { for: id }, text: name, style: { textDecoration:
110                         done() ? 'line-through' : 'none', color: done() ? 'gray' : 'inherit' }"></label>
111                     <input type="checkbox" name="task" data-bind="value: id, attr: { id: id
112                         }, checked: done" />
113                 </div>
114             </fieldset>
115         </div>
116         <!-- END UI-CONTENT -->
117
118         <!--
119             CONTACTS PANEL
120             -----
121             <div data-role="panel" data-bind="attr: { id: contactsPanelID() }"
122                 data-position="right" data-display="overlay" data-dismissible="true" class="
123                     overlayPanel">
124             <div class="ui-panel-inner">
125                 <div class="panel-content">
126                     <div data-role="collapsible" data-expanded-icon="carat-d"
127                         data-collapsed-icon="carat-r" data-bind="jqmRefreshList: members">
128                         <h4><span data-bind="text: membersHeading"></span></h4>
129                         <ul data-role="listview" data-bind="foreach: members, jqmRefreshList:
130                             members">
131                             <li>
132                                 <a href="#" class="ui-icon-minus" data-bind="text: name.formatted,
133                                     click: $parent.removeMember"></a>
134                             </li>
135                         </ul>
136                     </div>
137                     <div style="height: 8px"></div><!-- Unschön, aber da der Platz sonst nach
138                         automatischem Refresh wieder verloren geht. -->
139                     <div data-role="collapsible" data-expanded-icon="carat-d"
140                         data-collapsed-icon="plus" data-theme="b" data-bind="jqmRefreshList:
141                             $root.contacts, event: { 'collapsibleexpand': $root.findContacts }">
142                         <h4>Mitglieder hinzufügen</h4>
143                         <ul data-role="listview" data-bind="foreach: $root.contacts,
144                             jqmRefreshList: $root.contacts, visible: $root.contactsNotEmpty" data-theme="b">
145                             <li>
146                                 <a href="#" data-bind="text: name.formatted, click:
147                                     $parent.addMember, css: $parent.memberStatus($data), style: { 'background-color':
148                                         $parent.isMember($data) ? '#aed66f' : '#333' }"></a>
149                             </li>
150                         </ul>
151                     </div>
152                 </div>
153             </div>
154         </div>
155     </div>
156 
```

```

136         </div>
137         <a href="#" data-rel="close" data-role="button" class="close-button"
138             data-bind="click: $root.reset">Schließen</a>
139         </div>
140     </div>
141     <!-- END CONTACTS PANEL -->
142
143     <footer data-role="footer" data-position="fixed">
144         <a data-bind="attr: { href: contactsPanelRef() }, text: membersHeading"
145             data-role="button" class="ui-btn ui-icon-user ui-btn-icon-left"></a>
146     </footer>
147 </div>
148 <!-- /ko -->
149
150 <!-- END SUBPAGE: TASKS -->
151
152 <!--
153 SCRIPTS
154
155 <!-- TYPES ----->
156 <script src="js/types/Contact.js"></script>
157 <script src="js/types/ContactAddress.js"></script>
158 <script src="js/types/ContactError.js"></script>
159 <script src="js/types/ContactField.js"></script>
160 <script src="js/types/ContactFindOptions.js"></script>
161 <script src="js/types/ContactName.js"></script>
162 <script src="js/types/ContactOrganization.js"></script>
163 <script src="js/types>List.js"></script>
164 <script src="js/types/Task.js"></script>
165
166 <!-- BASIC SCRIPTS ----->
167 <script src="js/ObserverMap.js"></script>
168 <script src="js/consts.js"></script>
169 <script src="js/util.js"></script>
170
171 <!-- MODEL -->
172 <script src="js/model/storage.js"></script>
173 <script src="js/model/model.js"></script>
174
175 <!-- VIEW MODEL -->
176 <script src="js/viewModel/ViewModel.js"></script>
177 <script src="js/viewModel/bindingHandlers.js"></script>
178
179 <!-- APP SCRIPTS ----->
180 <script src="js/index.js"></script>
181 <script>
182     app.initialize();
183 </script>
184 </body>
185 </html>

```

Listing B.2: style.css.

```

1 .insert-new {
2     text-align: right;
3 }
4 .add-button {
5     margin-right : 0em;
6 }

```

```

7 .all-done {
8   background-image : url("../img/check-gray.svg");
9   background-repeat : no-repeat;
10  background-size: 18px 18px;
11  background-position: 1em 50%;
12  padding-left: 40px;
13 }

```

Listing B.3: List.js.

```

1 ****
2 * List Object
3 ****
4 /*
5 * TODO Eher in extra js-Datei auslagern.
6 */
7 function List(id, name, members, tasks) {
8 /*
9 * =====
10 * Properties
11 * =====
12 */
13 this.id = id;
14 this.name = ko.observable(name);
15 this.members = ko.observableArray(members);
16 this.tasks = ko.observableArray(tasks);
17 this.page = "#" + id;
18 this.listviewID = id + consts.LISTVIEW;
19
20 this.membersHeading = ko.computed(function() {
21   return consts.LIST_MEMBERS + " (" + this.members().length + ")";
22 }, this);
23
24 this.countOpenTasks = ko.computed(function() {
25   var count = 0;
26   this.tasks().forEach(function(task) {
27     if (!task.done()) {
28       count++;
29     }
30   });
31   return count;
32 }, this);
33
34 this.allDone = ko.computed(function() {
35   return (this.countOpenTasks() == 0) && (this.tasks().length > 0);
36 }, this);
37
38 this.displayName = ko.computed(function() {
39   var countDisplay = " (" + this.countOpenTasks() + ")";
40
41   if (this.countOpenTasks() > 0) {
42     displayName = this.name() + countDisplay;
43   } else {
44     displayName = this.name();
45   }
46   return displayName;
47 }, this);
48
49 this.memberStatus = function(contact) {
50   return this.isMember(contact) ? 'ui-icon-check' : 'ui-icon-plus';
51 };

```

```

52
53     this.contactsPanelID = ko.computed( function() {
54         return consts.CONTACTS_PANEL + this.id;
55     }, this);
56
57     this.contactsPanelRef = ko.computed( function() {
58         return "#" + this.contactsPanelID();
59     }, this);
60
61     /*
62      * =====
63      * Operations
64      * =====
65      */
66     // Add New Task:
67     // TODO vielleicht eher in extra objekt auslagern?
68     this.newTaskName = ko.observable("");
69     this.addTask = function() {
70         var newTaskID = util.createUID(consts.TASK);
71
72         var newTask = new Task(newTaskID, this.newTaskName(), false);
73
74         if (this.newTaskName() != "") {
75             this.tasks.push(newTask);
76             this.newTaskName("");
77         }
78
79     }.bind(this);
80
81     this.addMember = function(contact) {
82
83         // Nicht hinzufügen, wenn Kontakt bereits Mitglied.
84         /*
85          * TODO Genauer Prüfen!! Bspw. über E-Mail-Adressen (Über reload der
86          * Kontakte können Objekte doch wieder zweimal hinzugefügt werden).
87          */
88         if (this.isMember(contact)) {
89             console.warn("Der Kontakt '" + contact.name.formatted
90                         + "' ist bereits Mitglied der Liste.");
91             return;
92         }
93
94         console.log("addMember: " + contact.name.formatted);
95         this.members.push(contact);
96
97     }.bind(this);
98
99     this.removeMember = function(member) {
100        console.log("removeMember: " + member.name.formatted);
101        this.members.remove(member);
102
103    }.bind(this);
104
105    this.isMember = function(contact) {
106        var result = this.members().indexOf(contact) > -1;
107        console.debug("isMember(" + contact.name.formatted + ") : " + result);
108        return result;
109    };
110
111 }

```

Listing B.4: Task.js.

```

1 /*****
2  * Task Object
3 ****/
4 /*
5  * TODO Weitere Attribute hinzufügen
6 */
7 function Task(id, name, done) {
8  /*
9   * =====
10  * Properties
11  * =====
12  */
13 this.id = id;
14 this.name = ko.observable(name);
15 this.done = ko.observable(done);
16
17 /*
18  * =====
19  * Operations
20  * =====
21 */
22
23 }

```

Listing B.5: ObserverMap.js.

```

1 /**
2  * Helper for an observable object.
3  *
4  * Saves a list of the observer's eventHandlers for each eventType, that are
5  * mapped to a given eventType-string and calls them when notifyObservers() is
6  * called.
7 */
8 function ObserverMap() {
9  console.debug("Init new ObserverMap.");
10
11  var self = this;
12
13  self.handlerLists = new Array();
14
15 /**
16  * Adds an EventHandler to the collection.
17  *
18  * @param {String} eventType
19  *           The string that specifies the type of the event, on which
20  *           appearance the eventHandler should be called.
21  * @param {Function} eventHandler
22  *           A function that is called, when the event is dispatched.
23 */
24 self.put = function(eventType, eventHandler) {
25
26  var handlerList = self.findHandlerList(eventType);
27
28  if (handlerList == null) {
29    // If no HandlerList was found, create a new one.
30    console.debug("No matching HandlerList found, "
31      + "creating a new one.");
32    handlerList = new HandlerList(eventType);
33    self.handlerLists.push(handlerList);

```

```
34     }
35     handlerList.addEventHandler(eventHandler);
36
37 };
38
39 /**
40  * Notifies all saved observers that are registered for the given eventType.
41 *
42 * @param eventType
43 *          The string that specifies the type of the event, on which
44 *          appearance the eventHandler should be called.
45 * @param data
46 *          Some data, that the model can ship with the event dispatch, so
47 *          that observers can handle them.
48 */
49 self.notifyObservers = function(eventType, data) {
50   console.debug("Searching for observer to notify...");
51
52   var matchingHandlerList = self.findHandlerList(eventType);
53
54   if (matchingHandlerList != null) {
55     console.debug();
56     matchingHandlerList.notifyObservers(data);
57   } else {
58     console.warn("No Observer with type '" + eventType + "' found. "
59                 + "Nobody to notify.");
59   }
60
61 };
62
63
64 /**
65  * Find a List of EventHandlers for a specified eventType string.
66 */
67 self.findHandlerList = function(eventType) {
68   console.debug("Searching for matching HandlerList....");
69
70   var result = null;
71
72   self.handlerLists.forEach(function(handlerList) {
73     if (handlerList.eventType.valueOf() === eventType.valueOf()) {
74       console.debug("Found matching HandlerList: " + eventType);
75       result = handlerList;
76     }
77   });
78
79   if (result == null) {
80     console.debug("No handlerList for type '" + eventType + "' found.");
81   }
82
83   return result;
84 };
85 }
86
87 /**
88  * A List of EventHandlers for one EventType.
89 *
90 * @param eventType
91 *          A String that specifies the Type of the event.
92 */
93 function HandlerList(eventType) {
94   var self = this;
```

```

95
96     console.debug("Init HandlerList for type '" + eventType + "'.");
97
98     self.eventType = eventType;
99     self.eventHandlers = new Array();
100
101    self.addEventHandler = function(eventHandler) {
102        console.debug("Adding EventHandler '" + eventHandler.name + "'.");
103        self.eventHandlers.push(eventHandler);
104    };
105
106    self.notifyObservers = function(data) {
107        console.debug("HandlerList '" + self.eventType
108            + "': Notify observers...'");
109        self.eventHandlers.forEach(function(eventHandler) {
110            eventHandler(data);
111        });
112    };
113
114 }

```

Listing B.6: In der Datei consts.js werden allgemeine String-Konstanten sowie Event-Bezeichner definiert, die in der gesamten Anwendung nach Laden dieses Skripts (siehe Listing B.1) verwendet werden können, um Tipp-Fehler zu vermeiden und die Lesbarkeit des Codes zu erleichtern.

```

1 /*****
2 * Konstanten für die gesamte Anwendung
3 *****/
4 var consts = {
5     TITLE : "Besorgungen",
6     LIST : "list",
7     TASK : "task",
8     USER : "user",
9     LISTVIEW : "_listview",
10    LIST_MEMBERS : "Listenmitglieder",
11    CONTACTS_PANEL : 'contactsPanel',
12    DUMMY_CONTACT : null
13};
14
15 /**
16 * EventTypes for mapping EventHandlers to a specific type of event.
17 */
18 var events = {
19     DEVICE_READY : 'deviceready',
20     FOUND_CONTACTS : 'foundcontacts'
21};

```

Listing B.7: util.js.

```

1 /*****
2 * Allgemeine Utility-Funktionen, die von überall per util.FUNCTIONNAME()
3 * aufgerufen werden können
4 *****/
5
6 var util = {
7     /*****
8     * Script zum generieren einer rfc4122 version 4 Uniqpe ID. Quelle:
9     * https://stackoverflow.com/questions/105034/
      how-to-create-a-guid-uuid-in-javascript

```

```

10  /*
11   * createUID : function(prefix) {
12   *
13   *     var uid = 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/xy]/g,
14   *         function(c) {
15   *             var r = Math.random() * 16 | 0, v = c == 'x' ? r
16   *                 : (r & 0x3 | 0x8);
17   *             return v.toString(16);
18   *         });
19   *
20   *     return prefix + uid;
21   * },
22   ****
23   * Erstellt ein leeres Dummy-Contact-Object nach der
24   * navigator.contacts-Schnittstelle
25   * http://plugins.cordova.io/#/package/org.apache.cordova.contacts
26   */
27   createDummyContact : function() {
28     return new Contact(null, null, new ContactName(null, null, null, null,
29           null, null), null, [], [], [], [], [], null, null, [], [], []);
30   },
31
32   getDummyContact : function() {
33     if (consts.DUMMY_CONTACT === null) {
34       consts.DUMMY_CONTACT = this.createDummyContact();
35     }
36     return consts.DUMMY_CONTACT;
37   },
38
39   ****
40   * Initial Test-Data:
41   ****
42   getTestUsers : function() {
43     return new Array(new Contact(util.createUID(consts.USER), "Donald",
44       new ContactName("Donald Duck", "Duck", null, null, null, null),
45       "Don", null, null, null, null, null, null, null, null,
46       new Contact(util.createUID(consts.USER), "Mickey",
47       new ContactName("Mickey Mouse", "Mouse", "Mickey", null, null,
48       null), "Mick", null, null, null, null, null, null,
49       null, null, null, null), new Contact(util
50       .createUID(consts.USER), "Dagobert", new ContactName(
51       "Dagobert Duck", "Duck", "Dagobert", null, null, null),
52       "Daggy", null, null, null, null, null, null, null, null,
53       null));
54   },
55
56   getTestLists : function() {
57     return new Array(new List("list1", "Privat", [ this.getTestUsers()[0],
58       this.getTestUsers()[1] ], [
59         new Task("task1", "Kind abholen", true),
60         new Task("task2", "Arzttermin", true) ]), new List("list2",
61       "Arbeit", [ this.getTestUsers()[2] ], [
62         new Task("task3", "Druckerpatronen", false),
63         new Task("task4", "Papier", false) ]), new List(
64       "list3", "Haushalt", this.getTestUsers(), [
65         new Task("task5", "Milch", false),
66         new Task("task6", "Brot", false) ]));
67   },
68   ****
69   * Mehre CSS-Klassen dynamisch meheren Elementen eines Typs (Bsp.

```

```

70     * ' addButton') zuweisen, statt die selbe Reihe von Class-Werten mehrfach zu
71     * definieren.
72     */
73     applyStyles : function() {
74
75         $('.add-button').addClass(
76             "ui-btn ui-icon-plus ui-btn-icon-left ui-btn-inline ui-shadow");
77         $('.close-button')
78             .addClass(
79                 "ui-btn ui-icon-delete ui-btn-icon-left ui-btn-inline ui-shadow");
80         $('.ok-button').addClass(
81             "ui-btn ui-icon-check ui-btn-icon-left ui-shadow");
82         $('.overlay-panel')
83             .addClass(
84                 "ui-panel ui-panel-position-right ui-panel-display-overlay ui-body-b
85                 ui-panel-animate ui-panel-open");
86         $('[data-role=button]').addClass("ui-corner-all");
87
88         console.debug("Applying custom styles...");
89     }
90 };

```

Listing B.8: model.js.

```

1  /*
2  * Licensed to the Apache Software Foundation (ASF) under one
3  * or more contributor license agreements. See the NOTICE file
4  * distributed with self work for additional information
5  * regarding copyright ownership. The ASF licenses self file
6  * to you under the Apache License, Version 2.0 (the
7  * "License"); you may not use self file except in compliance
8  * with the License. You may obtain a copy of the License at
9  *
10 * http://www.apache.org/licenses/LICENSE-2.0
11 *
12 * Unless required by modellicable law or agreed to in writing,
13 * software distributed under the License is distributed on an
14 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
15 * KIND, either express or implied. See the License for the
16 * specific language governing permissions and limitations
17 * under the License.
18 */
19
20 /*
21 * TODO Eigenleistung deutlich machen!!!
22 */
23 var model = {
24
25     deviceReady : false,
26     observerMap : new ObserverMap(),
27
28     /*****
29     * Model Constructor
30     */
31     initialize : function() {
32         console.log("Connecting to Device...");
33         this.bindEvents();
34     },
35
36     *****

```

```
37  * Bind Event Listeners
38  *
39  * Bind any events that are required on startup. Common events are: 'load',
40  * events.DEVICE_READY, 'offline', and 'online'.
41  */
42 bindEvents : function() {
43     document.addEventListener(events.DEVICE_READY, this.onDeviceReady,
44         false);
45 },
46
47 /**
48  * deviceready Event Handler
49  *
50  * The scope of 'this' is the event. In order to call the 'receivedEvent'
51  * function, we must explicitly call 'model.receivedEvent(...);'
52  */
53 onDeviceReady : function() {
54     model.deviceReady = true;
55     model.receivedEvent(events.DEVICE_READY);
56 },
57
58 /**
59  * Update DOM on a Received Event
60  */
61 receivedEvent : function(id) {
62     console.log('model received Event: ' + id);
63 },
64
65 /**
66  * DeviceAccess:
67  * TODO options, filter (und evtl. auch onSuccess) als Parameter übergeben.
68  */
69 findContacts : function() {
70
71     if (!model.deviceReady) {
72         var errorMsg = 'Gerät ist nicht bereit!';
73         console.error(errorMsg);
74         alert(errorMsg);
75         return;
76     }
77
78     var onSuccess = function(contacts) {
79         var successMsg = contacts.length + ' Kontakte gefunden.';
80         console.log(successMsg);
81         model.observerMap.notifyObservers(events.FOUND_CONTACTS, contacts);
82     };
83
84     var onError = function(contactError) {
85         var errorMsg = 'Fehler beim Laden der Kontakte!';
86         console.error(errorMsg);
87         alert(errorMsg);
88     };
89
90     var options = new ContactFindOptions();
91     options.filter = "";
92     options.multiple = true;
93     var fields = [ "*" ];
94     navigator.contacts.find(fields, onSuccess, onError, options);
95
96 },
97
```

```

98 /**
99 *
100 */
101 addEventListener : function(eventType, eventHandler) {
102     model.observerMap.put(eventType, eventHandler);
103 },
104
105 removeEventListener : function(eventType, eventHandler) {
106     // TODO implement.
107     console.error("removeEventListener(): Diese Methode ist noch nicht implementiert.
108     ");
109 }
110 };

```

Listing B.9: ViewModel.js.

```

1 ****
2 * Main ViewModel:
3 ****
4 function ErrandsViewModel(lists) {
5     console.log("Init ViewModel...");
6
7     var self = this;
8
9     /*
10      * =====
11      * Properties:
12      * =====
13     */
14
15     self.lists = ko.observableArray(lists);
16     self.newListName = ko.observable("");
17     self.contacts = ko.observableArray([ util.getDummyContact() ]);
18
19     self.countAllOpenTasks = ko.computed(function() {
20         var count = 0;
21         self.lists().forEach(function(list) {
22             count += list.countOpenTasks();
23         });
24         return count;
25     }, self);
26
27     self.heading = ko.computed(function() {
28         var openTasks = self.countAllOpenTasks();
29         if (openTasks > 0) {
30             return consts.TITLE + " (" + openTasks + ")";
31         } else {
32             return consts.TITLE;
33         }
34     }, self);
35
36     /*
37      * =====
38      * Operations
39      * =====
40     */
41
42 /**
43 *
44 */

```

```

45 self.addList = function() {
46   var newListID = util.createUID(consts.LIST);
47
48   var newList = new List(newListID, self.newListName(), [], []);
49   if (self.newListName() != "") {
50     self.lists.push(newList);
51     self.newListName("");
52   }
53 }.bind(self);
54
55 /**
56  * TODO Besser wäre, wenn die Daten aus dem Model an die aus dem ViewModel
57  * gebunden werden könnten, sodass die GUI sich automatisch aktualisiert,
58  * wenn sich die Daten im Model geändert haben.
59 */
60 self.findContacts = function(event, ui) {
61   console.debug("Fordere Kontaktdaten an...");
62   model.findContacts();
63 };
64
65 /**
66  *
67  */
68 self.contactsNotEmpty = ko.computed(function() {
69   return (self.contacts().length != 0
70         && self.contacts()[0] != util.getDummyContact());
71 }, this);
72
73 /**
74  * TODO Rausschmeißen, wenn nicht benötigt.
75 */
76 self.store = function() {
77   var data = JSON.stringify(self);
78   console.log("storing: " + data);
79   localStorage['lists'] = data;
80 };
81
82 /**
83  * TODO Rausschmeißen, wenn nicht benötigt.
84 */
85 self.restore = function() {
86   var data = localStorage['lists'];
87   if (data != undefined) {
88     console.log("restored: " + data);
89   }
90 };
91
92 self.bindEvents = function() {
93   model.addEventListener(events.FOUND_CONTACTS, function(contacts) {
94     self.contacts(contacts);
95   });
96 };
97
98 self.bindEvents();
99 }

```

Listing B.10: bindingHandlers.js.

```

1 ****
2 * Trigger for re-apply QJM-Styles after adding items
3 ****

```

```
4 ko.bindingHandlers.jqmRefreshList = {
5   update : function(element, valueAccessor) {
6     // just to create a dependency:
7     ko.utils.unwrapObservable(valueAccessor());
8
9     // Refresh Listview after update:
10    $(element).listview().listview("refresh");
11
12    // Refresh CheckboxFieldset after update
13    $("[data-role=controlgroup]").enhanceWithin().controlgroup()
14      .controlgroup("refresh");
15
16    $(element).enhanceWithin().collapsibleset().collapsibleset("refresh");
17
18    // Apply dynamic style classes:
19    util.applyStyles();
20  }
21};
```

Listing B.11: index.js.

```
1 /**
2  * Main start script for the app.
3  */
4 app = {
5   viewModel : null,
6   initialize : function() {
7
8     // Initialize Model
9     model.initialize();
10
11    // Initialize ViewModel with test data
12    this.viewModel = new ErrandsViewModel(util.getTestLists());
13    ko.applyBindings(this.viewModel);
14    console.log("Applying DataBindings...");
15
16    // Apply custom styles
17    util.applyStyles();
18
19  }
20};
```

Anhang C

Glossar

Begriffe und Abkürzungen

ActionBar	Eine vierteilige Menüleiste am oberen Bildschirmrand.
API	Application Programming Interface (dt. „Programmierschnittstelle“).
App	Kurzform für engl. „Application“ (dt. „Anwendung“) im Sinne von Anwendungssoftware. Im deutschsprachigen Raum meist im Zusammenhang mit Smartphones oder Tablet-Computern.
Array	.
asynchron	.
Built	.
Cross-Compiling	.
Data-Binding	Verbindung von UI-Komponenten mit Datenfeldern auf Programmebene.
DOM	Document Object Model.
Entwicklungsumgebung	Eine Entwicklungsumgebung (IDE, für engl. „Integrated Development Environment“) ist eine Software, mit der Computer-Programme entwickelt werden können..
Error-Handler	.

Event	.
Event-Handler	.
Flag	.
Framework	.
Frontend	.
GPS	Global Positioning System.
GUI	Graphical User Interface (dt. „Grafische Benutzeroberfläche“).
Handler	.
Hybrid-App	.
ID	engl. „Identifier“. Eindeutiger Kenner (bspw. eines Objekts)..
iOS Developer Program	.
JavaEE	.
JFace	.
Listview	.
Look-And-Feel	Aussehen und Verhalten einer Benutzeroberfläche.
Look-And-Feel	.
Model	.
MVC	.
MVVM-Pattern	Model-View-ViewModel-Pattern. Ein Entwurfsmuster (engl. „Pattern“) der Informatik zur Trennung von Benutzeroberfläche und UI-Logik.
Observable	.
Observer	.
Observer-Pattern	.

Plattform	Basis-Software-Umgebung, in der eine andere Software (bspw. Anwendungen und Dienste) ausgeführt wird. Hier ist das Betriebssystem mobiler Geräte gemeint.
Plugin	.
QR-Code	.
SDK	Software Development Kit.
String	.
SWF	.
SWT	.
Tabris	.
Tabris UI	.
Tag	Ein Element von Auszeichnungssprachen. In XML-Dialekten wie bspw. HTML erkennbar an der Syntax mit spitzen Klammern. (Bsp.: <head>).
UI	User Interface (dt. „Benutzerschnittstelle“).
URL	Uniform Resource Locator (dt. „einheitlicher Quellenanzeiger“) steht für die Angabe eines Ortes in Netzwerken.
View	.
ViewController	.
ViewModel	.
W3C	World Wide Web Consortium.
Web-App	.
WebView	.
Widget	.
Xamarin.Android	.
Xamarin.iOS	.

Technologien

Knockout	.
jQuery	.
Alloy	.
Amazon FireOS	.
Android	Smartphone- und Tablet-Betriebssystem von Google.
Android Studio	Eine Entwicklungsumgebung für die Entwicklung von Apps für das Betriebssystem Android.
App Framework	.
Appcelerator	.
Blackberry 10	Smartphone-Betriebssystem für <i>Blackberry</i> -Geräte der Firma Blackberry. Blackberry 10 ist der Nachfolger des vorherigen Betriebssystems <i>Blackberry OS</i> .
Bytecode	.
C	Eine imperative Programmiersprache.
C++	.
C#	.
CLI	Kommandozeilen-Werkzeug (oder „Befehlszeilenschnittstelle“ von engl. Command-Line Interface (CLI)).
Cocoa Touch Widgets	.
Contacts API	.
Cordova	.
CSS	Cascading Style Sheets.
Eclipse	Eine u.a. durch Plugins stark anpassbare Entwicklungsumgebung von der <i>Eclipse Foundation</i> .
Firefox OS	.
Flash-Editing-Environment	.

Git	.
GitHub	Online-Portal, in dem Git-Repositories erstellt und verwaltet werden können mit dem Fokus auf Open-Source-Software.
HTML	HTML (Hyper Text Markup Language) ist eine Auszeichnungssprache für Websites. „HTML is the publishing language of the World Wide Web“ [36].
HTML5	Version 5 der HTML-Spezifikation. Unter HTML5 werden im Allgemeinen Web-Technologien wie HTML, CSS und JavaScript zusammengefasst, die es Entwicklern ermöglichen, auch komplexere Web-Anwendungen ohne die Notwendigkeit von zusätzlichen Technologien wie Browserplugins etc. zu entwickeln.
IndexedDB	Eine Datenbank....
IntelliJ IDEA	Eine Entwicklungsumgebung für Java der Firma JetBrains.
iOS	Smartphone- und Tablet-Betriebssystem von Apple.
iOS-Simulator	.
JavaScript	Eine Skriptsprache, die in einer größeren Umgebung (wie beispielsweise einem Browser) ausgeführt wird und in diesem Kontext z. B. verwendet werden kann, um HTML um dynamische Elemente, wie der Veränderung von Inhalten, aber auch Ausführung von Prozeduren etc., zu erweitern [16].
jQuery Mobile	.
JSON	JavaScript Object Notation.
Lime	.
Mac OS X	Betriebssystem für Mac-Rechner der Firma Apple.

Mono	.
MonoTouch	.
NekoVM	.
Netbeans IDE	Eine Entwicklungsumgebung auf Java-Basis von der <i>Oracle Corporation</i> .
ObjectiveC	Eine um objektorientierte Elemente erweiterte Variante der Programmiersprache C.
OpenFL	.
OSGi	.
PhoneGap	.
PhoneGap Build	Online-Portal von Adobe, das den Build-Prozess von PhoneGap-Apps in die Cloud auslagert und damit die Anbindung von Hybrid-Apps an plattformspezifische Komponenten verschiedener Mobilgeräte erleichtern soll.
Plugin Development Guide	.
Plugin Registry	.
Symbian	.
Titanium	.
Ubuntu Phone	.
WebSQL	.
WebStorage	WebStorage (auch: „LocalStorage“) ist eine Spezifikation des W3C zur lokalen Speicherung von größeren Datenmengen einer Webanwendung auf Client-Seite in Form von Schlüssel-Wert-Paaren.) [15].
webOS	.
Windows Phone	Smartphone-Betriebssystem von Microsoft.
Windows 8	Aktuelles Desktop- / Hybrid-Betriebssystem von Microsoft.

Xamarin

Xcode

Eine Entwicklungsumgebung für die Entwicklung von iOS- und Mac OS X-Software.

XML

XMLVM

Quellenverzeichnis

Bildquellen

- [1] URL: http://cordova.apache.org/docs/en/3.4.0/img/guide/cli/android_emulate_install.png (siehe S. 28).
- [2] *MVVMPattern*. Wikimedia Commons. URL: <http://commons.wikimedia.org/wiki/File:MVVMPattern.png> (besucht am 13. 04. 2014) (siehe S. 16).

Software-Quellen und Dokumentationen

- [3] *Apache Cordova Documentation*. Apache Foundation. Kap. iOS Platform Guide. URL: http://cordova.apache.org/docs/en/3.4.0/guide_platforms_ios_index.md.html#iOS%20Platform%20Guide (besucht am 28. 04. 2014) (siehe S. 4).
- [4] *Apache Cordova Documentation*. Apache Software Foundation. Kap. Plugin APIs. URL: http://cordova.apache.org/docs/en/3.4.0/cordova_plugins_pluginapis.md.html#Plugin%20APIs (besucht am 23. 05. 2014) (siehe S. 30, 31).
- [5] *Apache Cordova Documentation*. Apache Software Foundation. Kap. Plugin Development Guide. URL: http://cordova.apache.org/docs/en/edge/guide_hybrid_plugins_index.md.html#Plugin%20Development%20Guide (besucht am 23. 05. 2014) (siehe S. 30, 31).
- [6] *Apache Cordova Documentation*. Feb. 2014. Kap. Overview. URL: http://cordova.apache.org/docs/en/3.4.0/guide_overview_index.md.html#Overview (besucht am 22. 04. 2014) (siehe S. 22).
- [7] *Apache Cordova Documentation*. Feb. 2014. Kap. The Command-Line Interface. URL: http://cordova.apache.org/docs/en/3.4.0/guide_cli_index.md.html#The%20Command-Line%20Interface (besucht am 22. 04. 2014) (siehe S. 22, 24, 27, 29, 30).
- [8] *Apache Cordova Documentation*. Apache Foundation. Feb. 2014. Kap. Platform Support. URL: http://cordova.apache.org/docs/en/3.4.0/guide_support_index.md.html#Platform%20Support (besucht am 22. 04. 2014) (siehe S. 23).

- [9] *Cordova Android*. Version 3.5.0. Apache Software Foundation, Feb. 2014. URL: <https://github.com/apache/cordova-android/blob/3.5.0/framework/assets/www/cordova.js#L1178> (siehe S. 32).
- [10] *Cordova Plugin Registry*. Apache Software Foundation. Kap. Browse all 229 plugins. URL: <http://plugins.cordova.io/#/viewAll> (besucht am 23.05.2014) (siehe S. 31).
- [11] *Cordova Plugin Registry*. Version 0.2.8. Apache Software Foundation. Kap. org.apache.cordova.battery-status. URL: <http://plugins.cordova.io/#/package/org.apache.cordova.battery-status> (besucht am 24.05.2014) (siehe S. 32, 33).
- [12] *Cordova Plugin Registry*. Version 0.2.9. Apache Software Foundation. Kap. org.apache.cordova.device. URL: <http://plugins.cordova.io/#/package/org.apache.cordova.device> (besucht am 24.05.2014) (siehe S. 32, 33).
- [13] *Cordova Plugin Registry*. Version 0.2.10. Apache Software Foundation. Apr. 2014. Kap. org.apache.cordova.contacts. URL: <http://plugins.cordova.io/#/package/org.apache.cordova.contacts> (besucht am 15.05.2014) (siehe S. 31, 32, 43, 44, 47–49).
- [14] Ian Hickson. *Web SQL Database*. W3C. 18. Nov. 2010. URL: <http://www.w3.org/TR/2010/NOTE-webdatabase-20101118/> (siehe S. 7).
- [15] Ian Hickson. *Web Storage*. W3C. 2013. URL: <http://www.w3.org/TR/2013/REC-webstorage-20130730/> (siehe S. 7, 88).
- [16] *JavaScript Web APIs*. W3C. URL: <http://www.w3.org/standards/webdesign/script> (besucht am 08.04.2014) (siehe S. 87).
- [17] Nikunj Mehta u. a. *Indexed Database API*. W3C. Juli 2013. URL: <http://www.w3.org/TR/2013/CR-IndexedDB-20130704/> (siehe S. 7).
- [18] *org.apache.cordova.contacts*. Version 3.0.0. Juli 2013. URL: <https://github.com/apache/cordova-plugin-contacts/blob/3.0.0/www/contacts.js> (siehe S. 32).
- [19] *PhoneGap Build Documentation*. Kap. Supported Platforms. URL: http://docs.build.phonegap.com/en_US/introduction_supported_platforms.md.html#Supported%20Platforms (besucht am 22.04.2014) (siehe S. 34).
- [20] *PhoneGap Build Documentation*. Kap. Getting Started with Build. URL: http://docs.build.phonegap.com/en_US/introduction_getting_started.md.html#Getting%20Started%20with%20Build (besucht am 15.04.2014) (siehe S. 35).

Online-Quellen

- [21] *Adobe Announces Agreement to Acquire Nitobi, Creator of PhoneGap.* URL: <http://www.adobe.com/aboutadobe/pressroom/pressreleases/201110/AdobeAcquiresNitobi.html> (besucht am 15.04.2014) (siehe S. 20).
- [22] *Adobe PhoneGap Build.* URL: <https://build.phonegap.com/apps> (besucht am 14.04.2014) (siehe S. 34).
- [23] *Choose your plan.* URL: <https://build.phonegap.com/plans> (besucht am 15.04.2014) (siehe S. 35).
- [24] Xavier Ducrohet, Tor Norbye und Katherine Chou. *Android Studio: An IDE built for Android.* Mai 2013. URL: <http://android-developers.blogspot.in/2013/05/android-studio-ide-built-for-android.html> (siehe S. 4).
- [25] *iOS Developer Program.* Apple Inc. URL: <https://developer.apple.com/programs/ios/> (besucht am 28.04.2014) (siehe S. 4).
- [26] *jQuery Examples.* URL: http://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_hide_p (besucht am 11.04.2014) (siehe S. 15).
- [27] *jQuery Examples.* URL: http://www.w3schools.com/jquerymobile/tryit.asp?filename=tryjqmob_start (besucht am 13.04.2014) (siehe S. 19).
- [28] *Kostenlose GMX Mail App. Ihr Postfach immer & überall dabei.* 1&1 Mail & Media GmbH. URL: <http://www.gmx.net/produkte/mobile/mail-app> (siehe S. 6).
- [29] 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. Nov. 2013. URL: <http://www.idc.com/getdoc.jsp?containerId=prUS24442013> (siehe S. 3).
- [30] *Model View ViewModel.* URL: <http://de.wikipedia.org/wiki/MVVM> (besucht am 13.04.2014) (siehe S. 16).
- [31] *Offline - HTML5 Rocks.* Google Inc. URL: <http://www.html5rocks.com/de/features/storage> (siehe S. 7).
- [32] PhoneGap Blog. *PhoneGap, Cordova, and what's in a name?* 19. März 2012. URL: <http://phonegap.com/2012/03/19/phonegap-cordova-and-what%E2%80%99s-in-a-name/> (besucht am 15.04.2014) (siehe S. 21).
- [33] Rene Ritchie. *Google releases official Gmail app for iPhone, iPad.* Nov. 2011. URL: <http://www.imore.com/google-releases-official-gmail-app-iphone-ipad> (siehe S. 6).

- [34] *SELFHTML: JavaScript / Objektreferenz / navigator*. URL: http://de.selfhtml.org/javascript/objekte/navigator.htm#java_enabled (besucht am 24.05.2014) (siehe S. 32).
- [35] *SELFHTML: JavaScript / Objektreferenz / window*. Jan. 2014. URL: <http://de.selfhtml.org/javascript/objekte/window.htm> (besucht am 24.05.2014) (siehe S. 32).
- [36] *W3C HTML*. URL: <http://www.w3.org/html/> (besucht am 06.04.2014) (siehe S. 87).
- [37] *Want to Contribute?* Apache Software Foundation. URL: <http://cordova.apache.org/#contribute> (besucht am 02.06.2014) (siehe S. 30).
- [38] *WEB.DE App - Postfach Apps für Smartphone, iPhone und iPad*. 1&1 Mail & Media GmbH. URL: https://produkte.web.de/freemail_mobile_startseite (siehe S. 6).