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

Kapitel 1

Einleitung

1.1 Ziel der Arbeit

1.2 Motivation

1.3 Aufbau der Arbeit

Kapitel 2

Theoretische Grundlagen

2.1 Apps für mobile Geräte

2.1.1 Mobile (native) Apps

Unter mobilen *Apps (Kurzform für engl. „Application“)* versteht man im Allgemeinen Anwendungssoftware für Tablet-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 OS* des gleichnamigen Smartphone-Herstellers *Blackberry* [21]. 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 nötig. Seit 2013 bietet Google darüber hinaus die auf IntelliJ IDEA basierende und eigens für die Android-Entwicklung angepasste Entwicklungsumgebung *Android Studio* an [16], 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 [17]. 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

[1] Diese Restriktion fällt allerdings auch in der unten beschriebenen plattformunabhängigen App-Entwicklung nicht unbedingt weg (siehe Abschnitt 2.2).

Toolkits, Bibliotheken und Programmiersprachen verwendet, was die native App-Entwicklung für potenzielle Auftraggeber zu einem sehr kostenaufwändigen Projekt werden lassen kann. Andererseits bietet die native App-Entwicklung vollständige Unterstützung der betriebssystemeigenen Funktionalitäten wie den Zugriff auf Kamera, Adressbuch, Bewegungssensoren etc. der jeweiligen Plattform, sodass ein Softwareprojekt mit solchen besonders hardware- oder betriebssystemnahen Anforderungen die Entwicklung einer nativen (plattformspezifischen) App notwendig erscheinen lassen kann.[2]

2.1.2 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 Webbrowser 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 Zielplattformen bestimmt, da von Client-Seite aus verschiedene Brow-

[2] Mehr dazu in Abschnitt 2.2
[3] Die Rolle des Clients übernimmt hier also der Browser.
[4] Einige sind beispielsweise *PHP*, Java, *ASP* u. v. a. m.

ser durch die Verbreitung und Beachtung von Web-Standards weitgehend einheitliche HTML-Dokumente lesen und interpretieren können und das Backend nicht auf Clients mit unterschiedlichen Plattformen, sondern auf Webservern liegt, deren Plattform bei der Entwicklung entweder schon bekannt oder nicht relevant ist.[5]

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 [20, 25, 26]. 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 Oberfläche[6] 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 ande-

[5] Beispielsweise weil auch die Fachlogik plattformunabhängig mit PHP oder Java realisiert wurde.

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

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

res 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, 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 [9] sowie *WebSQL* bzw. *IndexedDB*, beides auf Web-Anwendungen optimierte Datenbanken-Spezifikationen, die vom *W3C* herausgegeben werden [8, 10]. Allerdings bestehen auch bei diesen Mechanismen teilweise Einschränkungen durch die Browservielfalt 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 Browsern unterstützt [23]. 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

[8] hier also der Website

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.

2.1.3 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 Web-View 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 hybri-

[9] siehe Unterabschnitt 2.1.3 und 2.2.

den 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 erfordert.[10]

2.2 Entwicklung von hybriden Apps

Wie in 2.1.3 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.

2.2.1 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 deut-

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

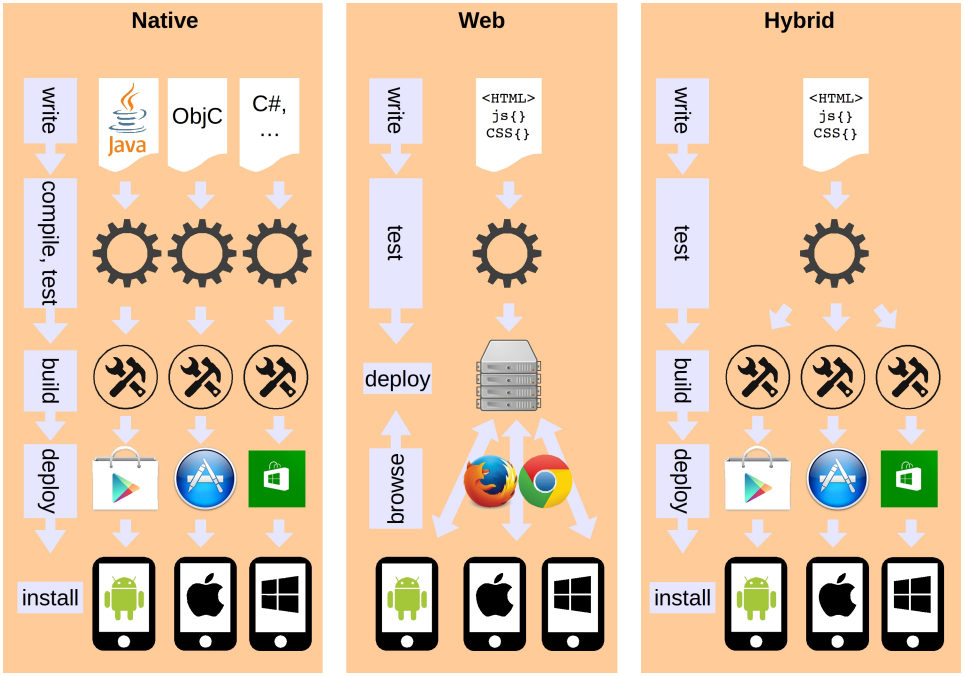

Abbildung 2.1: Entwicklungsstufen der verschiedenen Arten von Apps. Während bei der nativen App der gesamte Entwicklungszyklus einmal pro Plattform durchlaufen werden muss, verringert sich der Aufwand für die Web-App erheblich. Bei der Hybrid-App muss die Anwendung zwar einmal für jede Plattform gebaut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

lich 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 `getElementsByTagName()` aufgerufen werden muss (siehe Listing 2.2), ist bei jQuery der Zugriff auf alle Elemente eines *Tags* per Dollar(`$`)-Notation deutlich verkürzt (siehe Listing 2.1). Auch die nächste Anweisung zur Ausführung einer Operation für alle ausgewählten Elemente (hier: `hide()`, also *ausblenden*) fällt bei jQuery wesentlich kürzer aus, indem die Funktion noch in der selben Zeile wie der vorherigen Selektor aufgerufen werden kann (`$("`

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 2.2, Zeilen 5 - 7).

Listing 2.1: jQuery-Beispiel-Code zum Ausblenden aller Absätze [18].

```
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 2.2: Die gleiche Funktionalität wie in Listing 2.1 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>
```

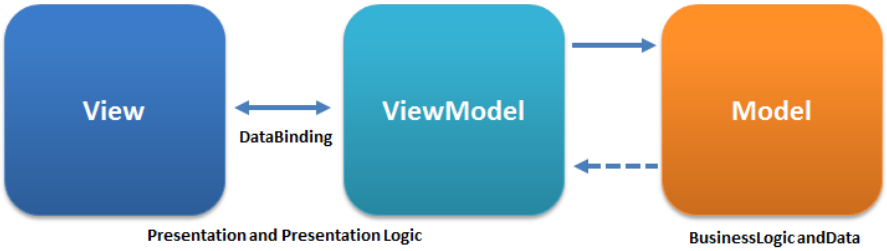

Abbildung 2.2: Schematische Darstellung des Entwurfsmusters MVVM: Die View ist über das Data-Binding mit dem ViewModel verbunden, indem die UI-Logik implementiert ist und das mit der Geschäftslogik (Model) interagiert.

Quelle: Wikimedia Commons [2].

2.2.2 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 Datenfeldern auf Programmebene, erheblich vereinfacht. Das Data-Binding wird bei Knockout durch das *Model-View-ViewModel (MVVM)*-Pattern 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 Benutzerschnittstellen erleichtern, indem die UI- und die Geschäftslogik von Softwareentwicklern übernommen werden kann, während Designer den Schwerpunkt auf die Gestaltung der Oberfläche legen können [22].

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 2.2).

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 2.3) 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 generiert 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, sodass 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 2.3).[11]

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

Listing 2.3: Einfaches Anwendungsbeispiel für die Verwendung der JavaScript-Bibliothek Knockout.

```
<html>
  <head>
    <script src="lib/jquery-1.10.2.min.js"></script>
    <script src="lib/knockout-3.1.0.js"></script>
    <script>
      // jQuery-Standard: Erst ausführen, wenn Dokument geladen.
      $(document).ready(function() {
        // ViewModel:
        var ViewModel = function(first, last) {
          this.firstName = ko.observable(first);
          this.lastName = ko.observable(last);
          this.fullName = ko.computed(function() {
            return this.firstName() + " " + this.lastName();
          }, this);
        };
        ko.applyBindings(new ViewModel("Planet", "Earth"));
      });
    </script>
  </head>
  <body>
    <p>First name:
      <input data-bind="value: firstName" />
    </p>
    <p>Last name:
      <input data-bind="value: lastName" />
    </p>
    <h2>Hello, <span data-bind="text: fullName"> </span>!</h2>
  </body>
</html>
```

2.2.3 JQuery Mobile

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

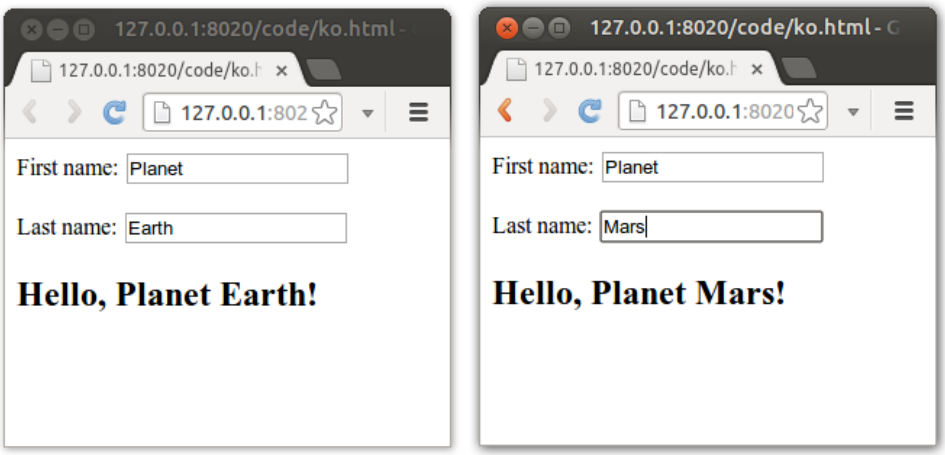

Abbildung 2.3: Knockout-Beispiel aus Listing 2.3 im Browser. Links im Bild: Anzeige bei Initialisierung der Oberfläche, rechts: Benutzereingabe ins Eingabefeld: „Mars“.
Änderungen in der UI (Hier: im Eingabefeld) werden sofort im ViewModel registriert und automatisch an alle verknüpften Anzeigen weitergereicht (Hier an das fettgedruckte Begrüßungselement).

Quelle: Eigener Screenshot.

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 2.4, Zeilen 3 - 5).

Listing 2.4: Einbindung von jQuery Mobile in eine HTML-Seite [19].

```
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. zugeteilt werden können, anhand derer das jQuery Mobile-Framework den HTML-Elementen die jeweilige Style-Definition aus dem Stylesheet zuweisen kann (siehe Listing 2.4, 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 2.4 und 2.6).

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 2.4, Zeile 12) für ein besseres Layout des `<div>`-Inhalts, wie der Vergleich ohne das `class`-Attribut in Abbildung 2.5 zeigt.

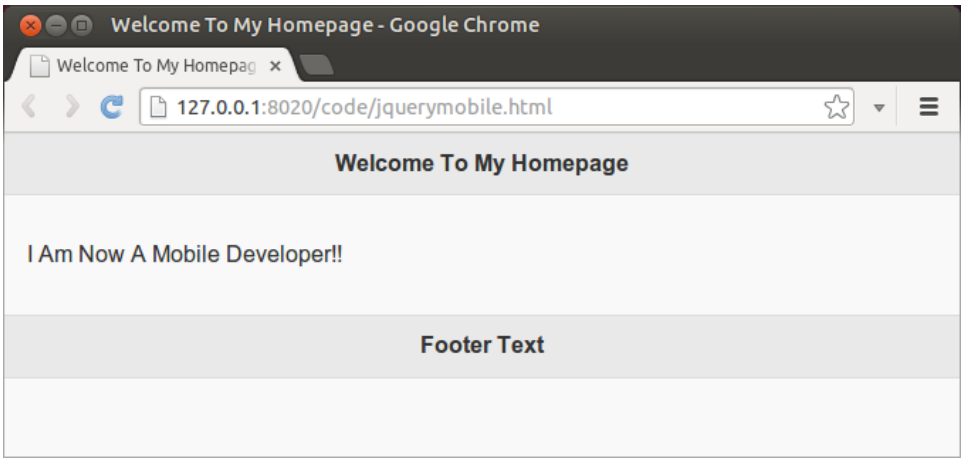

Abbildung 2.4: jQuery Mobile-Beispiel aus Listing 2.4 im Browser.
Quelle: Eigener Screenshot.

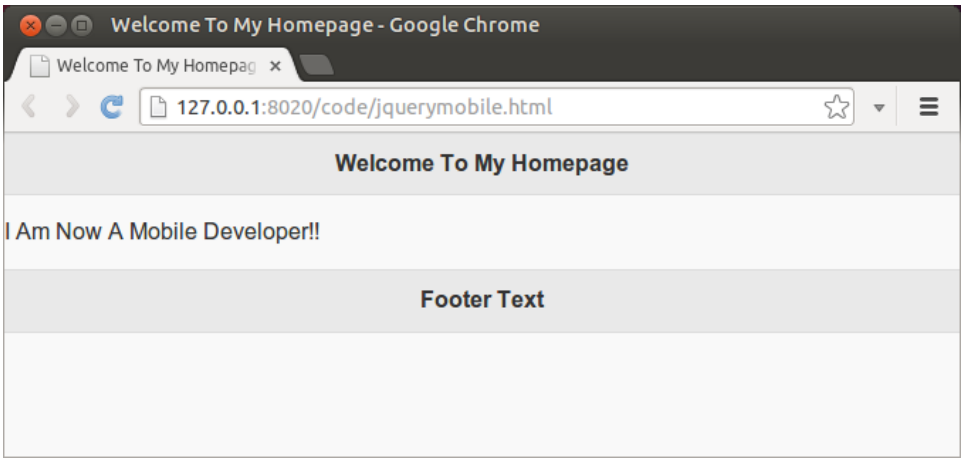

Abbildung 2.5: Beispiel-Oberfläche wie in Abbildung 2.4, ohne das class-Attribut in Listing 2.3, Zeile 12.
Quelle: Eigener Screenshot.

2.2.4 Phonegap / Cordova

2.2.4.1 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*

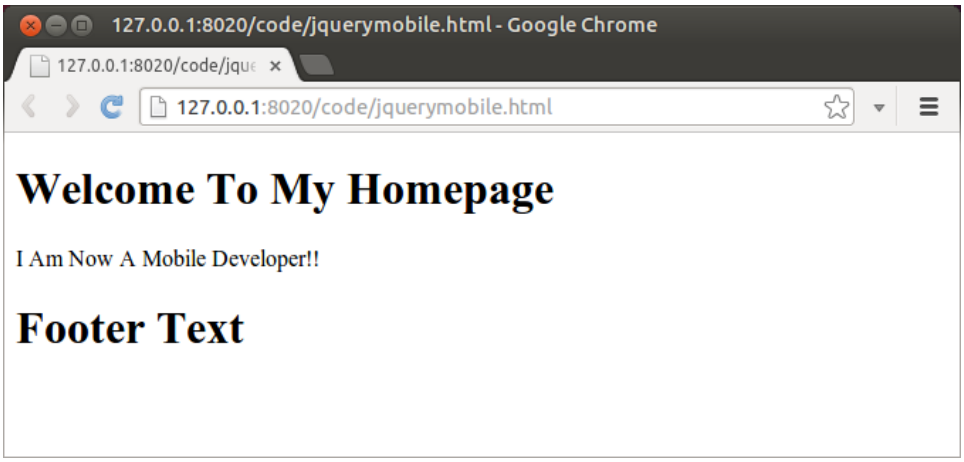

Abbildung 2.6: Beispiel-Oberfläche wie in Abbildung 2.4, aber ohne die jQuery Mobile-Bibliotheken.

Quelle: Eigener Screenshot.

entwickelt, die 2011 von Adobe aufgekauft wurde [13]. Später wurde die Code-Basis der *Apache 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 [24].

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 [24], wird im Folgenden weitgehend der Begriff „Cordova“ verwendet, die meisten Inhalte treffen aber ebenso auf die Adobe-Version PhoneGap zu.

Wie in Unterabschnitt 2.1.3 beschrieben, wird bei der hybriden App-Entwicklung eine Web-App programmiert, die dann in eine native WebView „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 Anpassung 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 [4].

In der Cordova-Dokumentation werden zwei grundlegende Entwicklungsszenarien 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 Entwicklungsworkflows, 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 [5].

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 [5].

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 2.7).

	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 2.7: 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 OS.
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 [6].

2.2.4.2 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 2.5).

Listing 2.5: 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 [5].

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 2.8). Auf oberster Ebene liegt die Konfigurationsdatei config.xml, in der grundlegende Eigenschaften sowie Informationen für den Build-Prozess hinterlegt werden können (siehe Listing 2.6). 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 [5].

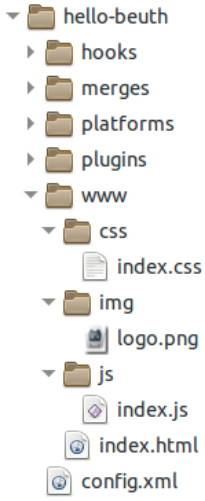

Abbildung 2.8: Dateistruktur einer mit Cordova erstellten App. Das Grundgerüst für die Hybrid-App wird automatisch angelegt und entspricht den üblichen Web-Entwicklungskonventionen.

Quelle: Eigener Screenshot.

Listing 2.6: Die Konfigurationsdatei config.xml, in der allgemeine Informationen über die Hybrid-App gespeichert werden. In Zeile 2 und 4 tauchen die in Listing 2.5 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 2.7, Zeile 21 und Listing 2.8, 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 2.7: 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 2.8: Standardmäßig von Cordova angelegte JavaScript-Datei index.js

```
var app = {
  // Application Constructor
  initialize : function() {
    this.bindEvents();
  },
  // Bind Event Listeners
  //
  // Bind any events that are required on startup. Common events are:
  // 'load', 'deviceready', 'offline', and 'online'.
  bindEvents : function() {
    document.addEventListener('deviceready', this.onDeviceReady, false);
  },
  // deviceready Event Handler
  //
  // The scope of 'this' is the event. In order to call the 'receivedEvent'
  // function, we must explicity call 'app.receivedEvent(...);'
  onDeviceReady : function() {
    app.receivedEvent('deviceready');
  },
  // Update DOM on a Received Event
  receivedEvent : function(id) {
    var parentElement = document.getElementById(id);
    var listeningElement = parentElement.querySelector('.listening');
    var receivedElement = parentElement.querySelector('.received');

    listeningElement.setAttribute('style', 'display:none;');
    receivedElement.setAttribute('style', 'display:block;');

    console.log('Received Event: ' + id);
  }
};
```

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 2.9). 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 2.10) [5].

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 wer-

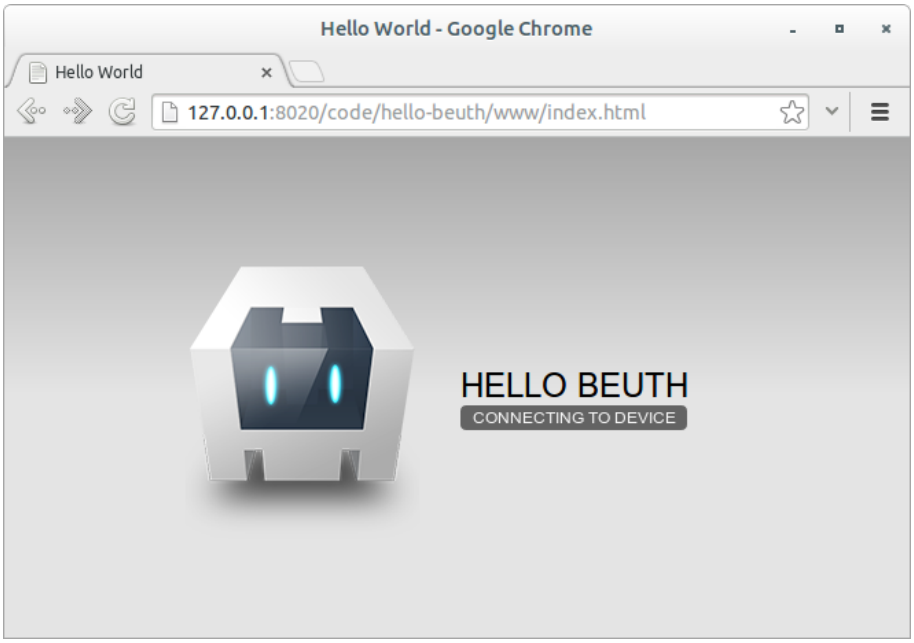

Abbildung 2.9: Die Startseite der Beispiel-App aus Listing 2.7 lässt sich auch im Desktop-Browser öffnen und anzeigen, allerdings kann hier kein `deviceready`-Event empfangen werden.

Quelle: Eigener Screenshot.

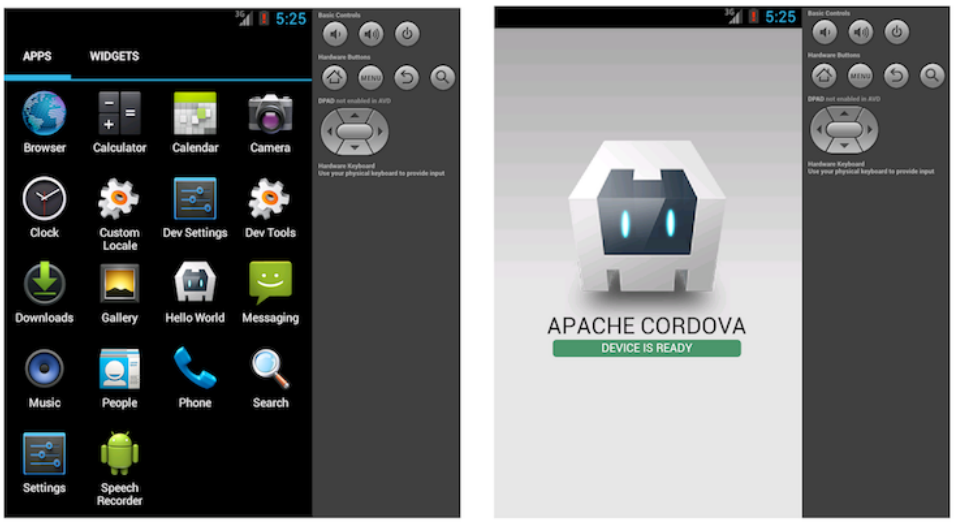

Abbildung 2.10: Anzeige in der Apps-Übersicht (links) und Ausführung der Cordova-Beispiel-App in einem Android-Emulator (rechts). Das grüne Label zeigt den Empfang des `deviceready`-Events an.

Quelle: Cordova-Dokumentation [1]

den, 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 [5].

Über den Cordova-Befehl `platform` lassen sich mit den Optionen `add` und `remove` Plattformen zur Projektkonfiguration der Anwendung hinzufügen bzw. entfernen (siehe Listing 2.9). Die Ausführung dieser Befehle wirkt sich auf den Inhalt des platforms-Ordners innerhalb der Projektstruktur aus [5].

Listing 2.9: 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 2.10).

Listing 2.10: Cordova-Skript zum Auflisten aller Plattformen des Projekts.

```
1 $ cordova platforms list
```

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

Listing 2.11: 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 2.12). 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.[12]

[12] vgl. Nativ-Entwicklungsansatz, siehe oben.

Listing 2.12: Ausführlichere Schreibweise für den `build`-Befehl aus Listing 2.11.

```
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 2.13) oder direkt auf einem an den Entwicklungsrechner angeschlossenen Mobilgerät ausgeführt werden (siehe Listing 2.14). Für letzteres müssen unter Umständen noch entsprechende Einstellungen auf dem Gerät vorgenommen werden [5].

Listing 2.13: Übergibt die App an den Emulator des Android-SDKs.

```
1 $ cordova emulate android
```

Listing 2.14: Führt die App auf einem angeschlossenen Mobilgerät aus.

```
1 $ cordova run android
```

2.2.4.3 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. Cordova bietet von Haus aus eine ganze Reihe von Plugins an, es können aber auch eigene entwickelt werden.[13] Plugins bestehen aus einem zusätzlichen Stück Code, der die Kommunikation zu den nativen Komponenten herstellt.

Einrichten von Plugins: Das Cordova-CLI bietet auch die Möglichkeit, alle verfügbaren Plugins zu durchsuchen, hinzuzufügen, aufzulisten und zu entfernen (siehe Listing 2.15, 2.16, 2.17 und 2.18).

Listing 2.15: Suchen von verfügbaren Plugins.

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

[13] 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 [5].

Listing 2.16: Plugin zur App hinzufügen.

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

Listing 2.17: Analog zum `list`-Befehl für Plattformen (siehe Listing 2.10) können auch Plugins des aktuellen Projekts aufgelistet werden.

```
1 $ cordova plugin ls
```

Listing 2.18: Plugin entfernen.

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

Verwendung von Plugins:

2.2.5 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 vorherigen[14] 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 Unterabschnitt 2.1.3). Dabei kann unter anderem besonderer Mehraufwand auf der administrativen Ebene entstehen, da, wie im Unterabschnitt 2.2.4 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. Aller-

[14] Also der Entwicklung des Codes und der Oberfläche sowie Tests.

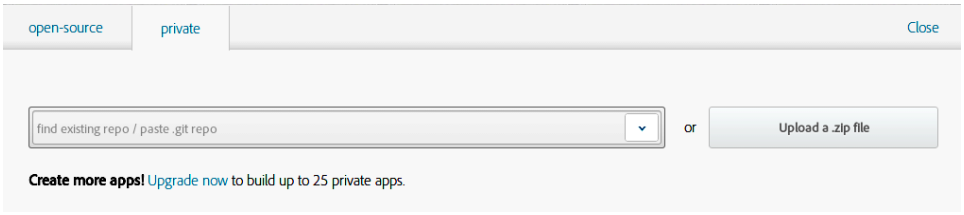

Abbildung 2.11: Dialog für die Erstellung einer neuen privaten App auf PhoneGap Build. Links das Eingabefeld zum Eintragen eines Git-Repository-Links und rechts der Button zum Hochladen von Zip-Archiven.

Quelle: Eigener Screenshot.

dings bietet auch PhoneGap Build nicht für sämtliche mobilen Plattformen, die von Cordova / PhoneGap unterstützt werden, seinen Service an. Während bis Version 2.9 noch Apps für iOS, Android, Windows Phone, Blackberry OS, *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 [11].

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*-Repository[15] in PhoneGap Build eingestellt werden können (siehe Abbildung 2.11), 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“ [14]*

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 2.12).

[15] Beispielsweise wie die öffentlichen Apps auf *GitHub* gehosted oder einem eigenen Git-Repository

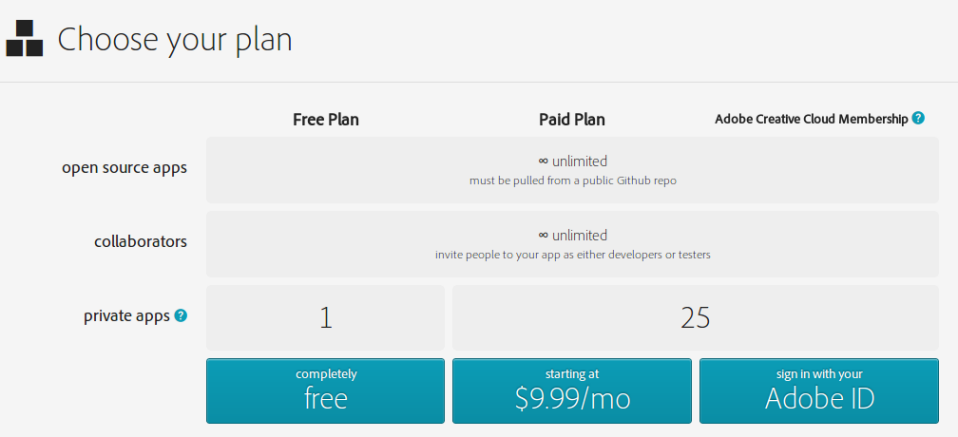

Abbildung 2.12: Übersicht über die verschiedenen Bezahlpakete von PhoneGap Build: Ab 9,99 $ im Monat können Entwickler bis zu 25 private Apps anlegen, in der kostenlosen Variante nur eine private, aber unbegrenzt öffentliche.

Quelle: Eigener Screenshot [15].

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 2.2.4) [12]. 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 2.13 und 2.14).

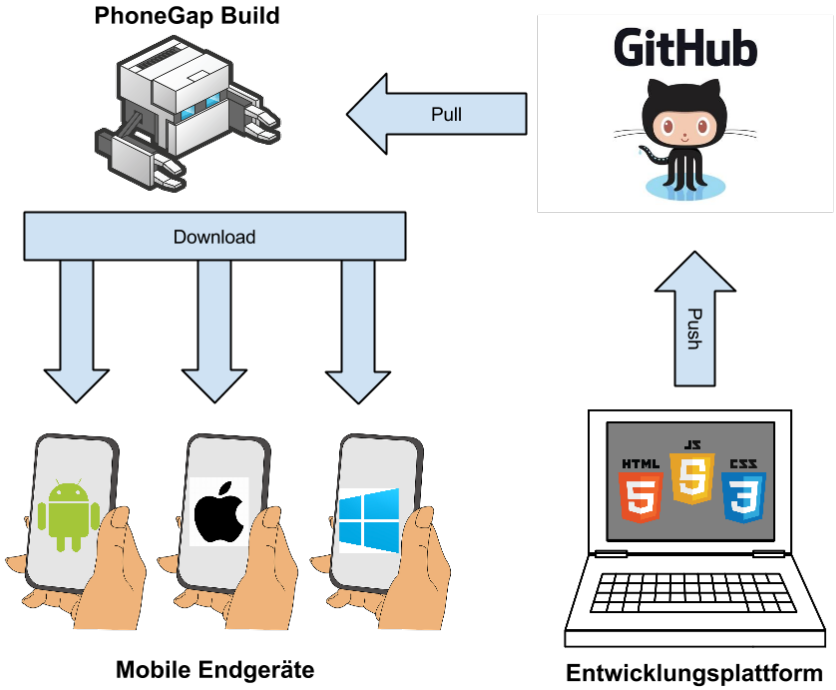

Abbildung 2.13: Schematische Darstellung eines möglichen Entwicklungsworkflows mit PhoneGap Build: Der Code wird als Web-Anwendung auf GitHub hochgeladen, bei PhoneGap Build über das GitHub-Repository aktualisiert und für die jeweiligen Plattformen gebaut, sodass die plattformspezifischen Apps heruntergeladen und auf den Zielgeräten installiert werden können.

Quelle: Eigene Grafik.

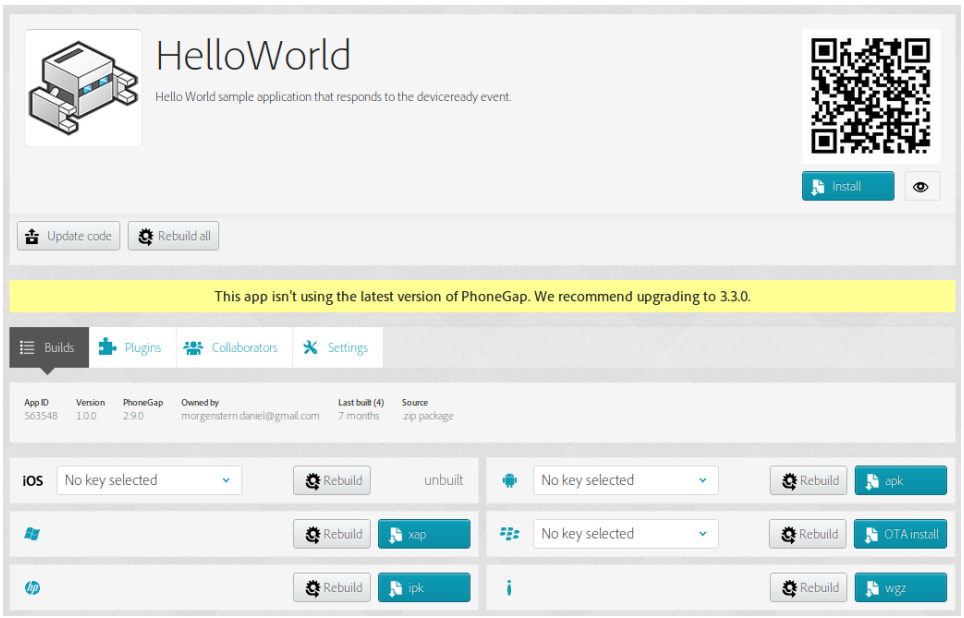

Abbildung 2.14: Oberfläche des PhoneGap Build-Portals: Detailansicht für eine Beispiel-App. Hier können verschiedene Einstellungen vorgenommen werden, sowie der Code aus einem Repository aktualisiert und die fertig gebaute App per Download-Button oder QR-Code heruntergeladen werden.

Quelle: Eigener Screenshot.

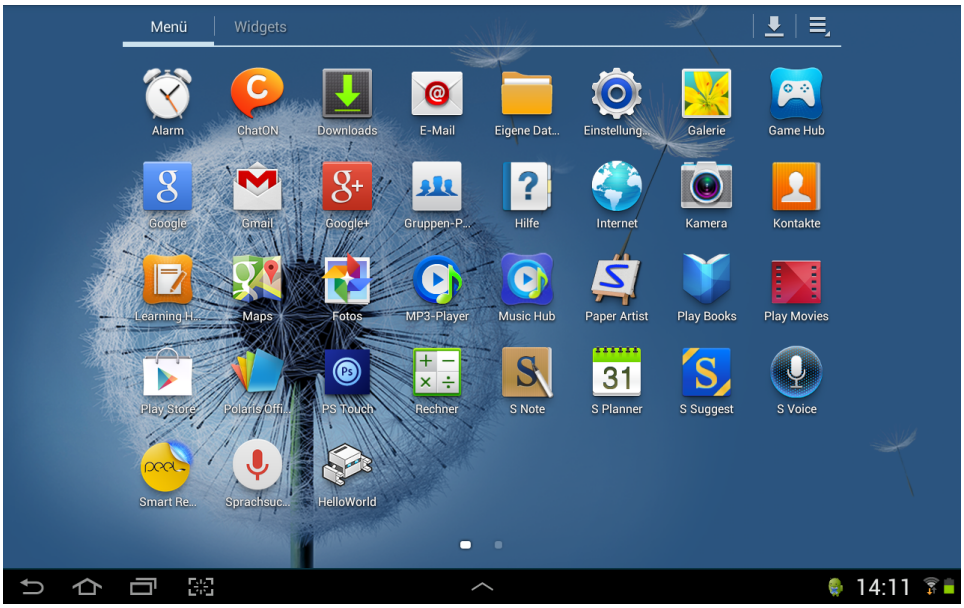

Abbildung 2.15: Die Beispiel-App („HelloWorld“) aus Abbildung 2.14 aus PhoneGap Build auf einem Android-Gerät installiert.

Quelle: Eigener Screenshot.

Kapitel 3

Praktische Umsetzung

3.1 Beispiel-Anwendung

3.1.0.1 Konzeption

3.1.0.2 Grundlegendes / Architektur

Grundlegend besteht die Anwendung aus den drei in **??** abgebildeten Teilen *View* in Form einer HTML-Seite, dem JavaScript-Objekt *Model*, das den Zugriff auf das Gerät bewerkstelligt und dessen Daten bereitstellt und dem *ViewModel*, welches die Eigenschaften und anzuzeigenden Daten der View beinhaltet. Das JavaScript-Framework Knockout hält dabei durch das Data-Binding die Anzeige der Daten auf der View mit dem Datenmodell im ViewModel synchron (siehe Unterabschnitt 2.2.2).

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.

3.2 Kriterien zur Bewertung

3.3 Umsetzung

3.3.1 Ausgewählte Technologie

3.3.2 Exploraion: Implementierung der Geräte-Schnittstelle

3.3.2.1 Zugriff auf die Kontaktverwaltung des Geräts

Im Anwendungsfall *Liste Teilen* der in Kapitel 3 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 2.2.4 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 [7].

Um Kontakte im Adressbuch zu suchen, muss im Wesentlichen die Methode `navigator.contacts.find(fields, onSuccess, onError, options)` ausgeführt werden, wobei `fields` die anzuzeigenden 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 3.1, Zeile 26).

Da in diesem Beispiel (Listing 3.1) keine bestimmten, sondern *alle* Kontakte angezeigt werden sollen, wird der `find`-Methode kein spezieller `filter` übergeben (Zeile 23). In der Oberfläche soll der Name verfügbar sein, sodass der `fields`-Parameter die entsprechenden Felder als Array beinhaltet (Zeile 25). Bei erfolgreicher Ausführung der `find`-Methode werden die zurückgegebenen Kontaktdaten als Parameter in Form eines Arrays, das die entspre-

chenden JavaScript-Objekte vom Typ `Contact` beinhaltet, an die `onSuccess` -Methode übergeben und vom Model an dessen Observers (in diesem Fall das ViewModel) versendet (Zeile 13).

Listing 3.1: 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 = [ "name" ];
26   navigator.contacts.find(fields, onSuccess, onError, options);
27
28 }
```

Die in Listing 3.1 per Observer-Pattern versendeten Daten werden in der entsprechenden Handler-Methode an die `contacts`-Eigenschaft des View-Models übergeben (Listing 3.2, Zeilen 16 - 18), deren Elemente (also die Kontakt-Objekte) anschließend durch das Data-Binding in der Oberfläche angezeigt werden (siehe unten).

Listing 3.2: 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 Kontaktdaten 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 3.3, Zeile 3). Durch die Bindung der Eigenschaft `contacts` in Form eines *Observable Arrays* des ViewModels (Listing 3.2, Zeile 6) an die *Listview* der Oberfläche werden in dieser Liste alle Kontakte des Adressbuchs angezeigt (siehe Listing 3.3, Zeile 7).

Listing 3.3: UI-Komponente zur Darstellung der Kontaktliste und Auswahl einzelner Kontakte.

```
<div data-role="collapsible" data-expanded-icon="carat-d"
     data-collapsed-icon="plus" data-theme="b"
    data-bind="jqmRefreshList: $root.contacts,
    event: { 'collapsibleexpand': $root.getContacts }">
  <h4>Mitglieder hinzufügen</h4>
  <ul data-role="listview"  data-theme="b"
    data-bind="foreach: $root.contacts,
    jqmRefreshList: $root.contacts">
    <li>
      <a href="#"
        data-bind="text: name.formatted,
        click: $parent.addMember,
        css: $parent.memberStatus($data),
        style: {
          'background-color':
            $parent.isMember($data) ? '#aed66f' : '#333'
          }">
      </a>
    </li>
  </ul>
</div>
```

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 OS, *Firefox OS*, iOS, Windows Phone und *Microsoft Windows 8* eine relativ breite Kompatibilität für alle größeren gängigen und mobilen Betriebssysteme. Auch werden einige Datentypen nicht von allen Plattformen unterstützt, so zum Beispiel das `ContactOrganization`-Objekt, das lediglich bei der Entwicklung für Android, Blackberry OS, Firefox OS, iOS und Windows Phone zur Verfügung steht, damit aber immer noch alle großen Plattformen unterstützt [7].

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 `Contacts` -Objekts unter iOS und Windows Phone nicht verfügbar und muss in diesem Fall durch das `name`- oder `nickname`-Feld ersetzt werden. Ähnliches gilt auch

für weitere Objekte oder deren Methoden und Felder, die unter bestimmten Plattformen nicht oder nur teilweise unter Berücksichtigung bestimmter Besonderheiten funktionieren [7].

Während bei der nativen App-Entwicklung für iOS oder Android UI-Komponenten für die Arbeit mit Kontakten bereitstehen, liefert die Cordova-API hier lediglich Mechanismen für den Zugriff auf die Kontaktdaten, 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 2.2.2).

Grundsätzlich bietet die Cordova-Plugins-API trotz einiger Einschränkungen (siehe oben) ein nützliches Werkzeug mit einem breiten Funktionsumfang für den grundlegenden Zugriff auf die Kontaktverwaltung aller großen mobilen Betriebssysteme.

3.4 Fazit

Kapitel 4

Resümee / Ausblick

Glossar

Abbildungsverzeichnis

Quellenverzeichnis

Bildquellen

[1] URL: http://cordova.apache.org/docs/en/3.4.0/img/guide/cli/android_emulate_install.png (siehe S. 24).

[2] *MVVMPattern*. Wikimedia Commons. URL: http://commons.wikimedia.org/wiki/File:MVVMPattern.png (besucht am 13. 04. 2014) (siehe S. 11).

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. 3).

[4] *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. 18).

[5] *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. 18, 20, 23, 25, 26).

[6] *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. 19).

[7] *Cordova Plugin Registry*. Apache Software Foundation. 2014. Kap. org.apache.cordova.contacts. URL: http://plugins.cordova.io/#/package/org.apache.cordova.contacts (besucht am 15. 05. 2014) (siehe S. 34, 37, 38).

[8] Ian Hickson. *Web SQL Database*. W3C. 18. Nov. 2010. URL: http://www.w3.org/TR/2010/NOTE-webdatabase-20101118/ (siehe S. 6).

[9] Ian Hickson. *Web Storage*. W3C. 2013. URL: http://www.w3.org/TR/2013/REC-webstorage-20130730/ (siehe S. 6).

[10] Nikunj Mehta u. a. *Indexed Database API*. W3C. Juli 2013. URL: http://www.w3.org/TR/2013/CR-IndexedDB-20130704/ (siehe S. 6).

[11] *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. 28).

[12] *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. 29).

Online-Quellen

[13] *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. 17).

[14] *Adobe PhoneGap Build*. URL: https://build.phonegap.com/apps (besucht am 14. 04. 2014) (siehe S. 28).

[15] *Choose your plan*. URL: https://build.phonegap.com/plans (besucht am 15. 04. 2014) (siehe S. 29).

[16] 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. 3).

[17] *iOS Developer Program*. Apple Inc. URL: https://developer.apple.com/programs/ios/ (besucht am 28. 04. 2014) (siehe S. 3).

[18] *jQuery Examples*. URL: http://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_hide_p (besucht am 11. 04. 2014) (siehe S. 10).

[19] *jQuery Examples*. URL: http://www.w3schools.com/jquerymobile/tryit.asp?filename=tryjqmob_start (besucht am 13. 04. 2014) (siehe S. 15).

[20] *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. 5).

[21] 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. 2).

[22] *Model View ViewModel*. URL: http://de.wikipedia.org/wiki/MVVM (besucht am 13. 04. 2014) (siehe S. 11).

[23] *Offline - HTML5 Rocks*. Google Inc. URL: http://www.html5rocks.com/de/features/storage (siehe S. 6).

[24] PhoneGap Blog. *PhoneGap, Cordova, and what's in a name?* 19. März 2012. URL: http://phonegap.com/2012/03/19/phonegap-cordova-and-what%C3%A2%C2%80%C2%99s-in-a-name/ (besucht am 15. 04. 2014) (siehe S. 17).

[25] 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. 5).

[26] *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. 5).