FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelorarbeit in Wirtschaftsinformatik

Evaluierung der Umsetzung nativer mobiler Anwendungen im Vergleich zu webbasierten Technologien

Christoph Mahlert

FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelorarbeit in Wirtschaftsinformatik

Evaluierung der Umsetzung nativer mobiler Anwendungen im Vergleich zu webbasierten Technologien

Evaluation of implementing native mobile Apps in comparison to web-based technologies

Bearbeiter:	Christoph Mahlert
Aufgabensteller:	Univ.-Prof. Dr. Uwe Baumgarten
Betreuer:	Dipl.-Ing. (Univ.) Andreas Bernhofer
Abgabedatum:	12.10.2011

Erklärung

Ich versichere, dass ich diese Bachelorarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Starnberg, den 12.10.2011 _____

1 Inhaltsverzeichnis

2 Motivation

Viele Unternehmen möchten ihre Dienstleistungen von überall aus und zu jeder Zeit anbieten, weshalb mobile Anwendungen in der heutigen Zeit eine große Rolle spielen. Die große Verbreitung von Smartphones, insbesondere Android-Smartphones, bietet diesen Unternehmen eine Plattform, ihre mobilen Anwendungen komfortabel an die Nutzer zu bringen. Auch unternehmensintern lassen sich mit Hilfe mobiler Anwendungen viele Prozesse und Kommunikationswege vereinfachen und beschleunigen.

Da Unternehmen in erster Linie kosteneffizient und qualitativ arbeiten möchten, stellt sich die Frage, wie genau diese mobilen Anwendungen umgesetzt bzw. entwickelt werden sollten. Auf der einen Seite gibt es die Möglichkeit native Anwendungen mit dem Native Development Kit (NDK) oder dem Software Development Kit (SDK) von Android zu erstellen. Auf der anderen Seite kann man auch webbasierte Technologien wie z.B. HTML5 und Javascript einsetzen. Möchte man eine mobile Anwendung schreiben, muss man also die Frage klären, welche Vorgehensweise am besten ist. Diese Frage stellt sich als eine Herausforderung dar, weil es viele Kriterien gibt, die die Antwort beeinflussen können. Angefangen bei persönlichen Fähigkeiten des Programmierers, Wünsche der Anwender, bis hin zu technischen Details wie Konnektivität, Sicherheit oder Persistenz der Daten oder aber die Portierung auf oder von anderen Systemen gibt es viele Einflussfaktoren, die Zeit, Kosten und Qualität der Entwicklung verändern.

Meine Bachelorarbeit wird u.a. die genannten Einflussfaktoren beschreiben und untersuchen. Zudem sollen weitere Kriterien gefunden werden, die für die Entscheidung der Umsetzung eine Rolle spielen. Dies soll auch an praktischen Beispielen erläutert werden, so dass durch geeignetes Instrumentieren eines Android-Smartphones bewertet werden kann, wie groß die Vor- und Nachteile der jeweiligen Methoden bzw. Techniken sind und was deren Ursache ist. Letztendlich soll die Arbeit eine Entscheidungshilfe bei der Wahl der Umsetzung geben.

Des Weiteren besteht auch ein betriebswirtschaftlicher Bezug, da die Verbreitung und Verwendung sogenannter *Apps* eine immer wichtigere Position einnimmt. Da diese Apps für ein Unternehmen keinen Selbstzweck darstellen sollen, ist es wichtig, die Entwicklung so zu gestalten, dass Zeit, Kosten, Qualität und viele weitere Faktoren berücksichtigt werden, um so den Nutzen für die Anwender und somit letztendlich den Unternehmenserfolg zu maximieren.

3 Einleitung

Das Thema „*Evaluierung der Umsetzung nativer mobiler Anwendungen im Vergleich zu webbasierten Technologien*“ ist eine Problematik in einer Zeit, in der die rechnergestützte Informationsverarbeitung allgegenwärtig ist und mobile Endgeräte daher zunehmend an Bedeutung gewinnen. Aus Entwicklersicht laufen auf all diesen Endgeräten die unterschiedlichsten Betriebssysteme wie z.B. Android, iOS, Bada, Symbian, Windows Phone oder RIM OS, so dass eine Applikation oftmals portiert werden muss, was einem hohen Aufwand entspricht. Es stellt sich die Frage, ob es für einen Entwickler Sinn ergibt, jede native Programmiersprache aller Betriebssysteme, auf denen die Applikation laufen soll, zu erlernen. Da die mobilen Endgeräte in der heutigen Zeit sehr gut vernetzt und zudem über leistungsfähige Hardware verfügen und Internettarife boomen [Pew11], kommen zunehmend auch webbasierte Techniken für die Entwicklung von Anwendungen in Betracht, die entweder nur mit geringem Aufwand portiert oder im Idealfall völlig plattformunabhängig agieren und gar nicht erst portiert werden müssen, um auf den Zielplattformen zu laufen.

Um die Vor- und Nachteile zwischen einer nativen und einer webbasierten Entwicklung zu evaluieren, könnte man ein beliebiges der genannten mobilen Betriebssysteme verwenden. Da Android im Jahre 2011 allerdings eine Vormachtstellung mit einem Marktanteil von fast 50 Prozent einnimmt [Can15], sollen die Vor- und Nachteile mit Hilfe dieses Betriebssystems erörtert werden.

Um das eingangs erwähnte Thema anhand von Android evaluieren zu können, ist es wichtig, dass man auch die Android Grundlagen versteht. Daher wird sich das folgende Kapitel mit den Grundlagen von Android beschäftigen, bevor auf die native Entwicklung eingegangen wird. Das zugrunde liegende Betriebssystem von Android ist eine Version von Linux (siehe 4.1.5), daher wäre die eigentliche native Programmiersprache C / C++. Google selbst empfiehlt allerdings die Programmierung in Java und nicht in C / C++ [Goo112]. Zwar stellt Google ein Native Development Kit (NDK) bereit, mit welchem man in C / C++ programmieren kann, allerdings ist die Programmierung mit dem NDK eher als Ausnahme zu verstehen und sollte nur in Ergänzung zum Software Development Kit (SDK) genutzt werden [Goo111]. Das SDK und die Architektur von Android und insbesondere die Android Laufzeitumgebung (siehe 4.1.4) zielen auf Java ab. Unter einer nativen Entwicklung versteht man die Entwicklung von Applikationen in einer plattform-spezifischen Sprache, welche dann von einem Compiler für diese Plattform optimiert wird. Fertig gestellte Applikationen müssen auf dem Endgerät installiert werden. Da die Architektur von Android auf Java ausgerichtet ist, kann man deshalb auch Java in diesem Zusammenhang als eine native Sprache unter Android bezeichnen. Deshalb wird im Kapitel der nativen Entwicklung das SDK eine herausragende Rolle spielen, wenngleich das NDK zumindest erwähnt werden soll.

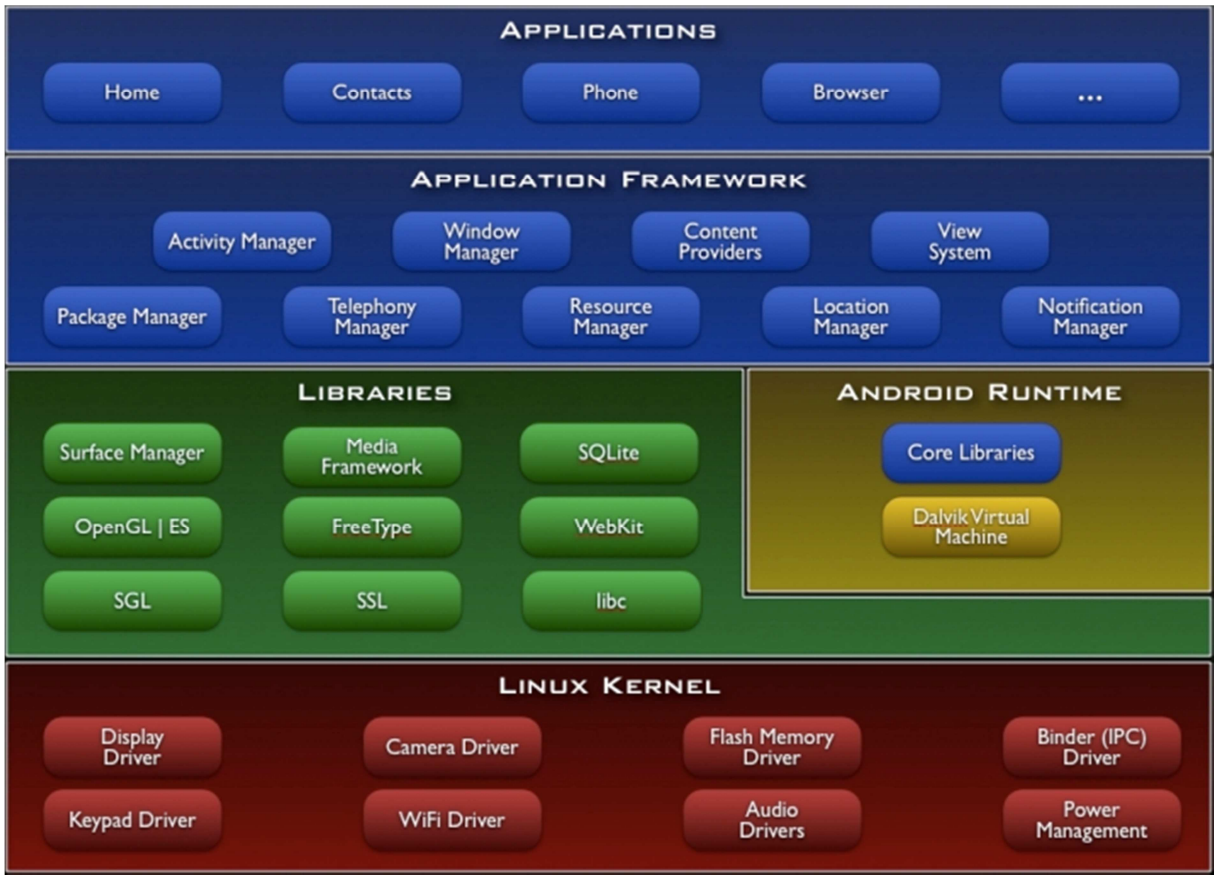
Bei den webbasierten Techniken gibt es mehrere unterschiedliche Techniken. Auch die Grundlagen der webbasierten Techniken finden sich in einem extra Kapitel, bevor auf die eigentliche Entwicklung eingegangen wird. Die Arbeit wird auf mobile Webseiten, sowie auf Web-Applikationen eingehen. Die Definition und die Abgrenzung dieser Begrifflichkeiten befinden sich in Kapitel 7 dieser Arbeit.

Zuerst soll allerdings mit den Android Grundlagen angefangen werden, da diese zum einen zum Verständnis der nativen Entwicklung beitragen, später allerdings auch die Möglichkeiten und Grenzen der webbasierten Techniken aufzeigen.

4 Android Grundlagen

4.1 Architekturübersicht von Android

Der in Abbildung 1 gezeigte Android Software Stack kann in fünf Hauptkomponenten geteilt werden. Im Folgenden wird auf die einzelnen Komponenten eingegangen.



Quelle: http://developer.android.com/guide/basics/what-is-android.html

Abbildung 1: Architekturübersicht von Android Applications

4.1.1 Applications

Auf der obersten Schicht befinden sich einige Anwendungen, mit denen Android ausgeliefert wird. Zu diesen Anwendungen gehören eine Kontakte-, eine E-Mail-, eine Kalender, eine SMS- und eine Browseranwendung. Alle Anwendungen wurden mit Hilfe von Java entwickelt. Des Weiteren befinden sich alle Applikationen auf dieser Schicht, die mit dem Android SDK entwickelt wurden.

4.1.2 Application Framework

Der Anwendungsrahmen abstrahiert die zugrunde liegende Hardware und enthält die Android-spezifischen Klassen. Er ist für einen Entwickler die interessanteste Schicht in der Architektur, da verschiedene Programmierschnittstellen bereitgestellt werden, die sowohl eine Kommunikation zwischen einzelnen Anwendungen, als auch zwischen Anwender und Applikation ermöglichen.

Der Anwendungsrahmen beinhaltet den **Activity Manager,** der die Anwendungen und deren Lebenszyklus, welcher im Kapitel 4.2.1 genauer erläutert wird, steuert.

Ein **Content Provider** bietet die Möglichkeit zum Austausch und den Zugriff auf Daten von anderen Anwendungen. Content Provider stellen eine abstrakte Sicht auf die Daten dar.

Damit ein Anwender mit einer Applikation interagieren kann, werden **Views** benötigt. Listen, Textboxen, Grids, Buttons und eingebettete Webbrowser gehören zu den Standardansichten von Android. Diese können auch miteinander kombiniert werden, um komplexere Ansichten zu erstellen.

Soll eine Anwendung Benachrichtigungen in der Statusbar anzeigen, muss der **Notification Manager** benutzt werden. Wenn eine Anwendung auf eine Benachrichtigung reagieren soll, kann an dieser ein *Intent* (siehe 4.3) hinterlegt werden.

Der **Resource Manager** ermöglicht es auf externe Dateien wie z.B. Grafiken zuzugreifen, die in die Anwendung einkompiliert wurden. Android unterteilt die Dateien in XML-Dateien, Bilddateien und sonstige Rohdateien.

4.1.3 Libraries

Die von Android-Anwendungen benötigten Funktionalitäten werden von einer Reihe von C/C++ Bibliotheken bereit- und dem Entwickler durch den Anwendungsrahmen zur Verfügung gestellt.

Die **System C library** ist eine Implementierung von der Berkeley Software Distribution (BSD) der Standard C-Bibliotheken (libc). Sie wurde für Linux-basierte eingebettete Systeme optimiert.

Die **Media Libraries** basieren auf dem quelloffenen Produkt *OpenCore* der Firma *PacketVideo*. Sie unterstützen die Wiedergabe und Aufnahme populärer Musik- und Videoformate und statischer Bilder. Unter anderem werden die die Formate MPEG4, H.264, MP3, AAC, AMR, JPG und PNG unterstützt.

Der **Surface Manager** verwaltet die Anzeige auf das Anzeigeuntersystem. Er kann 2D und 3D Grafikschichten aus mehreren Anwendungen vermischen, wozu er OpenGL ES und einen 2D-Hardwarebeschleuniger nutzen kann.

Die **LibWebCore** ist eine auf der quelloffenen Bibliothek „WebKit“ basierende Webbrowser-Umgebung. Dies bringt in Hinblick auf die webbasierte Anwendungsentwicklung viele Vorteile mit sich, da WebKit die Grundlage vieler Browser ist: Googles Chrome, Apples Safari, Nokias S60-Browser und der Blackberry Browser seit OS 6. Eine für WebKit optimierte einfache webbasierte Anwendung läuft somit in der Theorie ohne viel Portierungsaufwand auf den gängigsten mobilen Geräten, sowie durch Chrome und Safari auch auf einem Desktop-Rechner.

Die **SGL** (Scalable Graphics Library) ist die 2D-Grafik-Engine.

Die **3D libraries** basieren auf der OpenGL ES 1.0 API. Ist ein 3D-Hardwarebeschleuniger vorhanden, wird dieser verwendet.

FreeType ist ein Bitmap- und Vektorschrift Renderer.

Die **SQLite** kommt als relationales Datenbanksystem zum Einsatz. Sie steht allen Anwendungen zur Verfügung.

Die Standardbibliotheken sind ein fester Bestandteil des Systems und können von einem Anwendungsentwickler nicht modifiziert werden.

4.1.4 Android Runtime

Die Laufzeitumgebung von Android beinhaltet eine Reihe von Kernbibliotheken. Diese stellen die meisten Funktionen zur Verfügung, die auch in den Kernbibliotheken von Java zur Verfügung gestellt werden.

Die Dalvik Virtuelle Maschine (DVM) bildet den Kern der Laufzeitumgebung von Android. Sie basiert auf der quelloffenen „Apache Harmony“, wurde aber für die Bedürfnisse mobiler Endgeräte angepasst. Dalvik wurde so entwickelt, dass mehrere virtuelle Maschinen effizient parallel ausgeführt werden können. Dalvik lebt in Symbiose mit dem Linux Rechte-/Zugriffssystem. Jede Android Applikation läuft in einer eigenen virtuellen Maschine und in einem eigenen Betriebssystemprozess. Jeder Prozess läuft zudem unter einem Benutzer, der für die jeweilige Applikation angelegt wurde. Wird die Applikation beendet, dann wird auch der dazugehörige Prozess beendet. Android führt Programme ohne spezielle Berechtigungen in einer Sandbox aus. D.h. eine Applikation darf nur in einem eingeschränkten Bereich laufen. Der Zugriff einer Applikation auf Ressourcen wie Arbeitsspeicher, Telefonfunktion und anderen Anwendungen wird von der Sandbox geregelt. Möchte man die Sandbox verlassen, um auf andere Ressourcen zuzugreifen, müssen explizit Berechtigungen vergeben werden. Dies kostet zwar mehr Ressourcen, allerdings erhöht es auch die Sicherheit, weil die Anwendungen voneinander abgegrenzt sind und sich keinen Speicher teilen.

Dalvik führt Dateien im Dalvik Executable (.dex) Format, welches Dalvik-Bytecode enthält, aus. Ein Entwickler schreibt seine Anwendungen in Java. Auch zur Entwicklungszeit wird der Code von einem Java-Compiler in Java-Bytecode übersetzt. Um eine fertige Applikation in das DEX-Format zu transformieren, kommt das *dx-Tool* zur Anwendung, welches als Cross-Compiler im Android SDK enthalten ist:

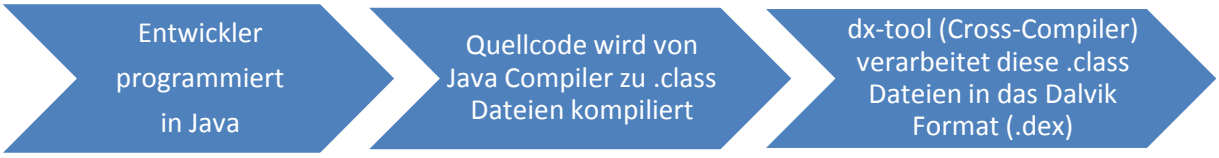


Abbildung 2: Cross-Compiling einer in Java entwickelten Android-Applikation

Der daraus entstandene Dalvik-Bytecode hat nur noch eine entfernte Ähnlichkeit zum Java-Bytecode.

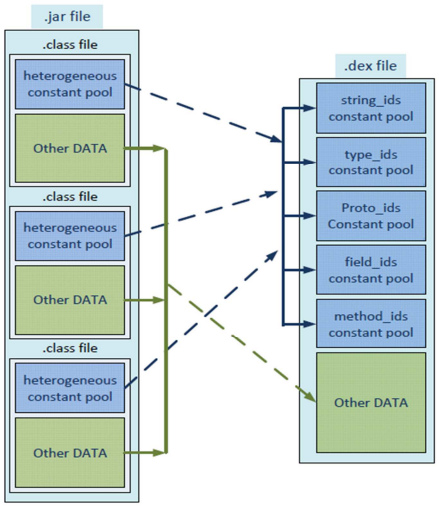


Abbildung 3: JAR- vs. DEX-Format

In der Abbildung 3 ist der Unterschied zwischen dem JAR- und dem DEX-Format zu sehen. Der Hauptunterschied besteht darin, dass in einer JAR-Datei alle Klassen als separate Dateien auftauchen, während in einer DEX-Datei alle Klassen einer Applikation in nur eine Datei gepackt werden. Dies ist nicht nur ein einfaches Packen, sondern es werden redundante Daten entfernt. Typischerweise gibt es in Java viele unterschiedliche Signaturen von Methoden in einer Klasse. Diese Signaturen werden dupliziert, wenn sie von einer anderen Klasse referenziert werden. In einer DEX-Datei hingegen teilen sich alle Klassen dieselben Methoden, Felder usw. In einem Beispiel wurden 11 CLASS-Dateien mit einer Gesamtgröße von 21395 Bytes in das DEX-Format konvertiert. Die DEX-Datei hatte danach eine Größe von 17664 Bytes, was einer Ersparnis von 17% entspricht.

Danach wurden die CLASS-Dateien als JAR gepackt, was eine Dateigröße von 13685 Bytes ergab. Die DEX-Datei ergab gepackt als APK lediglich 9148 Bytes. Das DEX Format kann komprimiert also bis zu 30% effizienter sein als das CLASS-Format, weshalb dieses Format durch die bessere Effizienz deutlich praktischer auf mobilen Endgeräten ist.

Allerdings bekommt der Entwickler in der Regel nichts von der DVM mit, da die Transformation vom Java Bytecode zum Dalvik Bytecode z.B. mit Hilfe des zur Verfügung gestellten Eclipse Plug-Ins (siehe 5.4.1) im Hintergrund abläuft. Das Ergebnis ist schließlich die fertige Anwendung in Form einer APK-Datei. Daher ist einem Entwickler auf dem ersten Blick auch gar nicht bewusst, dass die DVM Unterschiede zur Java Virtuellen Maschine (JVM) aufweist.

Die JVM ist Stack-basiert, d.h. Operationen nehmen Eingabewerte vom Stack und legen das Ergebnis zurück auf den Stack. Die DVM hingegen ist Register-basiert. Dalvik Register verhalten sich wie lokale Variablen. Jede Methode hat einen „frischen Satz“ von Registern, d.h. aufgerufene Methoden beeinflussen nicht die Register aufrufender Methoden. Der Vorteil der DVM gegenüber der JVM liegt auf der Hand, da moderne Prozessoren Registermaschinen sind. Diese Registerarchitekturen sind oft deutlich effizienter, da die CPU über eigene besonders schnell zugreifbare Speicherzellen, die Register, verfügt. Ein Stack hingegen braucht Zugriff auf den Speicher. Dalvik ist daher für die mobilen Bedürfnisse optimiert. Zudem kommt die DVM auch gut mit schwachen Prozessoren zurecht und berücksichtigt den Strombedarf von mobilen Endgeräten.

4.1.5 Linux Kernel

Das zugrunde liegende Betriebssystem von Android ist eine Version von Linux. Die Kernsysteme von Android basieren daher hinsichtlich der Sicherheit, der Speicherverwaltung, der Netzwerkschicht und des Treibermodels auf einen Linux-Kernel in der Version 2.6. Der Kernel dient zudem als Abstraktionsschicht zwischen der Hardware- und den Softwareschichten.

4.2 Android Komponenten

Ziel unter Android ist es, Anwendungen nach einheitlichen Richtlinien zu entwickeln. So kann jede Anwendung beispielsweise ihre Fähigkeiten veröffentlichen, so dass andere Anwendungen diese dann bei Bedarf nutzen können. Dies ermöglicht es einem Entwickler einzelne Komponenten durch eigene zu ersetzen oder fremde Komponenten in die eigene Applikation zu integrieren. Android definiert ein einfaches Komponentenmodell, dessen Grundbausteine im Folgenden erläutert werden.

4.2.1 Activity

Activities dienen zur Darstellung und Verwaltung von Oberflächen. Eine Applikation kann eine Oberfläche haben, muss es aber nicht. Applikationen, die mit einem Anwender interagieren, brauchen aber mindestens eine Activity. Activities bestimmen welche Elemente auf der Oberfläche angezeigt werden und behandeln Eingaben eines Anwenders. Beispielsweise lesen sie Eingabefelder aus und reagieren auf eine Auswahl in einem Menü. Eine Acitivity verwendet hierfür eine oder mehrere Views. Activities sind allerdings für mehr als nur die reine Darstellung von Daten und Formularen zuständig, da sie Komponenten einer Applikation sind.

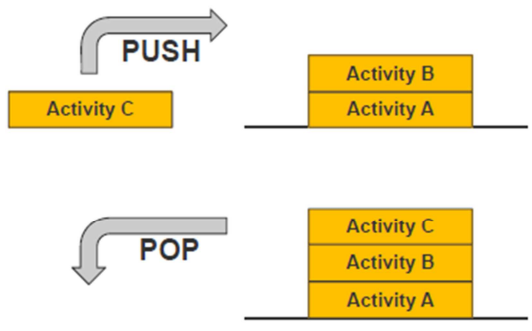


Abbildung 4: Activity Stack

Sie besitzen einen eigenen Lebenszyklus und können ggf. weitere Activities starten. In Abbildung 4 wird erst die Activity C gestartet, wodurch sie auf den Activity Stack als oberstes Element erscheint und auf dem Endgerät im Vordergrund angezeigt wird. Diesen Zustand einer Activity nennt man *running*. Daneben gibt es noch die Zustände *paused* und *stopped.* In beiden Fällen ist die Activity nicht mehr das oberste Element im Stack und kann durch das System automatisch beendet werden. Android beendet Activities dann, wenn der Speicher knapp wird. Activities bleiben als Java-Objekte in der Dalvik Virtuellen Maschine erhalten. Dabei bleiben auch alle Attribute des Java-Objektes erhalten. Ein erneuter Aufruf einer solchen Activity funktioniert dann natürlich schneller, da das Java-Objekt nicht neu erzeugt werden muss, allerdings birgt diese Technik auch ein Risiko, da das Speichern von großen Datenmengen in Objekt-Attributen den Speicher des Endgerätes schnell füllt und somit stark belastet. Es ist daher zu empfehlen, große Datenmengen beispielsweise in eine SQLite-Datenbank zu schreiben und sie ggf. dann temporär bei Bedarf nachzuladen.
Der Unterschied zwischen *stopped* und *paused* besteht darin, dass im Gegensatz zum Zustand *stopped* die Activity im Zustand *paused* für den Anwender noch sichtbar bleibt. Erscheint beispielsweise ein Popup, dann ist die Activity im Zustand *paused* und man kann die Anwendung im Hintergrund noch sehen. Wird aber aus der Anwendung heraus eine weitere Anwendung gestartet, beispielsweise mit Klick auf einem Hyperlink, welcher den Webbrowser öffnet, dann befindet sich die Anwendung im Zustand *stopped* und ist für den Benutzer nicht mehr zu sehen.

4.2.2 Service

Wie bereits erwähnt, muss eine Applikation oder ein Teil einer solchen nicht zwingend eine Oberfläche besitzen. Es gibt Operationen, die ein Anwender nicht sehen muss und daher im Hintergrund asynchron abgearbeitet werden können. Möchte man beispielsweise einen Musikplayer programmieren, so kann die Oberfläche einer Activity überlassen werden. Das eigentliche Abspielen der Musik kann von einem Service erledigt werden. Denn wird die Bedienoberfläche geschlossen, läuft der Service im Hintergrund weiter, wodurch man die Musik weiterhin hören könnte, obwohl man eine andere Activity gestartet hat.

4.2.3 Content Provider

Soll die Möglichkeit zum Laden oder Speichern von Daten genutzt werden, dann verwaltet ein Content Provider diese Daten und abstrahiert die darunterliegende Persistenzschicht. Ein Content Provider ist die empfohlene Art und Weise, um auf gespeicherte Daten zuzugreifen oder um diese freizugeben. Werden Daten innerhalb einer Applikation verwaltet und sollen diese für Applikationen zur Verfügung gestellt werden oder benötigt eine andere Anwendungskomponente Daten von einer anderen Anwendung, sollte dies mit Hilfe eines Content Providers implementiert werden. Sollen fremde Applikationen auf Daten einer anderen Applikation zugreifen können, brauchen sie dafür die Berechtigung der datenhaltenden Applikation. Bevor ein Content Provider genutzt werden kann, muss geklärt werden, welchen Inhalt dieser überhaupt liefert. Dabei unterscheidet Android zwischen *strukturierten Datenbankinhalten,* wo die Schnittstelle über einen Cursor zu den Datenbankdaten hergestellt wird und *Binärdaten,* auf denen mit Hilfe eines Streams zugegriffen wird. Der Zugriff auf die Daten eines Content Providers erfolgt mit Hilfe von Universal Resource Identifier (URIs). Eine *Content-URI* ist nach dem Muster *$scheme://$authority/$dataDescriptor[$id]* aufgebaut [Bec09].

4.2.4 Broadcast Receiver

Es ist sinnvoll, wenn Applikationen die Möglichkeit haben, auf bestimmte Systemereignisse zu reagieren. Dafür ist eine Kommunikation zwischen dem Betriebssystem Android und der einzelnen Applikation notwendig. Android verschickt deshalb Systemnachrichten, z.B. wenn der Akku schwach wird oder bei einem einkommen-

den Telefongespräch. Mit Hilfe eines Broadcast Receivers kann eine Applikation auf solche Systemzustandsänderungen reagieren.

4.3 Intent

Ein sich immer wiederholender Begriff für Entwickler unter Android ist das *Intent*. Da Android komponentenbasiert ist (siehe 4.2), müssen diese Komponenten auch verbunden werden. Ein Intent ist ein standardisierter Mechanismus, welcher die Komponenten miteinander verbindet. Er ist deshalb auch als Bindeglied oder auch als „Leim“ zwischen den Komponenten zu verstehen. Durch Intents wird eine lose Kopplung der Komponenten innerhalb einer Applikation erreicht.

Die Applikationen von Android laufen in einer Sandbox (siehe 4.1.4), weshalb eine Kommunikation zwischen zwei Komponenten unterschiedlicher Applikationen nicht möglich ist. Die Sandbox kann nur verlassen werden, wenn auch dementsprechend Berechtigungen definiert wurden. Diesen Berechtigungen sind auch Intents unterworfen.

Es gibt zwei unterschiedliche Arten von Intents: die expliziten und die impliziten Intents. Bei den expliziten Intents ist die Empfängerkomponente beim Aufruf bekannt und eindeutig identifiziert, wohingegen bei einem impliziten Intent der Empfänger unbekannt und es den Komponenten selbst überlassen ist, auf das Intent zu reagieren.

Intents enthalten die Beschreibung einer Absicht oder die Anforderung eines Dienstes. Dies könnte zum Beispiel das Öffnen einer Webseite sein. Aussagekräftige Wörter wie *VIEW, EDIT* oder *PICK* sind in der Regel die Handlungsattribute eines Intents. Die URI *http://in.tum.de* würde beispielsweise mit der Handlung *android.content.Intent.VIEW_ACTION* die Webseite der Informatik Fakultät der Technischen Universität München im Webbrowser von Android öffnen.

Über so genannte *Intent Filter* kann festgelegt werden, welche Activities, Services oder Broadcast Receiver bestimmte Arten von Intents empfangen und verarbeiten können.

4.4 Android Manifest

Bisher wurde auf die einzelnen Komponenten von Android eingegangen. Eine Android Applikation kann aus Activities, Services, Content Provider und Broadcast Receiver bestehen. Diese Komponenten können mit Hilfe von Intent Filter (siehe 4.3) Intents veröffentlichen. Zusätzlich verlangt eine Applikation möglicherweise auch nach bestimmten Zugriffsrechen, um die Sandbox verlassen zu können. Um eine Android-Applikation starten zu können, müssen all diese Informationen im Android Manifest zusammengefasst werden. Die Datei *AndroidManifest.xml*, die alle Aspekte einer Android-Applikation beinhaltet, befindet sich im Wurzelverzeichnis einer Applikation.

Die *AndroidManifest.xml* enthält weiterhin den Namen des Paketes, wodurch die Applikation identifiziert werden kann. Sie spezifiziert die Mindestanforderung an die API Version, so dass ältere Geräte mit einer eventuell veralteten Android-Version ausgeschlossen werden, da sie möglicherweise nicht mehr kompatibel mit der in der entwickelten Applikation verwendeten API-Version sind. Gleichermaßen schränkt sie die unterstützte Hardware und die Displaygröße ein. Auch werden notwendige Bibliotheken aufgelistet.

Die folgende *AndroidManifest.xml* ist dem *ToDo-List*-Beispiel aus dem Kapitel 5.5.1 entnommen:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="de.mahlert.ToDo_Liste"
```

```
    android:versionCode="1"
    android:versionName="1.0">
  <uses-sdk android:minSdkVersion="8" />

        <application android:icon="@drawable/todo" android:label="@string/app_name">
                <activity android:name=".TodosUebersicht" android:label="@string/app_name">
                        <intent-filter>
                                <action android:name="android.intent.action.MAIN" />
                                <category android:name="android.intent.category.LAUNCHER" />
                        </intent-filter>
                </activity>

                <activity android:name=".TodoDetails"
                        android:windowSoftInputMode="stateVisible|adjustResize">
                </activity>
        </application>
</manifest>
```

Erkennbar in diesem Beispiel ist, dass das Paket *de.mahlert.ToDo_Liste* heißt, die Applikation in der Version *1.0* vorliegt und nur auf Geräten läuft, auf denen mindestens Android 2.2 installiert ist, was der *minSdkVersion 8* entspricht. Des Weiteren ist ein Icon definiert, welches in der Liste der installierten Anwendungen angezeigt wird. Wird die *ToDo-List*-App gestartet, wird zuerst die Activity *ToDosUebersicht* aufgerufen. Weiterhin verwendet die Applikation Intents, weshalb dies auch im Manifest registriert ist. In der Activity *ToDoDetails* gibt es Textfelder. Das Attribut *android:windowSoftInputMode* legt das Verhalten der Activity fest, wenn die Fixierung auf einem Textfeld liegt. In diesem Fall bleibt die Activity durch die Anpassung der Größe oberhalb der Touchscreen-Tastatur sichtbar (siehe Abb. 17).

4.5 Hardware

Android Geräte sind typischerweise primär Telefone, wodurch die Möglichkeit gegeben ist, Telefonate zu initiieren und Kurzmitteilungen zu senden und zu empfangen. Dennoch können die Geräte in der Regel noch weitaus mehr leisten. Die Klasse *android.hardware.Sensor* im Android SDK definiert die Arten von Sensoren, die ein Android-Gerät haben kann [Goo09]:

Sensortyp: TYPE_ACCELEROMETER
Beschreibung: *TYPE_ACCELEROMETER* ist ein Beschleunigungssensor in drei Achsen. Die Einheit ist m/s².

Sensortyp: TYPE_GRAVITY
Beschreibung: Der Sensor gibt die Stärke der Gravitation entlang drei Achsen an. Die Einheit ist ebenfalls m/s².

Sensortyp: TYPE_GYROSCOPE
Beschreibung: *TYPE_GYROSCOPE* beschreibt einen Lagesensor.

Sensortyp: TYPE_LIGHT
Beschreibung: Ein Lichtsensor, der das einfallende Licht in Lux beschreibt.

Sensortyp: TYPE_MAGNETIC_FIELD
Beschreibung: Das Erdmagnetfeld wird in drei Achsen definiert. Die Einheit ist Mikrotesla.

Sensortyp: TYPE_ORIENTATION
Beschreibung: Der Sensor beschreibt die Orientierung des Gerätes in Grad entlang drei Achsen.

Sensortyp: TYPE_PRESSURE
Beschreibung: *TYPE_PRESSURE* beschreibt einen Drucksensor.

Sensortyp: TYPE_PROXIMITY
Beschreibung: Der Sensor dient der Entfernungsmessung.

Sensortyp: TYPE_TEMPERATURE
Beschreibung: Mit diesem Sensor lässt sich die Temperatur messen.

Des Weiteren kann in einem Android Gerät auch eine Kamera verbaut sein, auf die ein Entwickler Zugriff besitzt. Auch haben viele Geräte ein GPS-Modul, welches ortungsbasierte Applikationen ermöglicht.

Standard eines jeden Gerätes sind jedoch zu mindestens ein interner Speicher, eine Netzwerkschnittstelle, sowie die Möglichkeit Ton über das Mikrofon aufzunehmen. Viele Geräte bieten zudem die Möglichkeit den internen Speicher durch microSD-Karten zu erweitern.

5 Native Entwicklung

Wie schon in der Einleitung erwähnt, ist das zugrunde liegende Betriebssystem von Android eine Version von Linux (siehe 4.1.5), daher wäre die eigentliche native Programmiersprache C / C++. Google selbst empfiehlt allerdings die Programmierung in Java und nicht in C / C++ [Goo112]. Die Begründung dafür liegt in der im vorherigen Kapitel erläuterten Architektur von Android und insbesondere in der Android Runtime (seihe 4.1.4). Allgemein versteht man unter einer nativen Entwicklung die Entwicklung von Applikationen in einer plattformspezifischen Sprache, welche dann von einem Compiler für diese Plattform optimiert wird. Deshalb ist in diesem Kapitel unter einer nativen Entwicklung auch der Einsatz mit dem Software Development Kit (SDK) gemeint.

5.1 Voraussetzungen

Sowohl das SDK, als auch das Native Development Kit (NDK), können kostenlos von Google heruntergeladen und benutzt werden. Um mit dem SDK programmieren zu können, muss ein Entwickler die objektorientierte Programmiersprache Java beherrschen. Aufgrund eines reduzierten Sprachumfangs ist Java im Vergleich zu anderen objektorientierten Programmiersprachen wie C++ oder C# einfacher. Beispielsweise werden überladene Operatoren und Mehrfachvererbung nicht unterstützt. Durch eine starke Typisierung, einen Garbage Collector und einen Verzicht auf Zeigerarithmetik ist die Wahrscheinlichkeit ungewollter Systemfehler klein. Da Java Multithreading unterstützt, profitiert es auch von modernen Prozessoren mit mehreren Rechenkernen [Wik114]. Java-Programme werden in den verschiedensten Bereichen von Handel, Industrie und Dienstleistung eingesetzt [Ste11]. Durch die große Palette der Einsatzmöglichkeiten lehren viele Universitäten Java schon ab dem ersten Semester im Studiengang Informatik. Auch die TU München, eine weltweit hoch angesehene Universität [TUM111], benutzt Java, um ihren Studenten die Konzepte der Programmierung beizubringen [TUM11]. Die Schlussfolgerung daraus ist, dass es vielen Entwicklern leicht fallen wird, mit der im SDK verwendeten Programmiersprache umzugehen. Anwendungen, die mit dem SDK entwickelt wurden, nutzen spezifische Android APIs und können daher nicht auf andere Systeme genutzt werden. Andersherum ist es auch nicht möglich native Applikationen anderer Systeme auf Android auszuführen.

Die im NDK verwendete Programmiersprache C++ hingegen ist für Anfänger nicht so leicht zu erlernen, worauf auch in C++ Lehrbücher wie dem von Dirk Louis hingewiesen wird [Lou01]. Zum Beispiel unterlaufen mit der in C++ verwendenden Zeigerarithmetik mit höherer Wahrscheinlichkeit schwerwiegende Programmierfehler, die häufig in einem Absturz der Applikation resultieren. C++ ist allerdings eine maschinennahe Sprache und in der Regel sehr effizient [Wik115]. Google empfiehlt daher die Verwendung des NDKs bei leistungskritischen Anwendungen. Gleichwohl weißt Google darauf hin, dass die Verwendung von C++ die Komplexität der Applikation erhöht, wenngleich die bloße Benutzung des NDKs nicht zwingend in einem Leistungsschub mündet, was in der Architektur von Android begründet ist [Goo112]. Ein Vorteil des NDKs ist, dass es die Möglichkeit eröffnet, C++ Bibliotheken einzubinden. Dies macht eine Portierung von und zu anderen Systemen leichter.

5.2 Möglichkeiten

Das SDK erlaubt den Zugriff auf die Hardware eines Gerätes und reicht in der Regel für die Entwicklung einer Applikation aus. Bei leistungskritischen Anwendungen, etwa einem Navigationssystem, können Teile der Anwendung für eine bessere Performanz auch das NDK nutzen. Der Kreativität eines Entwicklers sind bei einer nativen Entwicklung daher keine Grenzen gesetzt. Diese könnten ausschließlich in der Hardware der Geräte liegen. Da Android eine freie Software ist [Wik116] und mehrere Mobiltelefon-Hersteller darauf zurückgreifen

können, liegt das Problem eines Entwicklers weniger im Betriebssystem, sondern vielmehr in der Vielfalt der verwendeten Hardware. Google gibt den Herstellern keine strikte Hardwarevorschrift vor, weshalb die Unternehmen unterschiedliche Chipsätze verbauen können. Möchte ein Entwickler einen großen Benutzerkreis ansprechen, ist es ihm quasi nicht mehr möglich, seine Applikation mit sämtlichen Geräten zu testen. Dies ist ein Nachteil im Vergleich zu Apple, da diese strikte Vorgaben zur verbauten Hardware haben und ihr SDK auf diese optimieren können. Zwar liegt dem Android SDK ein Emulator bei, dennoch kann auch dieser natürlich nicht die eigentlichen Geräte ersetzen, da viele Funktionen, insbesondere die, die auf die Hardware eines Gerätes zugreifen, nur unzulänglich im Emulator getestet werden können.

5.2.1 Android Market

Aber nicht nur technisch, sondern auch inhaltlich gesehen, gibt es seitens Google kaum Einschränkungen. Der Android Market bietet Entwicklern die Möglichkeit, eine Applikation einer breiten Masse zugänglich zu machen, da dieser als App auf jedem Android-Gerät standardmäßig vorinstalliert ist. Der Anwender muss vor der Installation einer Anwendung den Zugriffsrechten zustimmen, die die Applikation zum Laufen benötigt. Ist die Anwendung erst einmal installiert und stellt der Entwickler Updates bereit, werden diese dem Benutzer automatisch über den Android Market angeboten.

Anders als zum Beispiel Apple führt Google für den Android Market eine sehr liberale Politik, bei der es so gut wie keine Restriktionen gibt. Dies erkennt man beispielsweise auch daran, dass im Android Market sogar Apps mit pornografischem Inhalt erlaubt sind [Fin11].

5.3 Support und Dokumentation

Entscheidet man sich für eine native Entwicklung, ist man auf die Dokumentation von Google angewiesen. Diese ist nicht auf Deutsch, sondern lediglich auf Englisch vorhanden. Google nimmt den Entwickler dabei an die Hand und erklärt Schritt für Schritt, wie eine Entwicklungsumgebung, wobei sich Google dabei auf Eclipse beschränkt, für unterschiedliche Betriebssysteme installiert wird. Anschließend werden kleine Tutorials angeboten, damit man sich schnell in Eclipse zu Recht findet und kleinere Anwendungen schreiben kann [Goo113]. Geht man Googles Step-by-Step Anleitung durch, lernt man gleichzeitig etwas über die Architektur Androids, wenngleich die Erläuterungen nur soweit gehen, als dass nur Informationen genannt werden, die Google für einen Entwickler als wichtig erachtet. Beispielsweise wird in der Dokumentation nicht eindeutig klar, warum mit der Dalvik VM eine neue Laufzeitumgebung entwickelt wurde. Zu dieser schreibt Google nur knapp, dass sie für mobile Geräte optimiert wurde [Goo114]. Einem Entwickler ist auf dem ersten Blick gar nicht bewusst, dass die Dalvik VM Unterschiede zur Java Virtuellen Maschine aufweist (siehe 4.1.4). Diese Informationsreduktion seitens Google sorgt im Developer Guide auf der einen Seite für eine gute Übersichtlichkeit. Auf der anderen Seite kann man bestimmte technische Details nur erfahren, wenn man aktiv nach anderen Quellen sucht.

Google legt den Fokus auf die eigentliche Entwicklung. Deswegen gibt es auch so genannte *Developer Resources*. Die API Referenz ist sehr umfassend und mit Beispielen gespickt. Bei komplexeren Abläufen, wie z.B. die Verwendung der Kamera eines Gerätes, werden Step-by-Step-Anleitungen bereitgestellt. Sollten diese nicht ausreichend sein, kann ein Entwickler bei den Developer Resources nochmals nachschlagen, denn dort gibt es eine Fülle von Artikeln, Tutorials und Beispielcode, welcher vollständig ausführbar ist und zudem wiederverwendet werden darf [Goo115].

Auch eine Möglichkeit für Entwickler untereinander zu kommunizieren und sich austauschen zu können, wird von Google bereitgestellt [Goo116]. In der *Android Community* gibt es ebenfalls viele Möglichkeiten mit anderen Entwicklern in Kontakt zu treten und Neuigkeiten zu erfahren: Google Groups, Email-Listen und IRC.

Da Android unter den mobilen Betriebssystemen im Jahre 2011 eine Vormachtstellung mit einem Marktanteil von fast 50 Prozent einnimmt [Can15] und zudem quelloffen ist, gibt es auch abseits der offiziellen Dokumentation eine große Community und viele Foren, bei denen man Hilfe suchen kann. Genannt werden soll hier stellvertretend die Community *xda-developers* [xda11] mit über vier Millionen registrierten Mitgliedern, bei der neben Android Entwicklern auch Windows Phone und WebOS Entwickler unter den Mitgliedern vertreten sind. Diese sind zum Teil technisch hoch versiert, so dass dort unter anderem alternative Android-Images für diverse Mobiltelefone entwickelt werden.

Zusammenfassend lässt sich sagen, dass Android eine gute Dokumentation vorweisen kann. Sollten Fragen von offizieller Seite nicht beantwortet werden, lassen sich in inoffiziellen Communities mit hoher Wahrscheinlichkeit andere Entwickler finden, mit denen man zusammen ein Problem lösen kann.

5.4 Entwicklungsumgebungen

Google bezieht sich in den Tutorials immer auf Eclipse. Sei es bei der Installation einer Entwicklungsumgebung oder bei vertiefenden Tutorials mit Quelltext. Entscheidet sich ein Entwickler für die Entwicklungsumgebung Eclipse, bekommt er von Google ein „Rundum-Sorglos-Paket“. Ein Grund dafür ist das ADT Plug-In für Eclipse (siehe 5.4.1), welches von Google selbst entwickelt wird. Bei anderen Entwicklungsumgebungen wird es etwas schwieriger, da man sich die Informationen selbst erarbeiten muss bzw. diese nicht auf der Seite von Google stehen und es zudem offiziell keinen Support seitens Google gibt.

Bevor man sich aber für eine Entwicklungsumgebung entscheidet, müssen das Java Development Kit (JDK) und das Java Runtime Environment (JRE) installiert sein, da diese zu den Systemanforderungen des Android SDKs gehören. Das Android SDK ist unter *developer.android.com* zu finden, wo man zwischen den Versionen für Windows, MAC OS X und Linux wählen kann. Startet man das heruntergeladene SDK, beginnt erst die eigentliche Installation, denn man hat die Auswahl zwischen den verschiedenen API Revisionen der verschiedenen Android-Versionen und kann zudem auch USB Treiber installieren, mit denen ein Android-Gerät als solches erkannt wird und mit denen es später möglich sein wird, direkt auf einem Gerät zu debuggen (siehe 5.6).

Mit Hilfe des SDKs lassen sich auch alle installierten API Revisionen, Treiber und sonstige Pakete, wie Googles Admob Paket, auf einen Blick erkennen, was in der unteren Abbildung zu sehen ist. Eventuelle Updates oder auch ältere Versionen können über den Reiter *Available packages* nachgeladen werden.

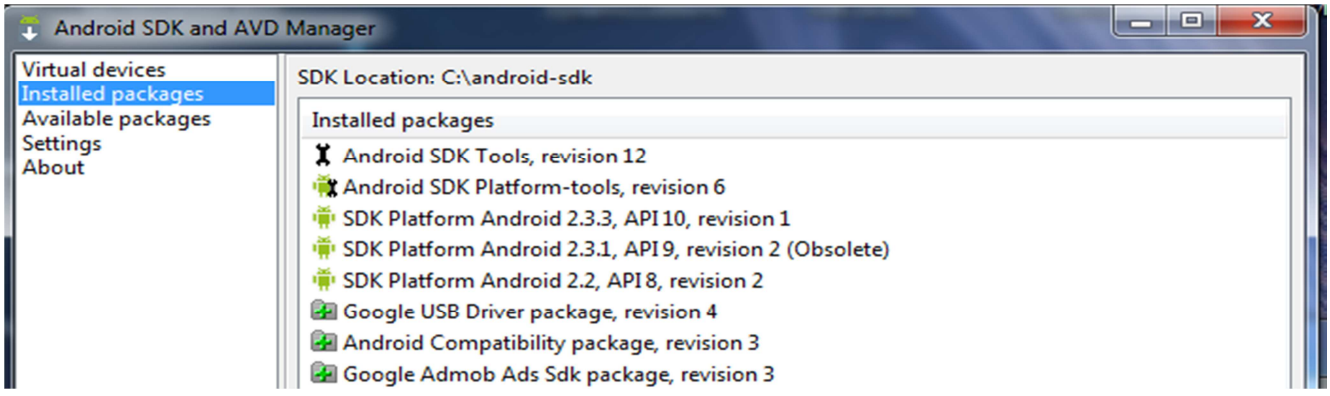


Abbildung 5: Android SDK Installed packages

Die Abbildung 6 zeigt den Menüpunkt *Virtual devices,* worunter sich alle schon installierten Android-Emulatoren verbergen, sowie die Möglichkeit weitere zu erstellen.

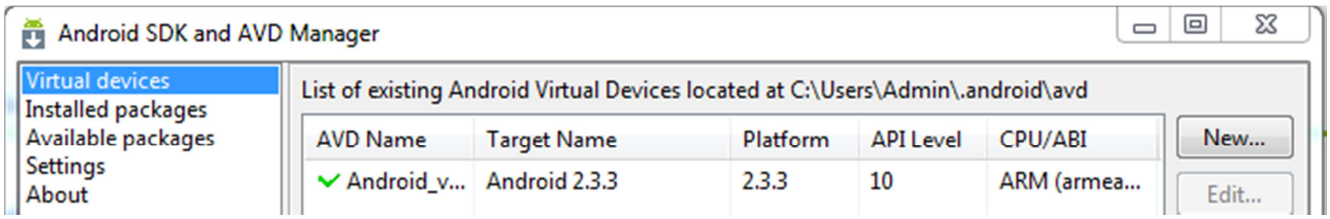


Abbildung 6: Android SDK Virtual devices

Mehrere Emulatoren machen insbesondere dann Sinn, wenn man sich nicht sicher ist, ob die entwickelte Anwendung auch unter älteren Android-Versionen lauffähig ist. Bei der Erstellung eines Emulators lässt sich außer der API Revision auch eine SD-Karte simulieren, sowie bestimmte Hardwareeigenschaften wie die Größe des Displays oder die Größe des Arbeitsspeichers.

Erst wenn diese Voraussetzungen erfüllt sind, kann man sich über die Entwicklungsumgebung, und wie diese mit dem SDK Tools zusammen arbeiten kann, Gedanken machen.

5.4.1 Eclipse

Eclipse dürfte die meistgenutzte Entwicklungsumgebung für Android sein, da es die Einzige ist, die den vollen Support seitens Google genießt und daher auch offiziell empfohlen wird. Für Eclipse entwickelt Google die Android Development Tools (ADT) [Goo1110]. Das ADT Plug-In macht Eclipse zu einer mächtigen und integrierten Entwicklungsumgebung für Android. Es integriert die Android APIs in Eclipse, ermöglicht die Erstellung der Benutzeroberfläche, den Build-Prozess inklusive der Signierung einer Android Applikation und das Debuggen einer solchen (siehe 5.6). Das ADT Plug-In kann über den Plug-In Manager von Eclipse heruntergeladen und installiert werden.

Abbildung 7: Eclipse Android Projekt

Die gute Integration macht sich schon bei der Erstellung einer neuen Android Anwendung bemerkbar. Zur Auswahl steht im Menü neben einer klassischen Java-Anwendung nun auch ein Android Projekt (vgl. Abb. 7).

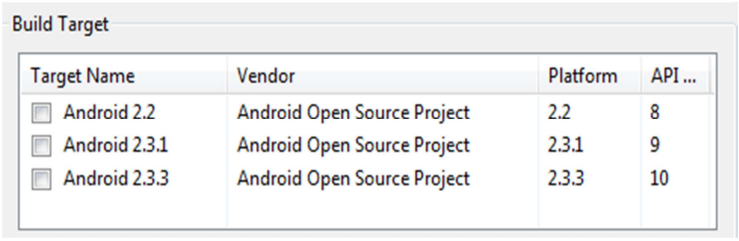


Abbildung 8:Eclipse Build Target

Auch alle im SDK unter dem *Installed Packages* Reiter befindlichen API Referenzen stehen im nächsten Schritt zur Verfügung (vgl. Abb. 5 und Abb. 8). Je nach Funktionsumfang kann ein Entwickler die von ihm gewünschte API Referenz auswählen. Diese wird dann von Eclipse automatisch eingebunden, d.h. alle in der ausgewählten API zur Verfügung stehenden Funktionen werden von Eclipse erkannt, so dass dann zum Beispiel auch eine Autovervollständigung, eine Fehlerkorrektur oder eine Hilfe bzw. API-Referenz zur Verfügung steht. Des Weiteren lässt sich eine Versionsnummer eines *minSDKs* angeben. Dieser Parameter wird automatisch in das Android Manifest (siehe 4.4) eingetragen und definiert die älteste von der Anwendung unterstütze Android-Version.

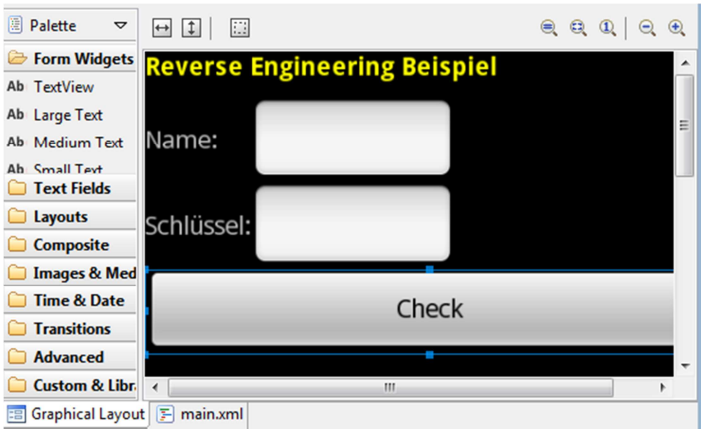


Abbildung 9: Eclipse Graphical Layout

Die in einem Android Projekt im Ordner *layout* befindlichen Layout-Dateien, die die Benutzeroberfläche gestalten, müssen nicht direkt in den XML-Dateien editiert werden. Stattdessen wird die Oberfläche, wie in der Abbildung 9 zu sehen, visualisiert. Einzelne UI-Elemente lassen sich per Drag-und-Drop auf die Activity ziehen, wodurch eine schnelle Entwicklung der Benutzeroberfläche ermöglicht wird.

Sollten dennoch manuelle Anpassungen in der jeweiligen XML-Datei nötig sein, kann man bei Bedarf die grafische Visualisierung deaktivieren.

Möchte man eine Anwendung veröffentlichen, d.h. zum Beispiel in den Android Market stellen oder sie als direkten Download auf einer Webseite anbieten, so muss die Applikation signiert werden. Dazu wird ein Zertifikat mit einem privaten Schlüssel benötigt. Dieses Zertifikat kann mit den im JDK enthaltenen Standard-Tools *Keytool* und *Jarsigner* selbst erstellt werden. Es dient dazu, den Urheber der Applikation identifizieren zu können. Während der Entwicklung bzw. währendes des Debuggens auf den Android Virtual Devices (AVD) und angeschlossenen Hardwaregeräten wird ein *debug-key* verwendet, welcher vom *Keytool* automatisch erzeugt wird. Das Signieren ist nur für das veröffentlichen einer Anwendung notwendig. Mit Hilfe des ADT Plug-Ins kann man mit dem ADT Export Wizard in einem einzigen Schritt die Anwendung kompilieren, einen *private key* generieren und die Applikation signieren. Dieser Export Wizard, der das Erstellen einer signierten APK-Datei stark vereinfacht, verbirgt sich ebenfalls leicht zu finden im Kontextmenü von Eclipse (vgl. Abb. 10).

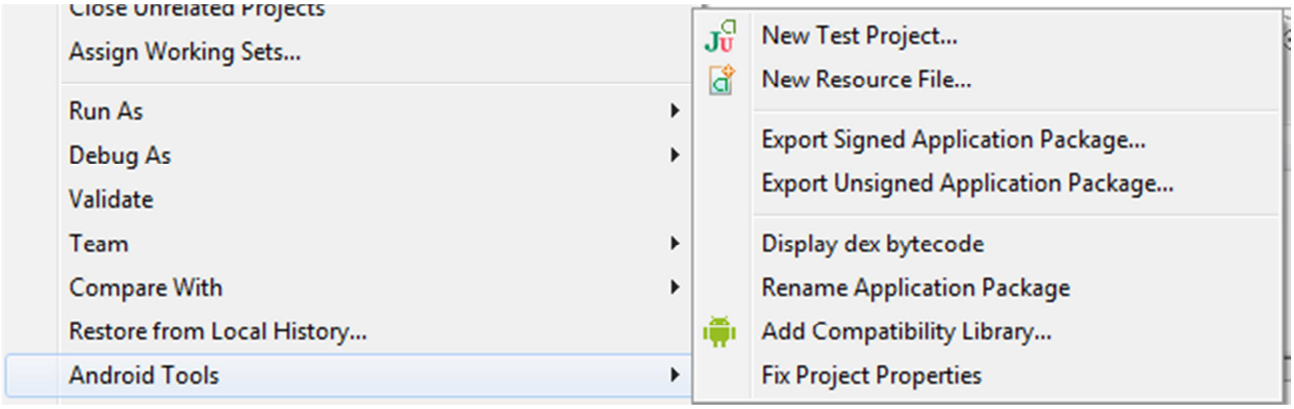


Abbildung 10: Eclipse Android Tools Kontextmenü

5.4.2 Netbeans

Als Alternative zu Eclipse soll im Folgenden Netbeans vorgestellt werden. Im Gegensatz zu Eclipse genießt Netbeans keine offizielle Unterstützung seitens Google. Dieser Umstand erschwert die Installation eines Plug-Ins für Netbeans, da es schlichtweg keine Anleitungen auf den Google-Seiten gibt. Entscheidet man sich gegen Eclipse und für eine alternative Entwicklungsumgebung wie Netbeans, muss der Entwickler selbst aktiv werden und nach inoffiziellen Plug-Ins suchen. Im Falle von Netbeans stößt man bei der Suche relativ schnell auf das NBAndroid Plug-In [NBA11], welches sich über den Plug-In Manager von Netbeans herunterladen und installieren lässt.

Anschließend steht, ähnlich wie bei Eclipse, neben den klassischen Java-Anwendungen ein Android Projekt zur Verfügung (vgl. Abb. 11).

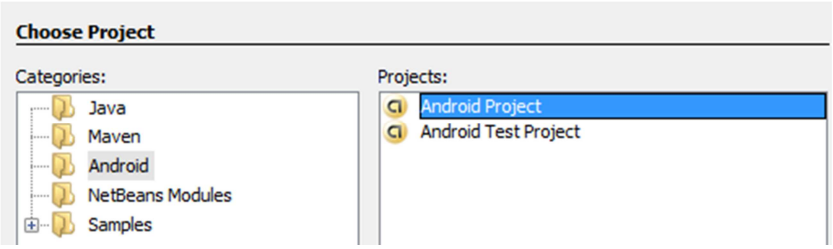


Abbildung 11: Netbeans Android Projekt

Auch werden alle im SDK unter dem *Installed Packages* Reiter befindlichen API Referenzen zur Auswahl zur Verfügung gestellt (vgl. Abb. 5 und Abb. 12). An dieser Stelle fehlt aber im Vergleich zu Eclipse der Parameter *minSDK.* Möchte man eine Untergrenze der zu unterstützenden Android-Version angeben, muss man in Netbeans das Android Manifest manuell editieren.

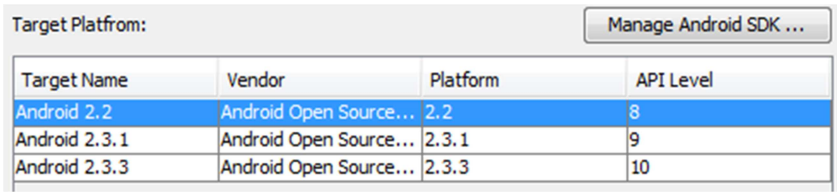


Abbildung 12: Netbeans Android Target Platform

Weiterhin stehen zwar die Android APIs zur Verfügung, diese sind aber nicht so gut in die Entwicklungsumgebung integriert, als dass zum Beispiel bestimmte Klassen automatisch erkannt werden:

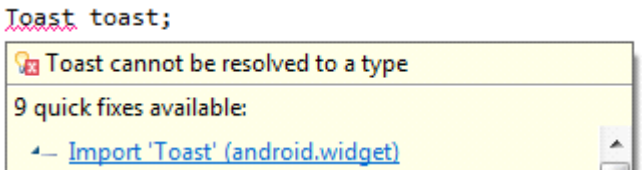


Abbildung 13: Erkennung von Klassen in Eclipse

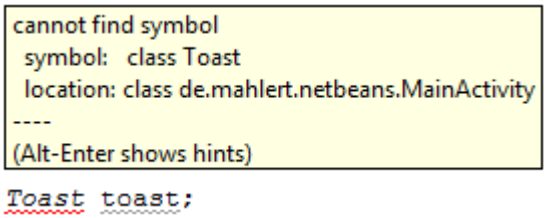


Abbildung 14: (Nicht-)Erkennung von Klassen in Netbeans

Im Beispiel wurde ein *Toast* definiert, ohne die entsprechende Klasse *android.widget.Toast* zu importieren. Eclipse schlägt die richtige Lösung vor und importiert die Klasse mit nur einem Klick automatisch, wonach der Fehler verschwunden ist. Netbeans liefert keine Lösungsvorschläge und lässt den Entwickler alleine. In Netbeans gehört der Fehler erst der Vergangenheit an, nachdem der Entwickler die Klasse manuell importiert hat.

In Netbeans gibt es mit dem NBAndroid Plug-In zudem keine grafische Visualisierung der Layout-Dateien. Stattdessen kann nur der in den XML-Dateien befindliche Text angezeigt und editiert werden, d.h. ein Drag-und-Drop von UI-Elementen ist unmöglich, was die Erstellung einer Benutzeroberfläche erschwert. Um dennoch eine grafische Unterstützung, sowie Drag-und-Drop, nutzen zu können, müsste auf Third-Party-Software zurückgegriffen werden. An dieser Stelle sei *DroidDraw* [Dro11] erwähnt, welches genau für einen solchen Einsatzzweck gedacht ist. Mit *DroidDraw* lassen sich bequem per Drag-und-Drop grafische Oberflächen erstellen. Das Tool generiert daraus den XML-Code, welchen man anschließend in Netbeans ersetzen kann.

Des Weiteren ist der Build-Prozess zwar einfach, erlaubt aber keine eigene Erstellung eines Zertifikates für die Signierung der Applikation. Stattdessen wird ungefragt der *debug key* für die Signierung verwendet, was in der unten stehenden Abbildung zu sehen ist.

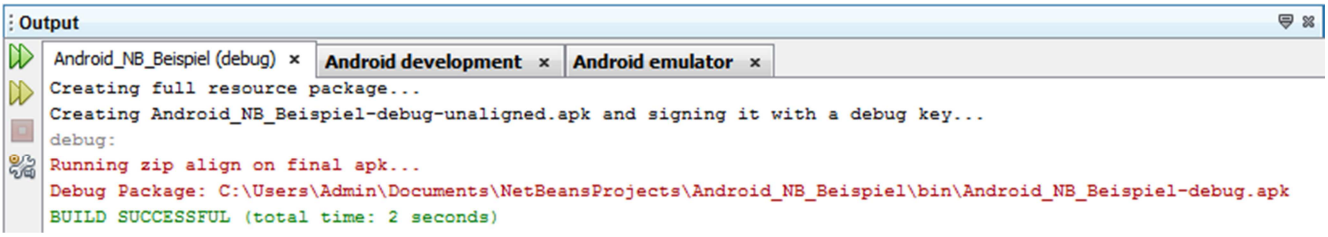


Abbildung 15: Netbeans Build Prozess

Um die Anwendung ordnungsgemäß mit einem *private key* zu signieren, muss das Zertifikat mit dem *Keytool* per Kommandozeile manuell erstellt werden. Anschließend kann in der Datei *build.properties* der Pfad zum Zertifikat angegeben werden:

```
key.store=/home/mahlert/android/mahlert.keystore
key.alias=mahlert
```

Abschließend muss in der Kommandozeile im Projektordner *ant release* ausgeführt werden, um die signierte Applikation zu erzeugen.

Zusammenfassend lässt sich sagen, dass eine alternative Entwicklungsumgebung wie Netbeans grundsätzlich möglich ist und auch funktioniert. Allerdings sind die Features längst nicht mit denen von Eclipse und dem ADT Plug-In zu vergleichen, denn die aufgezeigten Probleme sind in Eclipse nicht vorhanden. Es wurde auch kein bemerkenswerter Vorteil in Netbeans gefunden. Der einzige Vorteil liegt darin, dass sich Entwickler, die schon seit Jahren Netbeans nutzen, nicht umgewöhnen müssen. Diesen Vorteil zahlen sie allerdings mit dem Preis, dass die verfügbaren Plug-Ins nicht so komfortabel wie das ADT Plug-In sind. Da Google zudem auch keinen Support für andere Entwicklungsumgeben leistet, kosten Probleme, aufgrund der Tatsache, dass kein Developer Guide zu Verfügung steht und man selber nach Lösungen suchen muss, mehr Zeit. Oftmals muss man, wie im Beispiel von *DroidDraw*, dennoch auf andere Software zugreifen, so dass ein Umstieg auf Eclipse im Vorhinein doch lukrativer wirkt.

5.5 Beispiele

In diesem Kapitel sollen die zwei Beispiele *ToDo-Liste* und *IP Cam* vorgestellt werden. Ersteres nutzt keine besondere Hardware des Endgerätes, wohingegen die IP Cam auf die Kamera eines Gerätes zugreift. Die Beispiele sollen einerseits zum besseren Verständnis von Android beitragen und andererseits für den späteren Vergleich zu den webbasierten Techniken dienen. Zur Entwicklung wurde jeweils Eclipse inklusive ADT Plug-In genutzt (siehe 5.4.1).

5.5.1 ToDo-Liste

Die *ToDo-Liste* soll eine native Anwendung sein, in der man ToDo's bzw. Notizen in eine Liste persistent abspeichern kann, so dass sie nach einem Neustart der Anwendung immer noch vorhanden sind. Des Weiteren sollen die Notizen vom Benutzer auch bearbeitet und gelöscht werden können.

Um die Daten persistent zu speichern, wird eine SQLite Datenbank verwendet. SQLite ist eine in Android eingebettete relationale Datenbank (siehe 4.1.3) und verbraucht während der Laufzeit nur etwa 300 KByte an Speicher [Squ11]. Die Datenbank erfordert weder eine Installation, noch eine Administration, sondern wird von Android automatisch verwaltet. Möchte man die Daten anderen Anwendungen zur Verfügung stellen, muss dafür ein Content Provider (siehe 4.2.3) implementiert werden.

Zum Erzeugen einer SQLite Datenbank wird eine Helper-Klasse erstellt, in welcher beim Starten der Anwendung eine Verbindung zur Datenbank aufgebaut wird. Dazu wird als erstes geprüft, ob die Datenbank und die benötigten Tabellen überhaupt existieren. Sollte die Datenbank schon vorhanden sein, wird eine Verbindung zu dieser hergestellt, anderenfalls wird die Datenbank erzeugt und anschließend zu dieser verbunden. Dazu muss die Methode *onCreate()* überschrieben werden. Zusätzlich muss auch die Methode *onUpgrade()* überschrieben werden. Diese Methode wird aufgerufen, wenn sich das Datenbankschema oder die Datenbankversion ändert:

```
        @Override
        public void onCreate(SQLiteDatabase database) {
                database.execSQL("CREATE TABLE todo_data (_id INTEGER PRIMARY KEY AUTOINCREMENT, "
                + "ueberschrift TEXT NOT NULL, beschreibung TEXT NOT NULL);");
        }
        @Override
        public void onUpgrade(SQLiteDatabase database, int oldVersion,
                        int newVersion) {
                Log.w(TodoDatenbankHelper.class.getName(), "Datenbank wird geupgraded - Daten gehen verloren!");
                database.execSQL("DROP TABLE IF EXISTS todo_data");
                onCreate(database);
        }
}
```

Ist eine Verbindung zur Datenbank hergestellt, stellt die Klasse *TodoDatenbankAdapter* die Methoden zum Einfügen, Updaten und Löschen von Einträgen bereit. Für die ersten beiden genannten Methoden ist dabei das Obekt *ContentValues* von großer Bedeutung, mit welchem man Key/Value Paare definieren kann, welche man dann den Einfüge- und Updatemethoden übergibt:

```
        private static final String DATENBANK_TABELLE = "todo_data";
        public static final String KEY_UEBERSCHRIFT = "ueberschrift";
        public static final String KEY_BESCHREIBUNG = "beschreibung";

        public long erstelleTodo(String ueberschrift, String beschreibung) {
                ContentValues initialValues = createContentValues(ueberschrift,
                                beschreibung);

                return database.insert(DATENBANK_TABELLE, null, initialValues);
        }
        private ContentValues createContentValues(String ueberschrift,
                        String beschreibung) {
                ContentValues values = new ContentValues();
                values.put(KEY_UEBERSCHRIFT, ueberschrift);
                values.put(KEY_BESCHREIBUNG, beschreibung);
                return values;
        }
```

Datenbankabfragen lassen sich auf zwei Arten realisieren. Zum einen steht die Methode *rawQuery()* bereit, welche SQL-Abfragen verarbeitet, und zum anderen die Methode *query()*, die ein Interface für SQL-Abfragen darstellt, d.h. man übergibt nur die gewünschten Parameter und muss sich keine Gedanken über die SQL-Syntax machen. In beiden Fällen wird ein Cursor zurückgegeben, welcher das Ergebnis der SQL-Abfrage darstellt. Mit Hilfe des Cursors lassen sich alle Ergebnisse durchlaufen.

```
        public Cursor fetchAllTodos() {
                return database.query(DATENBANK_TABELLE, new String[] { KEY_ROWID,
                                KEY_UEBERSCHRIFT, KEY_BESCHREIBUNG }, null, null, null,
                                null, null);
```

```
}
```

Für die Applikation werden die Activities *TodosUebersicht* und *TodoDetails* erstellt. In der Übersicht sollen alle schon abgespeicherten Titel der Notizen angezeigt werden. In dieser Activity erhält der Anwender auch die Möglichkeit Notizen hinzuzufügen und zu löschen. Nach dem Klick auf den Titel einer Notiz wird die Activity *TodoDetails* aufgerufen, in der der eigentliche Inhalt der Notiz angezeigt wird und bearbeitet werden kann. Auch wird diese Ansicht genutzt, wenn ein weiterer ToDo-Punkt hinzugefügt wird.

Die Activity *TodosUebersicht* enthält eine *ListView*, welche in der Lage ist, scrollbare Items anzuzeigen. Die Daten bekommt ein *ListView* via einen Adapter. Neben der Möglichkeit eigene Adapter zu implementieren, bietet Android standardmäßig einen *ArrayAdapter* und einen *CursorAdapter* an. Der *ArrayAdapter* verarbeitet Arrays und Listen, während der *CursorAdapter* das Ergebnis einer Datenbankanfrage, einen schon erwähnten *Cursor*, auswertet. Im Falle des Beispiels kann der *CursorAdapter* implementiert werden, da der letzte Codeausschnitt die die komplette Tabelle der ToDo's ausliest und einen Cursor zurückgibt:

```
private void fuelleDaten() {
        cursor = dbHelper.fetchAllTodos();

        String[] from = new String[] { TodoDatenbankAdapter.KEY_UEBERSCHRIFT };
        int[] to = new int[] { R.id.label };

        SimpleCursorAdapter todos = new SimpleCursorAdapter(this,
                        R.layout.todo_row, cursor, from, to);
        setListAdapter(todos);
}
```

Die gespeicherten Notizen lassen sich löschen, indem man lange auf die zu löschende Notiz drückt und die entsprechende Option im Kontextmenü auswählt.
Durch den Klick auf den definierten *Hinzufügen-Button* ruft ein Intent (siehe 4.3) die *TodoDetails* Activity auf:

```
private void erstelleTodo() {
        Intent i = new Intent(this, TodoDetails.class);
        startActivityForResult(i, 1);
}
```

In dieser Activity lassen sich Einträge mit Hilfe der schon vorgestellten *TodoDatenbankAdapter* hinzufügen und editieren.

Die untenstehenden Abbildungen zeigen die installierte ToDo-Applikation in Aktion.

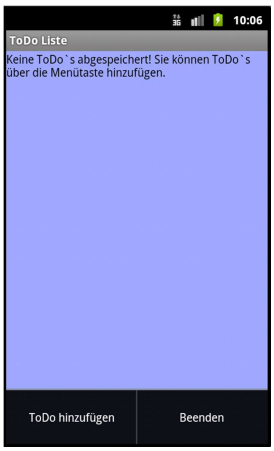


Abbildung 16: erster Programmstart

Abbildung 17: Hinzufügen es ToDo's

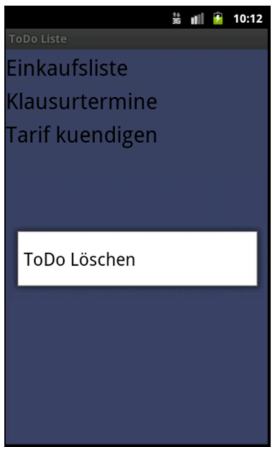


Abbildung 18: Löschen eines ToDo's

Die gleiche Anwendung wird in Kapitel 7 dieser Arbeit auch mit Hilfe webbasierter Techniken realisiert. Für einen besseren Vergleich zwischen einer nativen Entwicklung und einem webbasierten Framework soll in etwa gleich viel Zeit für die Entwicklung verwendet werden. Aus diesem Grund soll an dieser Stelle auf eine schönere Benutzeroberfläche, die in einer nativen Entwicklung relativ zeitaufwendig ist, verzichtet werden, damit der Vergleich nicht verfälscht wird.

Das fertige Beispiel ist als Quellcode und als signierte APK-Datei auf der CD im Ordner *nativ**ToDo-Liste* enthalten.

5.5.2 IP Cam

Die *IP Cam* ist eine Applikation, welche vom Autor dieser Arbeit zusammen mit einem Kommilitonen im Rahmen des Google Android Praktikums an der Technischen Universität München im Wintersemester 2010/11 entwickelt wurde. Die Entwicklung war ein Projekt, welches über ein ganzes Semester verlief und relativ komplex war. Aus diesem Grund kann an dieser Stelle nicht im Detail auf den Quellcode eingegangen werden. Stattdessen soll dieses Beispiel in erster Linie die Möglichkeiten unter Android verdeutlichen.

Eine IP Cam ist ein Gerät, welches im Wesentlichen aus einer Kamera und einer Netzwerkschnittstelle besteht. Die Idee dahinter ist, dass man auf eine solche IP Cam per Netzwerk oder Internet auf den Stream der Kamera zugreift – von überall aus, zu jeder Zeit und von mehreren Leuten gleichzeitig. Im Zeitalter von Web 2.0 möchten viele Leute sämtliche Erfahrungen mit ihren Freunden teilen. Moderne Mobiltelefone haben oftmals eine Kamera (siehe 4.5). Da ein Mobiltelefon handlich ist und immer in der Nähe liegt, bietet es sich an eine Applikation zu schreiben, die ein Telefon in eine IP Cam „verwandelt." Darüber hinaus soll es gleichzeitig möglich sein, dass der Besitzer eines Gerätes während des Streamings ggf. auch Snapshots schießen kann.

Das Konzept sah vor, dass auf dem Android Gerät ein Socket geöffnet wird und Clients sich über diesen per Webbrowser verbinden können. Im Webbrowser soll der Stream der Kamera ausgegeben werden.

5.5.2.1 Architektur

Um diesen komplexen Anforderungen gerecht zu werden und in Anbetracht der Tatsache, dass zwei Entwickler beteiligt sind, musste eine geeignete Architektur gefunden werden. Die Entscheidung fiel auf eine Schichtenarchitektur, wobei einzelne Aspekte der Funktionalität in Komponenten zusammengefasst und je nach Abstraktionsniveau in die Architektur angeordnet werden. Dabei handelt es sich um eine „strenge" Schichtenarchitektur, d.h. die Komponenten einer Schicht können nur von den Komponenten der oberen Schichten aufgerufen werden. Die Architektur besteht aus der Präsentationsschicht, der Logikschicht und er Serviceschicht. Durch die Teilung in Komponenten ist es problemlos möglich, dass mehrere Entwickler gleichzeitig parallel an einer Anwendung arbeiten können.

Die Präsentationsschicht kümmert sich nur um die grafische Präsentation der Applikation und die Behandlung von UI-Events. Sie steuert die Interaktion des Benutzers und führt Funktionen der Logikschicht aus. Diese wiederum besteht aus Funktionen, die im Kontext der IP Cam als "komplex" bezeichnet werden können, da sie beschreiben *wie* die Applikation auf eine Nutzerinteraktion reagiert. Die komplexen Funktionen bestehen meistens aus mehreren Funktionen der Serviceschicht. Die Serviceschicht ist die unterste Schicht und besteht aus Funktionen, die als „einfach" bezeichnet werden können. Die Komponenten dieser Schicht bauen direkt auf die Android-APIs auf.

Folgende Grafik verdeutlicht die Architektur anhand eines konkreten Anwendungsfalls:

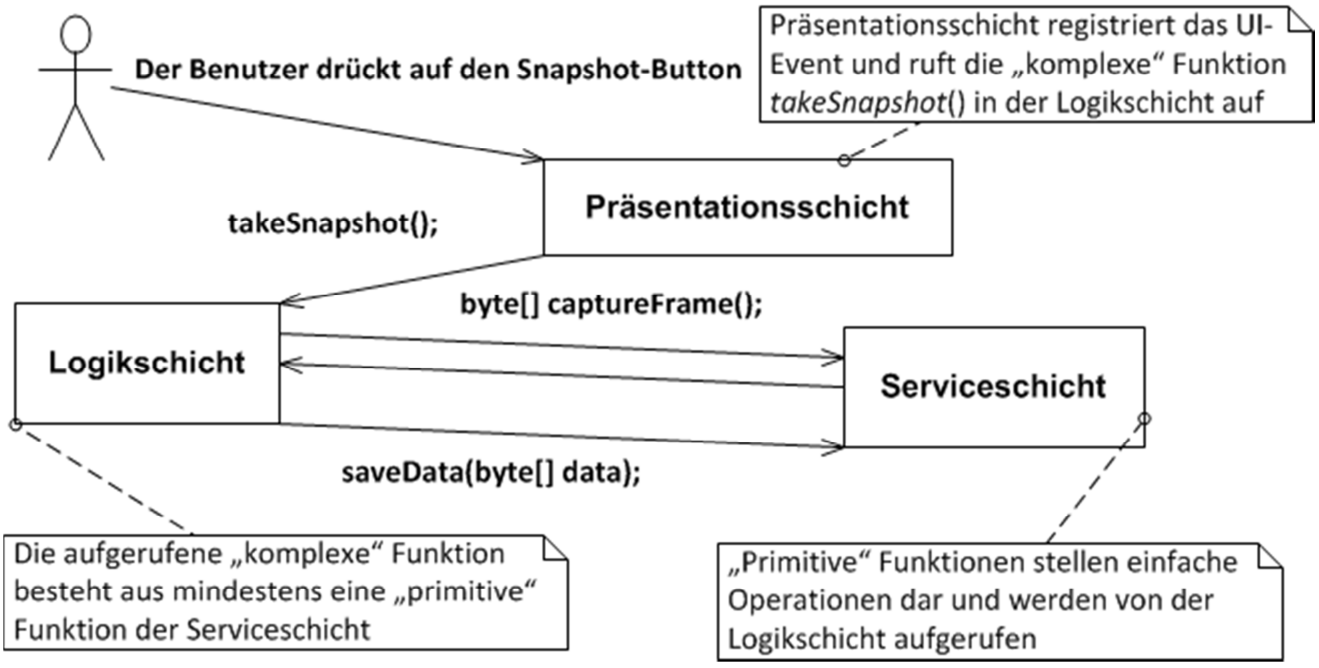


Abbildung 19: IP Cam Architektur Anwendungsfall

Der Nutzer drückt auf seinem Display den Button, mit dem man Snapshots schießt. Dieser Button ist als UI-Element im XML-Teil der Applikation in *button.xml* definiert:

```
<!--Snapshot Button -->
    <ImageButton
        android:layout_width="50dip"
        android:layout_height="50dip"
        android:src="@drawable/camera"
        android:scaleType="center"
        android:layout_alignParentRight="true"
        android:id="@+id/snapshot_button"
        android:background="@android:color/transparent"
        android:visibility="gone"
    />
```

Beim Starten der Applikation wird zu diesem Button durch die ID *snapshot_button* ein *OnClickListener* angebunden:

```
private void initSnapshotButton(){
    this.snapshotButton = (ImageButton) ActivityResource.getView(R.id.snapshot_button);
    this.snapshotButton.setOnClickListener(new OnClickListener(){
        @Override
        public void onClick(View arg0) {
            counterText.setVisibility(View.GONE);
            logicManager.snapshotToSDCard(inputContainer);
        }
    });
}
```

Wird der Button auf der Präsentationsschicht gedrückt, wird mit *logicManager.snapshotToSDCard(inputContainer)* eine “komplexe” Funktion der Logikschicht aufgerufen:

```
public void snapshotToSDCard(final View view){
    view.setVisibility(View.GONE);
    this.serviceManager.getSnapshot(new SimpleListener(){
        @Override
        public Object onAction(Object... obj) {
```

```
                    if(obj[0] != null && obj[0] instanceof byte[]){
                        serviceManager.writeJPEGToSDCard((byte[]) obj[0]);
                        view.setVisibility(View.VISIBLE);
                        Toaster.makeText(NameFromTime.getLastSaved());

                    }

                    return null;
                }
            });
        }
```

Diese Methode ruft weiterhin zwei weitere Funktionen der Serviceschicht auf. Erst wird die "einfache" Funktion getSnapshot(…) aufgerufen, woraufhin auf eine Antwort durch die SimpleListener-Instanz gewartet wird. Nachdem die Antwort bzw. ein Byte Array, welches das Snapshot enthält, zurückgeliefert wird, wird eine weitere "einfache" Funktion der Serviceschicht aufgerufen, die das Snapshot auf die SD-Karte speichert.

5.5.2.2 Besondere Herausforderungen während der Implementierung

Google stellt in der API Referenz für die Klasse *android.hardware.Camera* eine Step-by-Step Anleitung bereit, damit ein Entwickler die Kamera API ohne weitere Probleme implementieren kann [Goo117]. Jedoch hat sich diese Anleitung als zu ungenau erwiesen und sich die Implementierung somit als schwieriger herausgestellt, als man es zunächst annehmen konnte. Außer der genannten Anleitung gibt es keine weiteren Tutorials oder Quellcodebeispiele seitens Google zu diesem Thema. So ist zum Bespiel nirgendwo erklärt, dass die Kamera für ihre Funktionalität unbedingt ein spezielles *SurfaceView* braucht, in welchem das Preview angezeigt wird. Damit man auf die einzelnen Frames der laufenden Kamera zugreifen kann, muss man diese unbedingt in so einem View anzeigen:

```
this.surfaceView =(SurfaceView) ((RelativeLayout) ActivityResource.getView(R.id.main)).getChildAt(0);
                    SurfaceHolder surfaceHolder = surfaceView.getHolder();
                    surfaceHolder.addCallback(this);
                    surfaceHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
```

Weiterhin ist in der offiziellen Dokumentation nicht erklärt, dass man auf einen Callback des Views warten muss, bevor die Kamera gestartet wird:

```
@Override
        public void surfaceChanged(SurfaceHolder arg0, int arg1, int arg2, int arg3) {
            startCameraStream();
        }
```

Versucht man die Kamera vor diesem Callback zu starten, wird die Applikation sofort beendet. Erschwerend kommt hinzu, dass man auch keinen Exception Stack in der Konsole findet. D.h. es wird auch keine Exception geworfen. Die Untersuchung dieses komplexen Verhaltens der Kamera hat die Fertigstellung der *IP Cam* verzögert.

Nach der Festlegung einer passenden Architektur und die Implementierung der Komponenten, die auf die Funktionalität der Kamera API aufbauen, sind wir auf die nächste große Herausforderung gestoßen. Man brauchte einen mächtigen und funktionalen HTTP-Server. Da diese Funktionalität in der Android API fehlt, müssten wir dieses entweder selber implementieren oder eine kompatible Open Source API suchen. Die Implementierung eines Servers auf Socketebene hat sich als eine Aufgabe erwiesen, dessen Komplexität und Aufwand viel höher ist als der Rest des Projekts. Weiterhin hat diese Aufgabe wenig mit den Android APIs zu tun, da die Bibliotheken, die man dafür nutzt, auch im Java SDK zu finden sind. Die Suche nach einer Open

Source API wurde dadurch erschwert, dass die meisten Server APIs auf Java-Servlets basieren. Die Java-Servlets findet man aber nicht in Android. Schließlich sind wir auf das *Simple Framework* Open Source Projekt [Sim11] gestoßen. Das Framework kümmert sich um alle Aspekte eines HTTP-Servers, wie zum Beispiel die Erstellung von Threads für einzelne Nutzer bzw. deren Requests an den Server. Weiterhin ist das Framework mit Android so kompatibel, dass man die Java-Dateien des Frameworks direkt und ohne Anpassungen in den Android-Sourceordner kopieren und nutzen kann.

Nachdem auf die einzelnen Frames der laufenden Kamera zugegriffen werden konnte und man über einen robusten HTTP-Server verfügte, musste noch die Frage geklärt werden, wie man die Frames zum Webbrowser übertragen kann. Dazu wurde der Motion JPEG (MJPEG) ausgewählt, welcher auch bei klassischen IP-Kameras zum Einsatz kommt [Wik117].

Die fertige Anwendung findet man im Android Market unter *Happy Droids IP Cam*. Auf der folgenden Abbildung sieht man auf der linken Seite die laufende IP Cam, während rechts der Stream auf einem Client in Googles Chrome ausgegeben wird.

Abbildung 20: Die fertiggestellte IP Cam

5.6 Debugging

Bei der Entwicklung von Anwendungen treten oftmals Bugs auf, die ohne Debugging nicht zu finden wären. Dem Android SDK liegt ein Tool namens Android Debug Bridge (ADB) bei. Die ADB ist ein Kommandozeilenprogramm, welches mit dem Android Emulator und mit einem an den Rechner über USB verbundenem Android Gerät kommunizieren kann. Sie ist ein Client/Serverprogramm und besteht aus 3 Komponenten [Goo118]:

- Einem Client, der auf dem Android Gerät oder Emulator läuft.
- Einem Server, der als Hintergrundprozess auf dem Android Gerät oder Emulator läuft.
- Einem Daemon, der als Hintergrundprozess auf dem Android Gerät oder Emulator läuft.

Möchte man auf einem Android Gerät debuggen, muss *USB-Debugging* in den Einstellungen des Gerätes zunächst aktiviert werden.

Da ein kommandozeilenbasiertes Debugging komplex ist, hat Google im ADT Plug-In für Eclipse (siehe 5.4.1) das Debugging via ADB integriert, so dass dies über die grafische Oberfläche von Eclipse möglich ist. Andere Entwicklungsumgebungen ohne ADT Plug-In müssen das Java Debug Wire Protocol (JDWP) unterstützen, um auf die Funktionen von ADB zugreifen zu können [Goo119]. Im Folgenden wird Eclipse als Entwicklungsumgebung genutzt.

Um seine Anwendung zu Debuggen, ist es sinnvoll *Logcat* zu verwenden. Mit Logcat ist es leicht und schnell möglich, Informationen von der laufenden Anwendung zu erhalten. Die Klasse *android.util.Log* muss für eine Verwendung von Logcat zunächst importiert werden. Danach kann die statische Klasse *Log* aufgerufen werden. Logcat kann mit verschiedenen Parametern genutzt werden:

V — Verbose (niedrigste Priorität)
D — Debug
I — Info
W — Warning
E — Error
F — Fatal
S — Silent (niedrigste Priorität)

Der Parameter *D* wird für das Debugging verwendet. Fehler werden von Logcat beim Aufruf mit dem Parameter *E* ausgegeben.

```
public class DebuggingActivity extends Activity {

    public static final String LOG_TAG = "Mahlert_Debug_Beispiel";

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        int Zufallszahl = (int) ((Math.random()*100)+1);

        Log.d(LOG_TAG, "Zufallszahl: " + Zufallszahl);
    }
}
```

Abbildung 21: Android Debug Beispiel

Im Beispiel wird eine Zufallszahl generiert, welche mit Logcat ausgegeben werden soll (vgl. Abb. 21). Logcat wird mit zwei Argumenten aufgerufen. Das erste Argument dient zur Identifikation der Applikation. Es ist daher sinnvoll diese Information, auch *LOG_TAG* genannt, als *public* zu definieren, um von anderen Klassen der gleichen Anwendung darauf zugreifen zu können. Das zweite Argument ist die Information, die geloggt werden soll. Im Beispiel ist das die Zufallszahl.

Bevor man die Anwendung startet, muss das Logcat-Fenster aufgerufen werden, welches in Eclipse über *Window > Show view > Other, Android > Logcat* angezeigt werden kann.

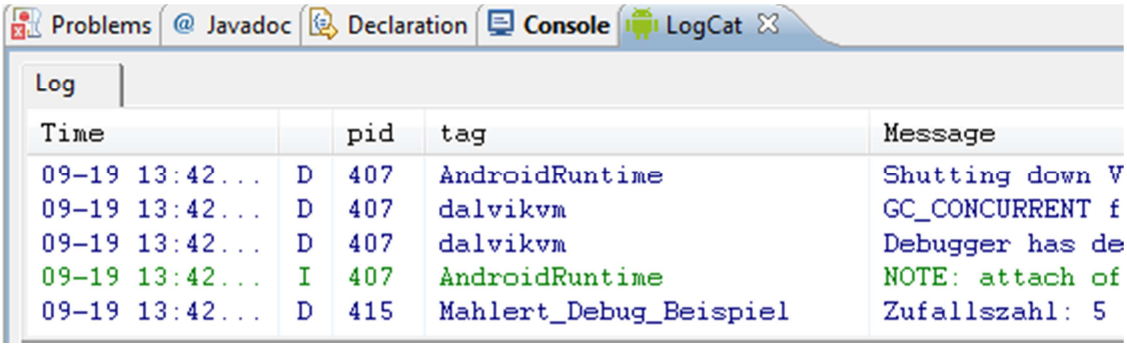


Abbildung 22: Android Logcat

In der oberen Abbildung ist die generierte Zufallszahl nun einfach zu erkennen. Diese Art des Debuggens kann in vielen Fällen hilfreich sein, doch geht sie noch nicht weit genug.

Android erlaubt es auch Breakpoints zu setzen. Dies bedeutet, dass man die Anwendung an einer spezifischen Stelle anhalten und den Zustand der Anwendung zu diesem Zeitpunkt sehen kann. Um einen Breakpoint in E-

clipse zu setzen, muss man lediglich auf die Zeile des jeweiligen Codes doppelt klicken. Eclipse illustriert einen Breakpoint mit einem Punkt vor der Zeile:

```
int Zufallszahl = (int) ((Math.random()*100)+1);
```

Abbildung 23: Eclipse Breakpoint

Startet man das Debugging, erkennt Eclipse den Breakpoint und fragt nach der Debug-Ansicht. Diese Ansicht ist selbsterklärend, denn das Debugging lässt sich mit drei Pfeilen steuern:

Abbildung 24: Eclipse Debug Steuerung

Der erste Pfeil bedeutet *Step In*. Ist der Breakpoint auf einen Funktionsaufruf gesetzt, dann lässt sich mit F5 diese Funktion aufrufen und Zeile für Zeile durchgehen, während mit F6, *Step Over*, die Funktion übersprungen werden würde. Mit F7, *Step Return*, kann zu den vorherigen Zuständen zurückgekehrt werden.

Im Beispiel wurde der Breakpoint auf die Wertzuweisung der Zufallszahl gesetzt. Um den Wert der Zufallszahl sehen zu können, muss zunächst die Zeile ausgeführt werden. Wie in der nachfolgenden Abbildung zu sehen, kann man nach dem Benutzen von F6 den Wert der Zufallszahl ablesen.

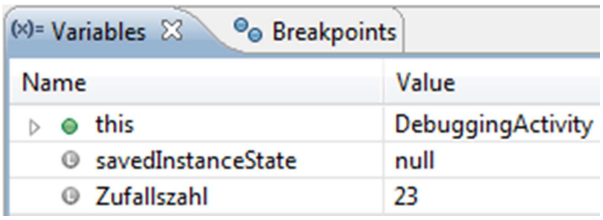


Abbildung 25: Eclipse Debug Variablen

Diese Art des Debuggens ist sehr mächtig und wird bei komplexen Anwendungen mit hoher Wahrscheinlichkeit öfter genutzt.

In den meisten Fällen reicht das Debugging via Emulator. Ein Nachteil des Emulators ist jedoch dessen Geschwindigkeit, denn diese ist selbst auf modernsten Systemen als träge zu bezeichnen. Im Internet finden sich dazu viele Einträge und Fragen von Entwicklern, die den Emulator gern schneller hätten [And11]. Der Grund für die Trägheit des Emulators ist der Tatsache geschuldet, dass in den Android Geräten ARM CPUs verbaut sind, während in einem PC x86 Prozessoren laufen und die CPU Instruktionen zunächst umgewandelt werden müssen. Da Google und Intel vor kurzem angekündigt haben, dass Android ab 2012 auch auf Intel Atom Prozessoren lauffähig sein wird [tel11], was eine offizielle Öffnung der x86-Plattform für Android entspricht, besteht eine nicht geringe Wahrscheinlichkeit, dass der Emulator in zukünftigen Versionen performanter sein wird.

Das Debuggen mittels angeschlossenen Geräts ist bisher aber dem Emulator bzgl. der Performanz überlegen. Des Weiteren kann die Möglichkeit bestehen, dass eine Anwendung zwar im Emulator und auch auf vielen anderen Android-Geräten problemlos läuft, aber auf vereinzelten Geräten aufgrund von herstellerspezifischen Eigenheiten nicht. In diesem Fall käme man mit dem Debuggen via Emulator nicht weiter, sondern muss dies mit dem Gerät vornehmen, auf dem die Probleme auftauchen. Weiterhin lassen sich bestimmte Applikationen nur auf einem Gerät sinnvoll debuggen. Als Beispiel käme dafür die *IP Cam* (siehe 5.5.2) in Frage, da diese auf die interne Kamera der Geräte zugreift und man innerhalb eines Emulators nicht viel sehen könnte.

5.7 Reverse Engineering

Möchte man mit seiner Applikation Geld verdienen, beinhaltet sie unternehmenskritische Daten oder verfolgt man das Ziel, dass nicht jeder den Quellcode der eigenen Applikation einsehen kann, dann muss man sich überlegen, wie man seinen Quellcode vor anderen schützen kann.

Ein Entwickler schreibt seine Anwendungen mit dem SDK in Java. Auch zur Entwicklungszeit wird der Code von einem Java-Compiler in Java-Bytecode übersetzt. Um eine fertige Applikation in das für Android benötigte DEX-Format zu transformieren, kommt das *dx-Tool* zur Anwendung, welches als Cross-Compiler im Android SDK enthalten ist (siehe 4.1.4). Der daraus entstandene Dalvik-Bytecode hat nur noch eine entfernte Ähnlichkeit zum Java-Bytecode. Aufgrund dieser Unterschiede funktionieren Java-Decompiler wie zum Beispiel *JD* [JD11] nicht.

Eine Android Anwendung ist in einer APK-Datei gepackt. Im Wesentlichen stellt dies ein Zip-Archiv da, in welchem alle Ressourcen einer Applikation liegen. Anders als in Java, wo alle Klassen als separate CLASS-Dateien kompiliert werden, kompiliert das *dx-Tool* diese zu einer einzigen DEX-Datei (siehe 4.1.4). Mit dem frei erhältlichen *APK-Tool* ist es möglich, eine APK-Datei und insbesondere die darin gepackte DEX-Datei, die den gesamten Quelltext einer Anwendung erhält, zu dekodieren [Wiś11]. Das bedeutet, dass man nach der Nutzung dieses Tools die verwendeten Java-Klassen, sowie deren Dalvik-Bytecode, sehen kann. Die Opcodes der Dalvik VM sind für jeden zugänglich [The11], so dass ein Reverse Engineering möglich ist.

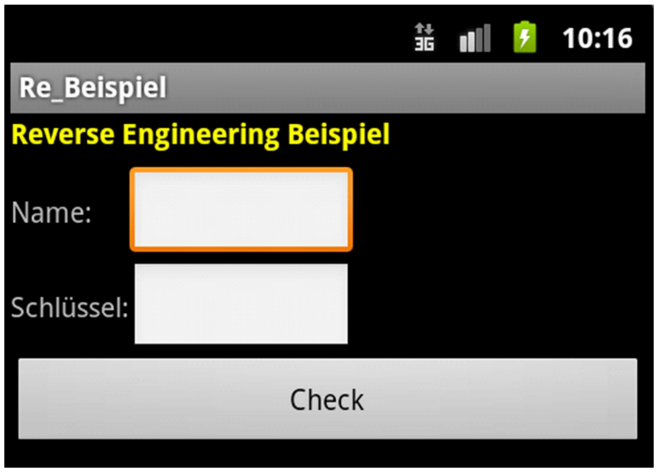


Abbildung 27: Reverse Engineering Beispiel

Der Prozess des Reverse Engineerings soll anhand eines Beispiels gezeigt werden. In dem Beispiel wird nach einem Namen und einem Schlüssel gefragt (vgl. Abbildung links). Stimmt die Eingabe, wird eine erfolgreiche Meldung ausgegeben. Um den korrekten Namen und Schlüssel zu finden, wird das *APK-Tool* eingesetzt, um Einsicht in den Dalvik-Bytecode zu erlangen. Der Aufruf *apktool d re_beispiel.apk re_beispiel_decode* dekodiert das Beispiel in den Ordner *re_beispiel_decode.* Der Output ist in der Abbildung 26 zu sehen. Für jede in der Applikation verwendete Klasse ist eine Datei erstellt worden, die den Dalvik-Bytecode beinhaltet. Dabei besitzen die Klassen den gleichen Namen wie im Java-Projekt.

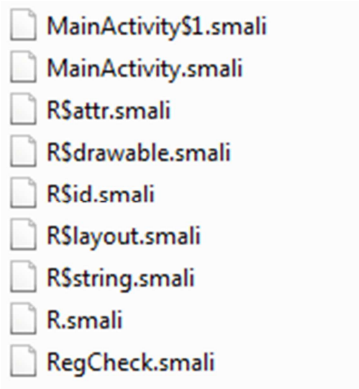


Abbildung 26: APK-Tool Output

Für einen potenziellen Angreifer ist die Klasse mit dem Namen *RegCheck* auffällig. Öffnet man diese Klasse in einem Editor, findet man die Funktion *checkNameUndKey(String, String)* mit einem Boolean als Rückgabewert, im Quelltext anhand des Buchstaben *Z* zu erkennen (Auszug):

```
.method public static checkNameUndKey(Ljava/lang/String;Ljava/lang/String;)Z
   .parameter "name"
   .parameter "key"

   const-string v0, "Bachelor"
   invoke-virtual {p0, v0}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z
   move-result v0
   if-eqz v0, :cond_0
   const-string v0, "Arbeit"
```

```
    invoke-virtual {p1, v0}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z
    move-result v0
    if-eqz v0, :cond_0
    const/4 v0, 0x1

    :goto_0
    return v0

    :cond_0
    const/4 v0, 0x0
    goto :goto_0
.end method
```

Mit Hilfe der Dalvik-Opcodes ist der Dalvik-Bytecode relativ einfach zu lesen. Man erkennt, dass es zwei Parameter p0 und p1 gibt, wobei in p0 der übergebene Name und in p1 der Schlüssel steht. Der Name wird gegen *Bachelor* und der Schlüssel gegen *Arbeit* geprüft. Stimmen diese Werte überein, wird der Wert 1 (true) zurückgegeben, andernfalls der Wert 0 (false). Auf diese Art und Weise wurde der Name und der Schlüssel durch Reverse Engineering schnell ermittelt.

Möchte man diese Art des Reverse Engineerings weiter entwickeln, könnte man ein Programm schreiben, welches den vom *APK-Tool* dekodierten Dalvik-Bytecode zu Pseudocode oder gar zurück in Java-Code übersetzt.

Um sich vor Reverse Engineering zu schützen, gibt es für Java so genannte *Obfuscator*, welche aus einen gut les- und nachvollziehbaren Quelltext für Menschen eine schwerer lesbare Form generiert, die aber von der verarbeitenden Maschine etwa genauso schnell ausgeführt wird und die gleiche Funktionalität ausübt wie der Originalcode [Wik118]. Dem SDK hat Google *ProGuard* [Laf11] beigelegt, welches jedoch vom Entwickler noch aktiviert werden muss. Dazu muss die Zeile *proguard.config=proguard.cfg* in die *default.properties* Datei geschrieben werden. Im Idealfall wird der Code danach während des Build-Prozesses verschleiert. In meiner Entwicklungsumgebung kam es jedoch zu folgender Fehlermeldung: *Conversion to Dalvik format failed with error 1*. Dieser Fehler wurde durch die Datei *proguard.bat*, im Ordner der SDK-Tools liegend, verursacht. Ursache der Fehlermeldung war folgende Zeile innerhalb der Datei:

```
call %java_exe% -jar "%PROGUARD_HOME%"\lib\proguard.jar %*
```

Diese Zeile wurde zu Folgender geändert:

```
call %java_exe% -jar "%PROGUARD_HOME%"\lib\proguard.jar %1 %2 %3 %4 %5 %6 %7 %8 %9
```

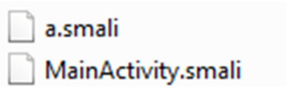


Abbildung 28: RE Beispiel Obfuscated

Anschließend lief der Build-Prozess ohne Fehlermeldung durch. Die Abbildung links zeigt den Output des APK-Tools von der neu erstellten APK. Der Klassenname *RegCheck* ist nicht mehr zu sehen. Stattdessen wurde daraus der bedeutungslose Name *a*. Auch der Dalvik-Bytecode hat sich verändert, so dass beispielsweise die Parameternamen *name* und *key* verschwunden sind.

Das komplette Beispiel befindet sich auf der CD im Order *nativ\Reverse Engineering*.

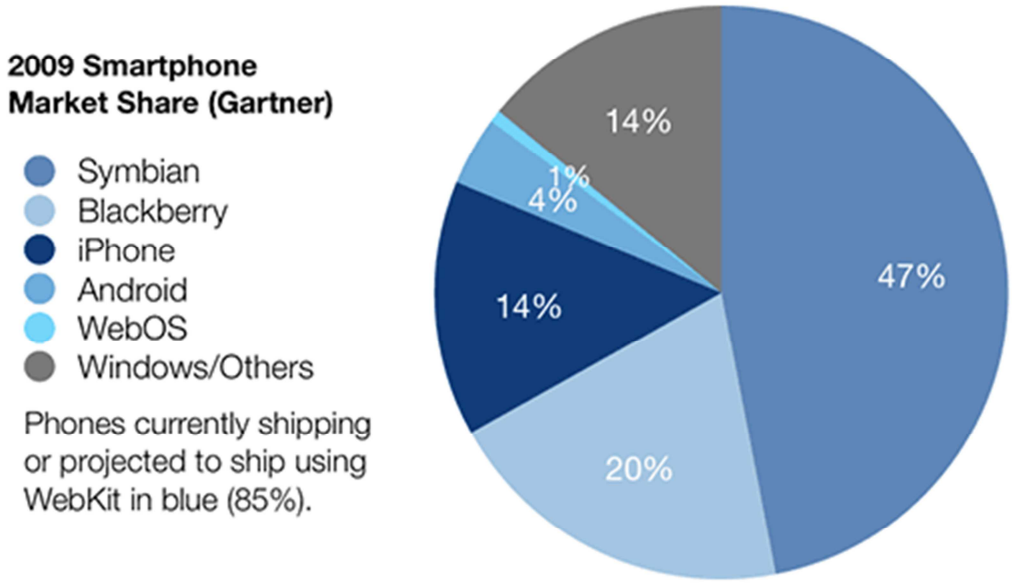
Abschließend sei noch bemerkt, dass es auch andere Ansätze zum Reverse Engineering unter Android gibt. Beispielsweise kann man auch den Cross-Compiler *dx* (siehe 4.1.4) zum Vorbild nehmen und einen Cross-Compiler in die andere Richtung entwickeln, d.h. von DEX zu JAR. Diesen Ansatz verfolgt das *dex2jar* Projekt [dex11]. Der Vorteil dieser Variante ist, dass es für Java schon eine Reihe von Decompilern gibt, die man nach dem Cross-Compiling nutzen kann.

6 Grundlagen webbasierter Techniken

Neben der nativen Entwicklung gibt es die Möglichkeit Applikationen mit Hilfe webbasierter Techniken zu entwickeln. Um diese Techniken verstehen zu können, dürfen die Grundlagen dafür nicht unerwähnt bleiben. Diese Grundlagen sind nicht Android-spezifisch und auf nahezu jedem mobilem Betriebssystem übertragbar (siehe 6.1). Besondere Vorsicht sollte man allerdings walten lassen, wenn man plant, seine mobile Webseite nicht nur mobil, sondern auch auf einem Desktop-Rechner anzubieten (siehe 6.2).

6.1 Webbrowser

Diese Arbeit stellt anhand von Android die Vor- und Nachteile der nativen und der webbasierten Entwicklung gegenüber. Dennoch soll das Ergebnis auf die wichtigsten mobilen Betriebssysteme übertragbar sein. Entscheidet man sich für eine native Entwicklung muss auf Grund unterschiedlicher Architekturen jede Applikation zu jedem Betriebssystem portiert werden, auf welchem man einen Benutzer erreichen möchte. Entscheidet man sich jedoch für eine webbasierte Entwicklung, ist eine Portierbarkeit der Applikationen wesentlich einfacher. Der Grund dafür ist, dass die Applikationen in einem Webbrowser laufen und nicht nativ.



Quelle: http://radar.oreilly.com/2010/05/mobile-operating-systems-and-b.html
Abbildung 29: 2009 Smartphone Market Share (Gartner)

Die oben aufgeführte Abbildung zeigt, dass *Android*, *iOS* (iPhone), *BlackBerry OS* und *Symbian* die wichtigsten mobilen Betriebssysteme sind. Diese vier machten 2009 85% aller verkauften Smartphones aus. Die Standardwebbrowser der genannten Systeme basieren alle auf WebKit und unterstützen HTML5. Die Erkenntnis dieser Statistik ist, dass das mobile Web in Richtung WebKit und HTML5 konvergiert. Eine mobile Webseite sollte daher in der Theorie auf den wichtigsten mobilen Betriebssystemen identisch angezeigt werden. Ein weiterer positiver Aspekt ist, dass die Desktop-Browser *Safari* und *Chrome* ebenfalls auf Webkit basieren [App11] [Goo11], wodurch auch dort die mobile Webseite ohne Anpassungen lauffähig sein sollte.

6.2 JavaScript

JavaScript ist eine Programmiersprache, welche hauptsächlich genutzt wird, um in Webseiten zusätzliche Funktionalitäten bereitzustellen. Seit dem Web 2.0 Paradigma werden interaktive Webseiten benötigt, in denen JavaScript eine große Rolle spielt. Mobile Webseiten und Web Applikationen sollen in der Regel ebenfalls mit dem Benutzer interagieren können. Auch einige neue HTML5 Features wie die *Geolocation API, Web Storage* oder *Canvas* benötigen den Gebrauch von JavaScript. Aus diesen Gründen stellt JavaScript eine Grundlage für die webbasierte Entwicklung dar.

JavaScript basiert auf dem Standard ECMA-262. Die letzte Spezifikation ist aus dem Jahr 2009 mit der JavaScript Version 1.8.5 [Wem11]. Nicht jeder Webbrowser unterstützt jede JavaScript Version [W3r11]. Alle wichtigen mobilen Webbrowser basieren auf Webkit (siehe 6.1), allerdings können die JavaScript-Engine und das Rendering getrennt kombiniert sein. Dennoch gibt es in der Regel keine Probleme in Hinblick auf die Portierbarkeit zwischen den unterschiedlichen mobilen Betriebssystemen. Möchte man eine Webseite jedoch sowohl auf einem mobilen Endgerät als auch auf einem Desktop-Rechner anbieten, sollte man sich nicht auf das Testen mit einem Webkit-basierten Browser beschränken, da es sonst zu Inkompatibilitäten kommen könnte, wodurch die Webseite beispielsweise im Internet Explorer, Markführer im Desktop-Segment [Man11], nicht funktionsfähig sein könnte.

6.3 HTML5

Die Hypertext Markup Language (HTML) ist eine Auszeichnungssprache zur Strukturierung von Inhalten, welche von einem Webbrowser dargestellt werden. Am 3. November 1992 erschien die erste Spezifikation von HTML [Wik23]. Inzwischen ist die Version 5 aktuell, welche gerade im Hinblick auf die mobile Entwicklung neue interessante Features mit sich bringt, auf die im Folgenden eingegangen wird. HTML5 ist noch kein Standard, sondern ein so genannter Arbeitsentwurf. Laut dem Zeitplan des W3C soll HTML5 2014 offiziell verabschiedet werden [W3C116].

6.3.1 Multimedia

Audio und Video konnten, bevor es HTML5 gab, nur mit Browser Plug-Ins abgespielt werden. Zwar gab es beispielsweise einen Tag namens *embed*, dieser wurde aber nie zum Standard, wodurch Plug-Ins nötig wurden, um sicher zu stellen, dass ein Benutzer unabhängig vom Browser die Musik hören oder das Video sehen konnte. Dabei hatten viele Firmen eine unterschiedliche Strategie, so dass es u.a. den Realplayer, Windows Media, QuickTime oder Adobe Flash gab. Zum einen ist es unkomfortabel, wenn ein Benutzer so viele Plug-Ins installiert haben muss, zum anderen unterstützen bis heute nicht alle Browser diese Standards. Beispielsweise ist es bis zum heutigen Tage nicht möglich im mobilen Safari Flashinhalte auszuführen. [Hog10]

Die Idee in HTML5 ist, dass ein Browser multimediale Inhalte wie Video und Musik nativ abspielen soll. Dafür werden die Tags *video* und *audio* bereitgestellt. Sollte ein älterer Browser kein HTML5 unterstützen, werden die im neuen Standard definierten Tags ignoriert, weshalb es in diesem Fall möglich ist, alternativ auf ein Plug-In zuzugreifen.

Im einfachsten Fall sieht ein *video*-Tag folgendermaßen aus:

```
<video src="myvideo.mp4"></video>
```

Dem Tag können weiterhin Attribute hinzugefügt werden. Mit *controls* werden dem Player bestimmte Buttons hinzugefügt, die es dem Benutzer möglich machen, das Video z.B. wiederholt abzuspielen. Das Attribut *preload* startet den Download eines Videos unmittelbar nach Aufruf der Webseite. Zusätzlich gibt es auch noch *loop*

und *autoplay*, die ein Video wiederholen oder automatisch starten. Auch die Breite und Höhe des Players kann definiert werden. Des Weiteren ist es innerhalb des *video*-Tags möglich, mehrere Quellen zu definieren. Bei der Verwendung der neuen Multimedia-Tags müssen die unterschiedlichen Container und Codecs für Video und Audio berücksichtigt werden, denn nicht jeder Browser unterstützt jedes Format. Daher ist es sinnvoll unterschiedliche Quellen mit unterschiedlichen Container und Codecs zu definieren [Wem11]. Um neben dem im Beispiel verwendeten MP4-Container auch *OGG* und *WebM* zu unterstützen, macht es Sinn, das Video in diese Formate zu konvertieren und ebenfalls als Quelle zu definieren. Das Beispiel sieht dann mit dem *control*-Attribut und mit Höhe und Breite versehen so aus:

```
<video width="320" height="240" controls>
<source src="myvideo.mp4" type="video/mp4" />
<source src="myvideo.ogv" type="video/ogg" />
<source src="myvideo.webm" type="video/webm" />
</video>
```

Mit Hilfe des *type*-Attributes kann man dem Browser noch genauere Details über das verlinkte Video übergeben.

Für den Audio-Tag stehen die gleichen Attribute wie für das *video*-Tag zur Verfügung [Wem11]. Ein Beispiel könnte folgendermaßen aussehen:

```
<audio controls>
<source src="myaudio.mp3"></source>
<source src="myaudio.ogg"></source>
</audio>
```

6.3.2 Data Storage

Das Speichern von Daten kann in bestimmten Anwendungen essentiell sein. Wollte man bisher Benutzerdaten speichern, so war dies entweder nur auf einem Server möglich, auf dem sich beispielsweise ein Benutzer einloggen muss, um die gespeicherten Daten zuordnen zu können. In dieser Variante wären die Daten dann nicht in der Kontrolle des Users. Oder die Daten wurden auf dem Client in Cookies gespeichert, die bei Entwicklern keine große Beliebtheit genießen [Law11].

In HTML5 gibt es neue Möglichkeiten Daten auf einem Client zu speichern, die im Folgenden vorgestellt werden.

6.3.2.1 Local Storage und Session Storage

Local Storage und Session Storage werden in der Literatur oft auch *Web Store* genannt, denn beide nutzen dasselbe *Storage()* Objekt Interface in JavaScript. Beide Techniken können als eine Verbesserung der Cookies angesehen werden. Abhängig von der Implementierung liegt die Größe der auf dem Client angelegten Datei bei maximal fünf bis zehn Megabytes. Die Daten werden im Gegensatz zu Cookies nicht automatisch an den Server gesendet. Des Weiteren greift das *Same-Origin-Policy* Sicherheitskonzept, wobei die Daten an die Domain gebunden sind, auf der die Web Anwendung läuft. [För11]

Der Unterschied zwischen Local Storage und Session Storage liegt darin, dass die Daten bei Verwendung von Local Storage persistent sind. Man könnte diese Variante mit den Cookies vergleichen, die ein Expiration Date haben, welches weit in der Zukunft liegt. Session Storage hingegen ist nicht persistent, d.h. die Daten werden nur für die jeweilige Session gespeichert und anschließend gelöscht. Schließt man das Browserfenster und öffnet es erneut, dann sind die Daten - im Gegensatz zu Local Storage - nicht mehr vorhanden. [För11]

Local Storage und Session Storage sind relativ einfach zu implementieren, was an einem Beispiel veranschaulicht werden soll. Auf der Webseite soll der Benutzer seinen Vor- und Nachnamen, sowie seinen Wohnort eingeben können. Diese Daten sollen persistent gespeichert und bei erneutem Aufruf der Seite automatisch angezeigt werden. Aus diesem Grund wird für die Implementierung Local Storage verwendet.

6.3.2.1.1 Beispiel

Die Webseite hat folgenden Aufbau:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html>
<html>
<head>
	<title>HTML5 Local Storage Beispiel</title>
	<link rel="stylesheet" type="text/css" href="main.css">
	<script language="javascript" src="helper.js"></script>
	<script language="javascript">
		// Local Storage Funktionalitäten müssen hier implementiert werden.
	</script>
</head>
<body>
<div id="content">
	<h1>HTML5 Local Storage Beispiel</h1>
	<div id="form">
		<form id="personForm">
		<table class="form">
			<tr><td class="label"> Vorname</td><td><input type="text" name="Vorname" /> </td></tr>
			<tr><td class="label">Nachname</td><td><input type="text" name="Nachname" /> </td></tr>
			<tr><td class="label">Wohnort </td><td><input type="text" name="Wohnort" /> </td></tr>
			<tr><td colspan="2" class="button">
			   <input id="formSubmit" type="button" value="Löschen" onclick="javascript:dbClear()" />
			  <input id="formSubmit" type="button" value="Hinzufügen" onclick="javascript:dbGo()" />
			</td></tr>
		</table>
		</form>
	</div>
	<div id="ergebnisse">
	<!-- Hier kommen die Ergebnisse hin -->
	</div>
</div>
</body>
</html>db
```

Das Beispiel nutzt – wie weitere HTML5 Beispiele in dieser Arbeit auch – eine *helper.js* und eine *main.css*. Die *helper.css* enthält JavaScript-Funktionen, mit denen man leicht eine Tabelle erstellen und manipulieren kann. Die *main.css* ist für das Design der Seite verantwortlich.

Mit Hilfe des Formulars kann der Benutzer seine Daten abschicken. Er hat die Möglichkeit Daten hinzuzufügen, wobei die *dbGo()* Funktion aufgerufen wird. Die Daten lassen sich mit der Funktion *dbClear()* löschen. Die eigentliche Implementierung der Local Storage Funktionen wird in JavaScript vorgenommen.

Zu allererst muss überprüft werden, ob der Webbrowser Local Storage unterstützt. Dazu überprüft man die Eigenschaft *localStorage* des window-Objekts. Wenn die Eigenschaft gefunden wird, wird sie der Variablen *db* zugewiesen, andernfalls wird eine Fehlermeldung ausgegeben:

```
var db = getLocalStorage() || dispError('Local Storage wird nicht unterstützt!');
```

```
function getLocalStorage() {
	try {
		if( !! window.localStorage ) return window.localStorage;
	} catch(e) {
		return undefined;
	}
}
```

Fügt der Benutzer seine Daten hinzu, wird die Funktion *dbGo() a*ufgerufen, in der die Formulardaten ausgelesen und gespeichert werden. Zum Speichern der Daten wird die Funktion *setItem(key, value)* genutzt. In diesem Beispiel sind die Schlüssel *Vorname, Nachname* und *Wohnort* und die Formulardaten deren Wert, der gespeichert werden soll. Ist ein Schlüssel noch nicht verfügbar, so wird er automatisch erzeugt. Sollte er allerdings schon vorhanden sein, wird er mit dem neuen Wert aktualisiert [W3C112].

Wird Local Storage nicht unterstützt, soll die Funktion nicht weiter ausgeführt werden:

```
function dbGo() {
	if(errorMessage) return;
	var f = element('personForm');
	db.setItem('Vorname', f.elements['Vorname'].value);
	db.setItem('Nachname', f.elements['Nachname'].value);
	db.setItem('Wohnort', f.elements['Wohnort'].value);
	dispResults();
}
```

Nachdem die Daten gespeichert wurden, sollen sie in einer Tabelle angezeigt werden, was durch einen Aufruf der Funktion *dispResults()* geschieht. Die Funktion erzeugt eine Tabelle, liest die gespeicherten Daten aus und schreibt das Ergebnis in den dafür vorgesehenen div-Tag. Das Auslesen aus dem *Local Storage* geschieht mit der Funktion *getItem(key).*

```
function dispResults() {
	if(errorMessage) {
		element('ergebnisse').innerHTML = errorMessage;
		return;
	}
	var t = new makeTable();
	t.addRow( ['Vorname', db.getItem('Vorname')] );
	t.addRow( ['Nachname', db.getItem('Nachname')] );
	t.addRow( ['Wohnort', db.getItem('Wohnort')] );
	element('ergebnisse').innerHTML = t.getTableHTML();
}
```

Sobald die Webseite beim nächsten Mal geladen wird, sollen die Informationen automatisch angezeigt werden, weshalb die Funktion *dispResults()* bei dem Ereignis *window.onload* aufgerufen wird:

```
window.onload = function() {
	dispResults();
}
```

Schließlich muss der Benutzer auch noch die Möglichkeit haben, seine Daten auch wieder zu löschen. Um alle Schlüssel und deren Daten zu löschen, kann man die Funktion *removeItem(key)* nutzen [W3C112]. Es gibt aber auch eine einfachere Methode namens *clear(),* bei der man die gesamten Daten mit nur einem Aufruf löschen kann, wodurch das Löschen jedes einzelnen Schlüssels überflüssig wird:

```
function dbClear() {
```

```
        if(errorMessage) return;
        db.clear();
        dispResults();
}
```

Der Unterschied zwischen Local Storage und Session Storage in der Implementierung ist marginal. Das *localStorage* Attribut wurde bei diesem Return an die Variable *db* gebunden:

```
if( !! window.localStorage ) return window.localStorage;
```

Würde man nun Session Storage einsetzen wollen, müsste man lediglich das Attribut zu *sessionStorage* ändern:

```
if( !! window.sessionStorage ) return window.sessionStorage;
```

Das komplette Beispiel ist weiterhin lauffähig und funktioniert problemlos. Der schon anfangs erwähnte Unterschied ist, dass die Daten jetzt nicht mehr persistent gespeichert werden. Schließt man den Browser und ruft die Seite erneut auf, dann sind die Daten nicht mehr vorhanden. Auch gibt es pro Tab im Browser eine unterschiedliche Session. D.h. ruft man das Beispiel in zwei oder mehr Tabs auf, so kann man pro Tab unterschiedliche Daten speichern. Bei Local Storage hingegen sind die Daten pro Tab identisch. Durch diesen kleinen Unterschied in der Implementierung erreicht man unterschiedliche Funktionalitäten. Dies macht eine Implementierung von Web Storage sehr flexibel.

Das komplette Beispiel befindet sich als Local Storage und als Session Storage Variante auf der CD im Ordner *webbasiert\HTML5*.

6.3.2.2 Web SQL Database

Web SQL Datenbanken sind eine weitere Möglichkeit Daten lokal beim Benutzer zu speichern. Wie der Name impliziert, handelt es sich dabei um eine echte Datenbank, bei der SQL-Queries und Joins möglich sind. Wenn man mit SQL vertraut ist, findet man sich mit der Web SQL Database API sehr schnell zurecht. Web SQL nutzt für die Speicherung *SQLite* [Law11]. Momentan ist die SQLite Datenbank Engine die Einzige, welche man nutzen kann, daher nutzen alle Web SQL Implementierungen SQLite [W3C113]. Dies ist die Stärke und zugleich größte Schwäche von Web SQL. Auf der einen Seite muss sich der Programmierer keine Gedanken über unterschiedliche SQL-Dialekte machen, weshalb keine eventuellen Anpassungen oder Fallunterscheidungen im Code gemacht werden müssen. Auf der anderen Seite hat sich das *World Wide Web Consortium* (W3C), ein Gremium zur Standardisierung der das World Wide Web betreffenden Techniken [Wik111], dazu entschieden, dass man keinen Standard für nur einen SQL-Dialekt verabschieden möchte, weswegen sie die Arbeit für eine Spezifikation der Web SQL eingestellt haben, mit der Begründung, dass mehrere unterschiedliche und unabhängige Implementierungen für eine Spezifikation von Nöten wären [W3C113].

Weil Datenbankabfragen einige Zeit dauern können, ist die Database API asynchron, weshalb man bei allen Methoden eine Callback-Funktion angeben kann. Die Asynchronität muss bei der Entwicklung berücksichtigt werden, damit es nicht zu einem ungewünschten Verhalten im Ablauf des Codes kommt. Jedoch werden die SQL Befehle in einer Warteschlange ausgeführt, so dass man sich sicher sein kann, dass z.B. eine Tabelle schon vorhanden ist, bevor man auf diese Abfragen ausführt [Law11].

6.3.2.2.1 Beispiel

In dem Beispiel sollen mehrere Personen und deren Wohnort gespeichert werden. Der Aufbau der Seite ist identisch mit Beispiel zu Local Storage und Session Storage, allerdings soll nun mit einer Web SQL Datenbank gearbeitet werden.

Eine Datenbank wird folgendermaßen angelegt:

```
function prepareDatabase() {
        var odb = getOpenDatabase();
        if(!odb) {
                dispError('Web SQL wird nicht unterstützt');
                return undefined;
        } else {
                var db = odb( 'Beispiel', '1.0', 'Eine Beispieldatenbank', 2 * 1024* 1024 );
                db.transaction(function (t) {
                t.executeSql( createSQL, [], function(t, r) {}, function(t, e) {
                        alert('Erstelle Tabelle: ' + e.message);
                });
        });
        return db;
        }
}
```

Die *getOpenDatabase()* Funktion überprüft anhand des Window-Objektes und der *openDatabase* Methode, ob Web SQL vom Browser unterstützt wird. Ist dies nicht der Fall, wird eine Fehlermeldung ausgegeben. Andernfalls wird die Datenbank geöffnet, wobei für diese ein Namen, eine Versionsnummer und eine Beschreibung definiert wird. Weiterhin muss eine Größe der Datenbank angegeben werden. Die Datenbank wird allerdings nicht mit der definierten Größe angelegt. Vielmehr beschreibt der Parameter die maximal zulässige Größe. Alle Operationen einer Web SQL geschehen innerhalb einer Transaktion [W3C113], weshalb auf diese Weise auch eine Datenbank angelegt wird. Es wird eine anonyme Funktion genutzt, die SQL ausführt, wobei der t-Parameter in dem Beispiel das Transaktions-Objekt ist. Der *executeSql*-Methode können vier Parameter übergeben werden. Der erste Parameter ist der eigentliche SQL-Befehl, welcher im Beispiel unter der Variable *createSQL* hinterlegt ist:

```
var createSQL = 'CREATE TABLE IF NOT EXISTS PersonDB (' +
                 'id INTEGER PRIMARY KEY,' +
                 'vorname TEXT,' +
                 'nachname TEXT,' +
                 'wohnort TEXT' +
                ')';
```

Es wird eine Tabelle namens *PersonDB* mit den Spalten *id, vorname, nachname* und *wohnort* angelegt, falls sie nicht existiert.

Mit dem zweiten Parameter können Argumente übergeben werden. Wurde die Operation erfolgreich ausgeführt, wird die im dritten Parameter definierte Callback Funktion ausgeführt, andernfalls wird die Error-Callback Funktion im vierten Parameter aufgerufen. Im letzten genannten Fall wird die Fehlermeldung im Browser ausgegeben.

Möchte man Einträge der Tabelle hinzufügen, selektieren, editieren oder löschen, so geschieht dies auf die gleiche Art und Weise. Die Implementierung gestaltet sich sehr intuitiv, wenn ein Entwickler der SQL-Sprache mächtig ist.

Das ausführbare Web SQL Beispiel befindet sich im Ordner *webbasiert\HTML5* auf der beiliegenden CD.

6.3.2.3 Indexed Database API

Die Indexed Database API (Indexed DB) ist ein von Oracle im Jahr 2009 vorgeschlagene Programmierschnittstelle für eine Datenbank, deren Datensätze einfache Werte und hierarchische Objekte beinhalten kann [Wik112]. Ein Datensatz besteht aus einem Schlüssel und einem dazugehörigen Wert. Über alle gespeicherten Datensätze pflegt die Datenbank Indizes. Zum Adressieren des Datensatzes muss ein Entwickler entweder den Schlüssel oder den Index nutzen. Im Vergleich zu einer klassischen Datenbank ist es nicht möglich, direkt nach einem Wert zu suchen. Des Weiteren gibt es keine einfache Möglichkeit Datensätze zu verbinden, wie es bei einer klassischen Datenbank mit einem Join-Befehl möglich wäre. Die Implementierung ist aufwendiger, da unterschiedliche Browser auch unterschiedliche Methoden des window-Objektes nutzen. Mozilla nutzt z.B. *window.mozIndexedDB* statt *window.indexedDB*. [W3C114]

An einem Beispiel soll an dieser Stelle verzichtet werden, da die Indexed Database API derzeit in den mobilen Browsern keine Rolle spielt. Sie wird derzeit weder von iOS Safari, noch vom Android Browser oder anderen Drittherstellern wie Opera Mini oder Opera Mobile in den aktuellen Versionen unterstützt [Dev11].

6.3.3 Offline Application Cache API

In HTML5 können webbasierte Anwendungen mit Hilfe der *Offline Application Cache API* offline verfügbar gemacht werden. Die Vorteile der API sind im Wesentlichen die Folgenden:

- *Offline Browsing*: Ein Anwender kann die Seite nutzen und innerhalb dieser navigieren, selbst wenn er offline ist.
- *Geschwindigkeit*: Alle Ressourcen können lokal gecached werden. Der Zugriff auf den Cache ist schneller als das Laden der Ressourcen über ein Netzwerk.
- *Reduzierter Traffic*: Die Applikation bzw. der Browser lädt nur Ressourcen vom Server, die sich verändert haben, wodurch sich der Traffic und die Serverauslastung verringert.

Die *Offline Application Cache API* erlaubt es Entwicklern die Dateien, die gecached werden sollen, genau zu spezifizieren. Die Applikation läuft selbst dann korrekt, wenn der Anwender den Refresh-Button betätigt, während er offline ist. Die Spezifizierung des Caches geschieht mittels einer Manifest-Datei (MIME-Type „text/cache-manifest“), welche im HTML-Tag über das Attribut *manifest* referenziert wird:

```
<!DOCTYPE HTML>
<html manifest="/beispiel.appcache">
[...]
</html>
```

Damit der richtige MIME-Type des Manifests erkannt wird, ist es unter Umständen von Nöten, die *.htaccess* Konfiguration des Servers zu verändern. Ein Apache-Server erkennt den Dateityp *.appcache* als korrekten MIME-Type, wenn folgende Zeile in die Konfiguration hinzugefügt wird:

```
AddType text/cache-manifest .appcache
```

Eine Manifest-Datei kann aus den drei Teilen *CACHE, NETWORK* und *FALLBACK* bestehen. Die unter *CACHE* stehenden Dateien werden gecached, während die unter *NETWORK* stehenden Dateien eine Netzwerkverbindung brauchen und vom Cache umgangen werden. Unter *FALLBACK* werden Dateien definiert, die statt der eigentlichen Datei aufgerufen werden, wenn diese nicht verfügbar ist.

Ist eine Anwendung erst einmal gecached, verbleibt sie solange in diesem Zustand bis einer der folgenden Fälle eintritt:

- Der Anwender löscht den Cache der Seite über seinen Browser.
- Die Manifest-Datei wird geändert.
- Der Application Cache wird programmatisch geupdated.

Für den programmatischen Ansatz wird das *window.applicationCache* Objekt verwendet. Mit der Eigenschaft *status* lässt sich der momentane Zustand des Caches auslesen. Beispielsweise beschreibt *window.applicationCache.UNCACHED*, dass die Elemente bisher nicht gecached wurden, während bei *window.applicationCache.UPDATEREADY* ein Update des Caches vorliegt. Der Entwickler kann mit der Funktion *update()* einen neuen Cache befüllen, dann mit *UPDATEREADY* überprüfen, ob dies erfolgreich war, um anschließend mit *swapCache()* den alten Cache mit den neuen Cache zu ersetzen:

```
var cache = window.applicationCache;
cache.update(); // Der Cache soll geupdated werden
[...]
if (cache.status == cache.UPDATEREADY) {
        cache.swapCache();  // Wenn das Update erfolgreich war, soll der neue Cache den alten ersetzen
}
```

Weiterhin ist es auch möglich *EventListener* für jeden Zustand hinzuzufügen. Zum Beispiel könnte die Funktion *handleCacheDownload(event)* aufgerufen werden, sobald der Browser mit dem Download des neuen Caches beginnt:

```
var cache = window.applicationCache;
function handleCacheDownload(event) {
        […]
}
cache.addEventListener('DOWNLOADING', handleCacheDownload, false);
```

6.3.4 Geolocation API

Die Geolocation-Funktion in HTML5 erlaubt den Zugriff auf die aktuelle Position des Gerätes eines Benutzers. Die Positionsangabe muss jedoch nicht zwangsläufig korrekt sein, da diese nicht nur via GPS, sondern beispielsweise auch via WiFi, IP-Adresse, GSM/CDMA Zelleninformationen oder auch manuell von einem Benutzer hinterlegt werden kann [W3C11].

Mobile Geräte haben häufig Ortungsdienste wie GPS oder einen Kompass (siehe 4.5), weshalb die Geolocation API überwiegend in mobilen Anwendungen benutzt wird.

Aktivierte Ortungsdienste eines mobilen Endgerätes verursachen in der Regel einen erhöhten Stromverbrauch. Die Geolocation API bietet deshalb unterschiedliche Konfigurationsmöglichkeiten hinsichtlich Genauigkeit, Timeout und Cachedauer an, wodurch der Strombedarf beispielsweise durch eine längere Cachedauer verringert werden kann [Hol11].

Ortungsdaten bzw. Positionsangaben eines Benutzers sind persönlich. Um die Privatsphäre nicht zu kompromittieren, muss die Geolocation API so implementiert werden, dass keine Daten ohne die Zustimmung des Nutzers verfügbar sind. Insbesondere sollte für einen Anwender auch ersichtlich sein, wofür die Daten genutzt werden. Werden die Positionsangaben gespeichert, muss der Anwender die Möglichkeit haben, diese Daten selbstständig löschen zu können [W3C11].

Die genannten Punkte sollen anhand eines Beispiels verdeutlicht werden.

6.3.4.1 Beispiel

Das Beispiel soll eine Webseite darstellen, auf der mit Hilfe der Geolocation API Breitengrad und Längengrad ausgegeben werden. Des Weiteren sollen diese Werte bei einer Veränderung der Position aktualisiert werden, wobei bei jeder Änderung ein Zeitstempel ausgegeben werden soll. Der Benutzer soll die Möglichkeit haben, die Positionsbestimmung zu starten und zu stoppen.

Der Aufbau der Webseite ist wie folgt:

```
<!DOCTYPE html>
<html>
<head>
        <title>Geolocation API Beispiel</title>
        <link rel="stylesheet" type="text/css" href="main.css" />
        <script type="text/javascript" src="helper.js"></script>
        <script type="text/javascript">
            // Funktionen der Geolocation API müssen hier implementiert werden
        </script>
</head>
<body>
<div id="content">
        <h1>Geolocation API Beispiel</h1>
        <p class="message" id="statusMessage"/>
        <div id="ergebnisse">
            <!-- Hier kommen die Ergebnisse hin -->
        </div>
        <form>
            <input id="startStop" type="button" value="Start" onclick="startStopWatching()" />
        </form>
</div>
</body>
</html>
```

Die Seite wird mit Hilfe der *main.css* formatiert. Weiterhin wird die schon aus den anderen Beispielen bekannten *helper.js* importiert. Im Tag mit der id *statusMessage* soll ausgegeben werden, ob die Ortungsbestimmung gerade aktiv ist oder nicht. Längen-, Breitengrad und Zeitstempel sollen im Tag mit der id *ergebnisse* ausgegeben werden. Mit Hilfe des Formulars kann der Benutzer die Positionsbestimmung starten und stoppen.

Um die Applikation initialisieren zu können, wird die *window.onload* Funktion genutzt, welche die Funktion *init()* aufruft, in welcher mit Hilfe der Funktion *getGeoLocation()* geprüft wird, ob die die Geolocation API überhaupt zur Verfügung steht, was anhand des Objektes *navigator.geolocation* realisiert werden kann. Steht das Objekt nicht zur Verfügung, wird eine Fehlermeldung ausgegeben. Andernfalls wird eine Tabelle generiert, in der die späteren Ergebnisse ausgegen werden:

```
var t = new makeTable();
var watchID;
var geo;
function getGeoLocation() {
        try {
            if(!!navigator.geolocation ) return navigator.geolocation;
            else return undefined;
          } catch(e) {
            return undefined;
          }
}
```

```
function init() {
        if((geo = getGeoLocation())) {
          t.setHeader( [ 'Breitengrad', 'Längengrad', 'Zeitstempel' ] );
          t.addRow( [ ' ', ' ', ' ' ] );
        } else {
          statusMessage('HTML5 Geolocation API wird nicht unterstuetzt')
        }
        dispResults();
}

window.onload = function() {
        init();
}
```

Steht das Objekt zur Verfügung, hat der Benutzer die Möglichkeit die Positionsbestimmung zu starten. Da eine Positionsbestimmung, wie bereits erwähnt, in die Privatsphäre des Benutzers eingreift, muss dieser beim Klick auf den Start-Button dem Dokument ausdrücklich eine Erlaubnis erteilen:

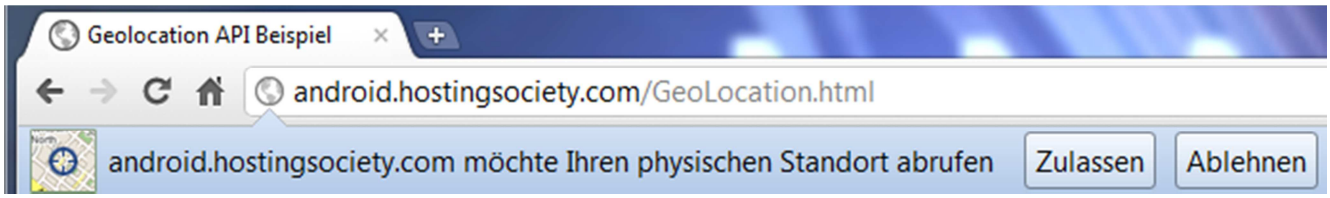


Abbildung 30: Geolocation Erlaubgsnisanfrage

Diese Erlaubnis wird vom Webbrowser gespeichert, kann aber nachträglich vom Benutzer entfernt bzw. abgelehnt werden, wodurch er seine Privatsphäre schützen kann. Was der Benutzer aber nicht weiß ist, wie seine Daten verarbeitet werden. Wenngleich im Beispiel die Informationen direkt ausgegeben werden und dadurch ersichtlich ist, warum die Berechtigung benötigt wird, könnte man beispielsweise die Position des Anwenders auch an einen anderen Server schicken, wodurch die Kontrolle der Daten nicht mehr beim Nutzer läge.

Wurde die Berechtigung vom Benutzer gewährt, erhält die Funktion *watchPosition* vom Gerät die aktuelle Position. Sobald das Ergebnis erfolgreich abgefragt wurde, wird es an die Callback Funktion *zeigeKoordinaten()* übergeben. Bei einem Fehler wird die Callback Funktion *error()* aufgerufen. Weiterhin ist es möglich, die Attribute *enableHighAccuracy, maximumAge* und *timout* als drittes Argument der Funktion *watchPosition()* zu übergeben. *enableHighAccuracy* beeinflusst die Genauigkeit, so dass bei einer hohen Genauigkeit (Attributwert = true) das GPS, sofern vorhanden, gestartet wird, woraus ein höherer Stromverbrauch resultiert. Bei einer niedrigen Genauigkeit hingegen wird weniger Strom verbraucht, indem die Position aus anderen Quellen wie z.B. WiFi bezogen wird. Es ist möglich, dass die Position gecached wird. Zuständig dafür ist das Attribut *maximumAge,* dessen Wert in Millisekunden angegeben wird. Erst wenn die definierte Zeit abgelaufen ist, wird eine neue Position bestimmt. Da eine Position aus diversen Gründen auch nicht bestimmbar sein kann, ist es sinnvoll das *timeout* Attribut zu definieren, dessen Wert ebenfalls in Millisekunden definiert wird. Wird dieser Wert überschritten, wird der Error Callback, in dem Fall die Funktion *error()* aufgerufen: [W3C11]

```
function startStopWatching() {
        startStopElement = element('startStop');
        if(startStopElement.value == 'Start') {
          watchID = geo.watchPosition(zeigeKoordinaten, error, {
            enableHighAccuracy: true,
            maximumAge: 1000, // eine Sekunde
            timeout: 60000 // eine Minute
          });
```

```
            startStopElement.value = 'Stop';
        } else {
            stopWatching();
            statusMessage('Stopped.');
            startStopElement.value = 'Start';
        }
}
```

Ist eine Abfrage der aktuellen Position erfolgreich, wird die Callback Funktion *zeigeKoordinaten(position)* mit einem Positions-Objekt aufgerufen. Dieses Positions-Objekt besitzt wiederum ein Koordinaten-Objekt, welches u.a. die Eigenschaften *latitude* (Breitengrad) und *longitude* (Längengrad) hat, die in diesem Beispiel wichtig sind. Nun müssen diese Eigenschaften lediglich in die Zeile der angelegten Tabelle eingetragen bzw. aktualisiert werden. Dazu kann die Methode *toString()* genutzt werden. Des Weiteren wird der aktuelle Zeitstempel ausgegeben:

```
function zeigeKoordinaten(position) {
        d = new Date();
        var lat = position.coords.latitude;
        var lon = position.coords.longitude;
        t.updateRow(0, [ lat.toString(), lon.toString(), d.toLocaleString()] );
        dispResults();
}
```

Im Webbrowser sieht das Ergebnis folgendermaßen aus:

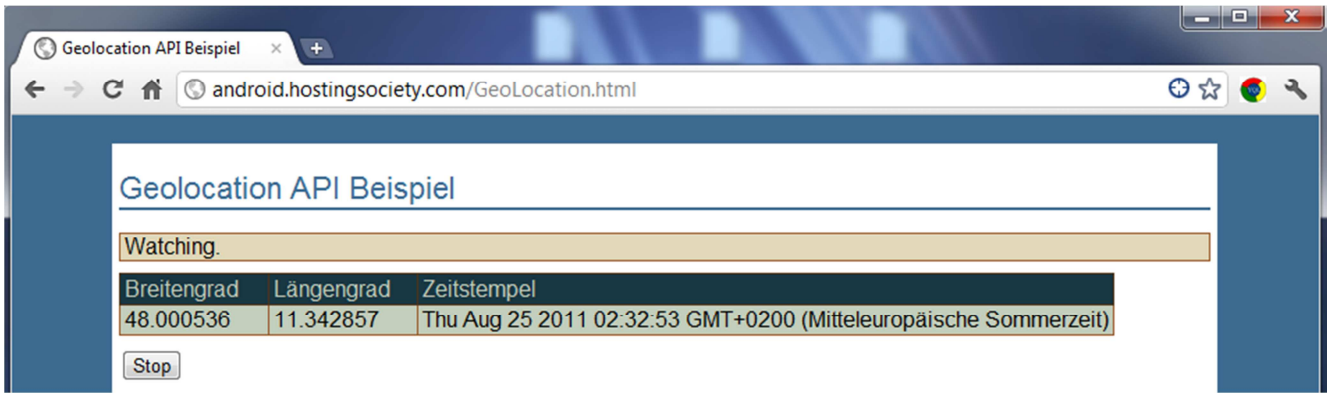


Abbildung 31: Ausgeführtes Geolocation Beispiel

Die Abbildung 31 zeigt Googles Chrome, welches auf einem unbeweglichen Rechner installiert ist, weshalb die Werte der Tabelle statisch erscheinen. Wird das Beispiel auf einem mobilem Gerät ausgeführt, so aktualisieren sich die Werte *Breitengrad, Längengrad* und *Zeitstempel* bei einer Positionsänderung sekündlich.

Tritt allerdings ein Fehler während der Positionsbestimmung auf, wird die Callback Funktion *error()* mit einem Error-Objekt aufgerufen. Das Error-Objekt hat ein Attribut *code*, in dem der Fehlercode gespeichert wurde. Momentan gibt es die Fehlercodes *TIMEOUT, POSITION_UNAVAILABLE* und *PERMISSION_DENIED.* Während bei einem *TIMEOUT* die definierte Timeout-Zeit überschritten wurde und die Position nicht schnell genug ermittelt wurde, konnte diese bei dem Fehlercode *POSITION_UNAVAILABLE* überhaupt nicht bezogen werden. Tritt allerdings der Fehler *PERMISSION_DENIED* auf, dann hat das Dokument nicht die Rechte, die es benötigt, um auf die Geolocation API zugreifen zu können [W3C11]:

```
function error(error) {
        stopWatching();
        switch(error.code) {
```

```
            case error.TIMEOUT:
                alert ('Geolocation API: Timeout');
                break;
            case error.POSITION_UNAVAILABLE:
                alert ('Geolocation API: Position nicht verfuegbar');
                break;
            case error.PERMISSION_DENIED:
                alert ('Geolocation API: keine Berechtigung');
                break;
            default:
                alert('Geolocation API - unbekannter Fehler: ' + error.code);
        }
}
```

Sollten der Geolocation API weitere Fehlercodes implementiert werden, würden sie in diesem Beispiel mit dem default case abgefangen und ausgegeben werden.

Das komplette Beispiel inklusive der hier nicht erläuterten Funktionen wie beispielsweise *dispResults()* befindet sich auf der beiliegenden CD im Verzeichnis *webbasiert\HTML5*.

6.3.5 Canvas

Canvas kommt aus dem englischen und bedeutet auf Deutsch *Leinweind*. Auf solch einer Leinwand kann man mit JavaScript dynamisch Bitmap-Grafiken malen. Das Canvas-Element ist Teil des werdenden HTML5-Standards [W3C111], wodurch kein zusätzliches Plug-In nötig ist, wie es z.B. bei Adobe Flash oder Microsoft Silverlight der Fall ist. Die Möglichkeiten des Canvas-Elements sind so weit gefasst, dass im Verbund mit JavaScript ganze Spiele programmiert werden können [Ful11].

HTML5 Canvas stellt 2D *rendering contexts* zur Verfügung. Vergleichen kann man dies mit Ebenen, auf welchen man zeichnen kann.

6.3.5.1 Beispiel

Ein Beispiel soll die Möglichkeiten von HTML5 Canvas im Kleinen verdeutlichen. In dem Beispiel soll auf einer Webseite ohne jede Hilfe von Plug-Ins ein Ball gezeichnet werden, welcher kreisförmig rotiert. Der Aufbau der Seite ist wie gefolgt:

```
<!doctype html>
<head>
<title>Canvas Beispiel</title>
<script type="text/javascript">
        // Funktionen des Canvas-Elements müssen hier implementiert werden
</script>
</head>
<body>
        <canvas id="canvas" width="500" height="500">
                HTML5 Canvas wird von Ihrem Webbrowser nicht unterstützt.
        </canvas>
</body>
</html>
```

Das Canvas-Element erhält eine Maße und eine ID zur Steuerung, sowie einen Fallback-Content, welcher ausgegeben wird, falls HTML5 Canvas vom Webbrowser nicht unterstützt wird. Die eigentliche Funktionalität wird mit JavaScript implementiert:

```
window.addEventListener('load', canvasBeispiel, false);
function canvasBeispiel() {
	function zeichneScreen() {
		context.fillStyle = '#EEEEEE';
		context.fillRect(0, 0, leinwand.width, leinwand.height);
		ball.x = kreis.centerX + Math.cos(kreis.winkel) * kreis.radius;
		ball.y = kreis.centerY + Math.sin(kreis.winkel) * kreis.radius;
		kreis.winkel += ball.geschwindigkeit;
		context.fillStyle = "#000000";
		context.beginPath();
		context.arc(ball.x,ball.y,15,0,Math.PI*2,true);
		context.closePath();
		context.fill();
	}
	var kreis = {centerX:250, centerY:250, radius:125, winkel:0}
	var ball = {x:0, y:0,geschwindigkeit:0.1};
	leinwand = document.getElementById('canvas');
	context = leinwand.getContext('2d');
	setInterval(zeichneScreen, 40);
}
```

Sobald der Webbrowser die Seite geladen hat, wird die Funktion *canvasBeispiel()* aufgerufen. In dieser Funktion wird ein Ball mit x- und y-Koordinaten definiert. Zusätzlich erhält er ein Attribut *Geschwindigkeit*, damit die Rotiergeschwindigkeit beeinflusst werden kann. Um diesen Ball kreisförmig zu rotieren, wird ein Kreis mit der Center-Position 250,250 und dem Radius 150 definiert. Der Kreis selbst wird nicht auf die Leinwand gezeichnet, sondern wird als Pfad für den rotierenden Ball benötigt. Dieser soll bei einem Winkel von 0 rad starten. Die Variable *leinwand* holt das Canvas-Element anhand seiner ID. Man zeichnet jedoch nicht auf das Element selbst, sondern auf einem Kontext des Elementes, welchen man mit dem Aufruf *leinwand.getContext('2d')* bekommt.

Nachdem die benötigten Variablen definiert sind, wird die Funktion *zeichneScreen()* im Intervall von 40ms aufgerufen, um eine Animation zu erzeugen. Mit Hilfe der Cosinus- und Sinusfunktion ist es bei bekanntem Kreisradius und dessen Centerposition möglich, die x- und y-Positionen des Balls zu bestimmen. Danach wird der Winkel des Kreises verändert, wodurch der Ball beim nächsten Aufruf von *zeichneScreen()* sich entlang des Kreispfades bewegt. Schließlich wird der Ball noch auf den Kontext gezeichnet:

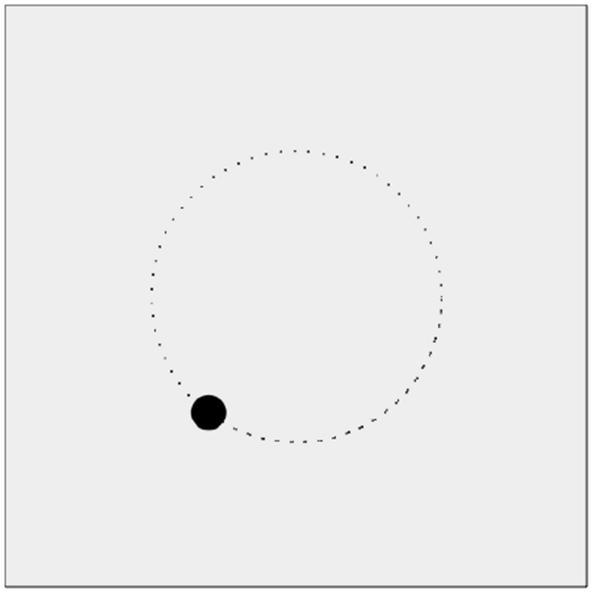


Abbildung 32: HTML5 Canvas Beispiel

Bei der Abbildung ist zu beachten, dass die Punkte im Webbrowser nicht zu sehen sind. Sie sollen in dem Bild lediglich den Kreispfad illustrieren.

Das ausführbare Beispiel befindet sich auf der CD im Ordner *webbasiert\HTML5*.

6.4 CSS3

Cascading Style Sheets (CSS) ist eine deklarative Sprache für Stilvorlagen von strukturierten Dokumenten, welches vor allem zusammen mit HTML verwendet wird. Die Grundidee von CSS ist, dass der Inhalt und dessen Darstellung voneinander getrennt werden. So steht in einem HTML-Dokument der eigentliche Inhalt, während mittels CSS weitgehend unabhängig davon die konkrete Darstellung definiert wird. In einem HTML-Dokument können einzelne Elemente anhand ihres Namens, ihrer ID oder ihrer Position identifiziert werden. CSS ermöglicht es für jedes Element bestimmte Darstellungsattribute festzulegen [Wik113]. Die Regeln für die konkrete Darstellung können direkt im Dokument, aber auch in einer separaten Datei, definiert werden. Letzteres erhöht die Wiederverwendbarkeit für andere Dokumente. Durch die Trennung von Inhalt und dessen Darstellung ist es möglich, dass für verschiedene Ausgabemedien unterschiedliche Darstellungen festgelegt werden. Beispielsweise hat ein mobiles Gerät in der Regel eine geringere Auflösung im Vergleich zu einem Desktop-System. Möchte man ein HTML-Dokument sowohl auf einem mobilen Gerät, als auch auf einem Desktop-System anbieten, so kann man mit Hilfe von CSS auf die unterschiedlichen Eigenheiten und Bedürfnisse eingehen, ohne dass der Inhalt im Dokument verändert werden muss.

Eine mobile Anwendung muss übersichtlich gestaltet sein. Darüber hinaus sollen die Designelemente so aussehen und sich die Anwendung so flüssig anfühlen, als wenn sie zum System gehöre. Das heißt, in der Regel soll eine webbasierte Anwendung für einen Benutzer nicht als solche zu erkennen sein. CSS stellt bei den webbasierten Techniken einen wesentlichen Faktor dar, denn de facto wird damit die grafische Oberfläche definiert.

CSS3 ist eine Erweiterung von CSS 2.1, welche viele interessante neue Features mit sich bringt. Es gibt für CSS3 nicht nur eine einzige Spezifikation. Stattdessen wurde CSS in viele kleine Module getrennt, die einzeln spezifiziert werden. Jedes dieser Module hat eigene Autoren und eine eigene Roadmap, die man auf der Seite des W3Cs einsehen kann [W3C115]. Der Vorteil dieser Methode gegenüber einer einzelnen Spezifikation ist, dass die Entwicklung von CSS3 nicht nur wegen eines Modules stagniert. Ein Entwickler kann auf eine stetige Weiterentwicklung einzelner Module zurückgreifen.

Im Folgenden sollen wesentliche Neuerungen von CSS3 kurz vorgestellt werden. Die für Googles Chrome bzw. WebKit optimierten Beispiele dazu befinden sich auch auf der CD im Ordner *webbasiert\CSS3*.

Abbildung 33: CSS3 Kreis

In CSS3 gibt es eine Menge neuer Eigenschaften, die es erlauben visuelle Effekte zu erzeugen, die man bisher nur mit Bildern darstellen konnte. Zu diesen Effekten zählen u.a. abgerundete Ecken, Schlagschatten, semitransparente Hintergründe und Farbverläufe.

Das Beispiel links zeigt einen mit CSS3 erstellten Kreis mit Schlagschatten und Farbverlauf.

HTML Inline- oder Blockelemente werden mit der Eigenschaft *border-radius* in einem definierten Radius abgerundet. Zudem können mit z.B. *border-top-right-radius* bestimmte Kanten, in diesem Fall die obere rechte Kante, gezielt angesprochen werden. Ein Kreis entsteht, wenn alle vier Kanten in einem Radius von der Hälfte der Gesamtbreite des Elements

abgerundet werden. Die Eigenschaft *box-shadow* erzeugt den Schlagschatten und *gradient* den Farbverlauf [Gil10].

Div-Container und andere Inline- oder Blockelemente lassen sich mit CSS3 schnell und einfach drehen, neigen und verzerren, um dreidimensionale Körper zu simulieren. Diese Effekte werden Transformationen genannt und sind im 2D-Transformations- und 3D-Transformationsmodul spezifiziert.

Um den in der Abbildung zu sehenden Würfel zu erzeugen sind die Attribute *transform: skewY(-30deg), transform: rotate(60deg)* und *transform:scaleY(1.16)* notwendig.

SkewY neigt einen Div-Container im angegebenen Winkel, wobei Y oder X hierbei die Richtung bestimmen. *Rotate* rotiert Selbigen im angegebenen Winkel und *scaleY* skaliert den Div-Container um den definierten Wert.

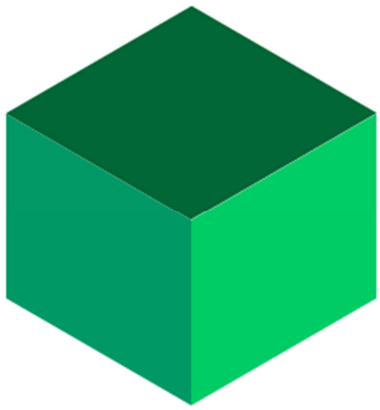


Abbildung 34: CSS3 Würfel

Mit dem Font-Modul wird in CSS3 die *@font-face* Regel eingeführt, welche es ermöglicht, auf Server liegende Schriftarten einzubinden. Bisher war ein Entwickler auf die Schriftarten limitiert, die auf dem Client verfügbar sind.

Des Weiteren gibt es in CSS3 neue Selektoren, mit denen HTML-Bereiche selektiert werden können, ohne den Tags IDs oder Klassen hinzuzufügen [Gil10].
Ein Anwendungsbeispiel: Auf einer Seite werden Dokumente als PDF- und Word-Dateien zum Download angeboten. Der Benutzer soll anhand eines Icons auf einem Blick erkennen, um welches Format es sich handelt. Zunächst wird eine CSS-Regel definiert, welche sicherstellt, dass für ein PDF- oder Word-Icon genügend Platz vorhanden ist und dieses auch nicht abgeschnitten wird. Zudem soll das Icon auch nur einmalig angezeigt werden:

```
ul a {
        display: block;
        min-height: 15px;
        padding-left: 20px;
        background-repeat: no-repeat;
        background-position: 0 3px;
}
```

Um alle Links zu finden, die Entweder ein PDF- oder ein Word-Dokument enthalten, wird ein in CSS3 neu eingeführter Attribut-Selektor benutzt:

```
a[href$=".pdf"] {
        background-image: url(images/icon_pdf.png);
}

a[href$=".doc"] {
        background-image: url(images/icon_doc.png);
}
```

Der Browser findet mit Hilfe dieser Selektoren alle Elemente, die auf *.pdf* oder *.doc* enden. Wird ein Element gefunden, dann wird das jeweilige Icon als Hintergrundbild definiert und ein Benutzer weiß auf einen Blick, um welches Format es sich handelt.

Weitere wesentliche neue Features in CSS3 sind *Transitions* und *Animationen*. Mit Transistions sind fließend animierte Übergänge möglich. Beispielsweise hatte bisher ein Link in jedem Zustand nur eine einzige Farbe und Übergänge waren nicht möglich. Auch Slide-Effekte benötigten JavaScript. All dies kann jetzt nur mit der Verwendung von CSS3 realisiert werden. Mit den Eigenschaften *transition-property, transition-duration, transition-timing-function* und *transition-delay* lässt sich der Zustand eines Elementes zu jedem Zeitpunkt anpassen. Auf der CD befinden sich dazu zwei Beispiele. Im ersten Transition-Beispiel werden die Effekte auf einen Link angewendet, wohingegen im zweiten Beispiel ein ganzes Menü mit den Effekten erstellt wurde. In den Beispielen zu den mobilen Anwendungen werden zudem Slide-Effekte bei dem Wechsel von einer zur anderen Seite angewendet.

Mit der Animations-Eigenschaft von CSS3 lassen sich HTML-Elemente ohne Flash oder JavaScript direkt im Browser animieren.

Abbildung 35: CSS3 Animation

Die Abbildung links ist ein Animationsbeispiel auf der CD. Bewegt man die Maus über eines der Vierecke, bewegt sich dieses nach rechts, um anschließend wieder auf die Ursprungsposition zurückzukehren.

Dazu werden die Eigenschaften *animation-duration, animation-name, animation-direction und animation-iteration-count* benötigt. Erstere definiert die Dauer der Animation, wohingegen letztere definiert, wie oft eine Animation ausgeführt werden soll. Mit *animation-direction* lässt sich die Art der Animation angeben. Im Beispiel handelt es sich um eine lineare Animation von A nach B. Eine weitere wichtige Eigenschaft ist *@keyframes*, welche über den Animationsnamen *animation-name* angesprochen wird und die Attribute *to* und *from* besitzt, welche den Anfangszustand und den Endzustand definieren [Gil10].

7 Mobile Webseite und Web Applikation

In Kapitel 6 wurde gezeigt, dass es webbasierte Techniken ermöglichen komplexe und dynamische Webseiten zu schreiben, die u.a. eine Datenbank verwenden und die Position eines Gerätes bestimmen können. Mit Hilfe von CSS3 kann man für diese Webseiten zudem eine anspruchsvolle Benutzeroberfläche gestalten, die dreidimensionale Körper, Transistions und Animationen enthalten kann. Weiterhin erlauben die mit HTML5 erstellten Webseiten das Abspielen von multimedialen Inhalten ohne jegliche Verwendung von Plug-Ins. Auch kann man die Webseite so gestalten, dass ein Anwender die Funktionen einer solchen offline nutzen kann. Doch wie kann man von all diesen webbasierten Techniken Gebrauch machen, um eine Anwendung zu schreiben, die auf mobilen Endgeräten läuft und dessen Erstellung als Alternative zur nativen Entwicklung gilt? Diese Frage soll in diesem Kapitel geklärt werden.

Zunächst muss auf die Überschrift des Kapitels eingegangen werden, denn die Definition einer *mobilen Webseite* und einer *Web Applikation (Web App)* ist nicht eindeutig. Um eine Webseite auf mobile Endgeräte komfortabel nutzen zu können, muss diese so angepasst werden, dass die Bedienung auf ein solches Gerät ausgelegt ist. Eine für mobile Geräte optimierte Webseite sollte beispielsweise den V*iewport Meta Tag* mitbringen, welcher die Darstellung einer Webseite passend zum Display eines Gerätes skaliert [W3C117]. Die bloße Anpassung der Bedienung und Skalierung greift aber nicht weit genug, denn ein Entwickler muss auch die besonderen Eigenschaften und Umstände eines mobilen Gerätes beachten, denn dieses wird im Gegensatz zu einem festen Desktop-Rechner auch außerhalb eines drahtlosen Netzwerks genutzt. Dementsprechend ist die Qualität der Konnektivität nicht immer für große Inhalte ausgelegt. Um die Antwortzeiten auch bei schlechtesten Bedingungen gering zu halten, muss der zu übertragende Inhalt demzufolge möglichst klein sein. Des Weiteren muss auch die Menüstruktur an die geringere Auflösung angepasst werden. Beachtet man diese Punkte, kann man im Grunde von einer *mobilen Webseite* sprechen. Vergleicht man beispielsweise die Desktop- und die mobile Version der *SPIEGEL ONLINE GmbH* [SPI11], stellt man fest, dass in der mobilen Version *http://m.spiegel.de* die Menüstruktur, die Skalierung, sowie der Inhalt der Seite angepasst wurden. Letzterer sieht auf dem ersten Blick zwar identisch aus, doch bei genauerem Vergleich sind die Bilder der mobilen Variante in einer geringeren Auflösung und stärker komprimiert vorhanden. Zudem wird auf bestimmte Inhalte gänzlich verzichtet, so dass die Darstellung der Seite kompakter ist und schneller übertragen werden kann.

In den bisherigen Überlegungen wurde jedoch das Look & Feel einer Anwendung außen vor gelassen. Das Beispiel der mobilen Version der *SPIEGEL ONLINE GmbH* ist nicht nur eine für mobile Bedürfnisse angepasste Webseite, sondern sie sieht auch so aus und fühlt sich auch so an. Mit *Fühlen* sind in dem Zusammenhang insbesondere die Benutzeroberfläche, sowie die Navigation innerhalb dieser gemeint. Wenn webbasierte Techniken als Alternative zur nativen Entwicklung in Betracht gezogen werden wollen, dann muss auch die Möglichkeit bestehen mit diesen ein Look & Feel zu erzeugen, welches einer nativen Anwendung in nichts nachsteht. Dies kann mit CSS3 und insbesondere mit den in den kommenden Unterkapiteln vorgestellten Frameworks erreicht werden.

Doch macht das Erzeugen eines nativen Look & Feels eine mobile Webseite zu einer Web App? Diese Frage kann nicht eindeutig beantwortet werden, denn der Übergang von einer mobilen Webseite zu einer Web App ist fließend, weshalb es auch keine eindeutige Definition der beiden Begriffe gibt. Apple spricht beispielsweise von einer Web App, wenn diese ähnliche Eigenschaften aufweist wie eine native Anwendung. Explizit spricht Apple dabei die Möglichkeit des Hinzufügens eines Icons auf den Homescreen und das Ausblenden des Menüs und der Statusleiste des Webbrowsers an [App111]. Google hingegen spricht in den *Best Practices for Web Apps* [Goo1111] schon von einer Web App, wenn der V*iewport Meta Tag* benutzt, der Inhalt wie im Beispiel von der *SPIEGEL ONLINE GmbH* für mobile Inhalte optimiert wird und man nur hoch und runter, statt auch

nach rechts und links, scrollen kann. Gleichzeitig beschreibt Google die Möglichkeit der Nutzung des SDKs und eines *WebViews*, in welchem man die auf webbasierte Techniken gefußte Anwendung aufrufen kann. Dadurch lässt sich auch unter Android ein Icon erstellen und zudem die Anwendung in den Android Market stellen.

In dieser Arbeit soll versucht werden, beide Begriffe besser voneinander abzugrenzen. Dazu wird zusätzlich zu den bisherigen Punkten die Offline-Funktionalität der Applikation betrachtet. Wenn eine Anwendung die *Data Storage* Funktion von HTML5 nutzt (siehe 6.3.2), dann liegen die Daten zwar bereits auf dem Client und bleiben auch erhalten, wenn dieser offline ist. Die Funktionalität der Anwendung selber, kann allerdings nur genutzt werden, wenn die ganze Anwendung mit Hilfe der *Offline Application Cache API* auf dem Client gespeichert wird (siehe 6.3.3).

Eine Web App soll in dieser Arbeit deswegen mit folgenden Eigenschaften definiert werden:

- Die Skalierung ist mit Hilfe des V*iewport Meta Tags* auf mobile Endgeräte ausgelegt.
- Das Look & Feel, insbesondere in Hinblick auf die Benutzeroberfläche und der Navigation, entspricht nahezu der einer nativen Applikation.
- Die Anwendung ist innerhalb einer APK-Datei gekapselt und kann daher installiert werden und besitzt ein eigenes Icon.
- Die Anwendung kann in den Android Market gestellt werden.
- Die Anwendung kann vollständig offline ausgeführt werden.

Eine mobile Webseite hingegen ist nicht innerhalb einer APK-Datei gekapselt, kann nicht installiert und aus diesem Grund auch nicht in den Android Market gestellt werden. Ob die Anwendung offline ausgeführt werden kann, soll bei einer mobilen Anwendung nur optional sein. Da in dieser Arbeit die Entwicklung webbasierter Anwendungen mit der Entwicklung nativer Anwendungen verglichen wird, soll sowohl bei einer Web App, als auch bei einer mobilen Webseite, das Look & Feel einer nativen Applikation entsprechen.

7.1 Voraussetzungen

Um eine mobile Webseite oder eine Web App zu erstellen, muss ein Programmierer HTML5, JavaScript und CSS3 beherrschen. Diese Grundlagen wurden im vorherigen Kapitel dieser Arbeit genauer beleuchtet. Das Kapseln einer Anwendung innerhalb einer APK-Datei erfordert zudem Grundkenntnisse von Android und dessen SDK, welche in den ersten zwei Kapiteln dieser Arbeit vermittelt wurden.

Webentwickler, die bisher keine Erfahrungen in der mobilen Entwicklung gesammelt haben, dürfte der Einstieg leicht fallen, da sie die webbasierten Techniken ohnehin beherrschen. Denn schaut man sich die mannigfaltigen Job-Angebote an, die für Webentwickler ausgeschrieben sind, sind HTML, JavaScript und CSS immer wiederkehrende Anforderungen [mak11]. Aufgrund dieser hohen Verbreitung gibt es auch reichlich Informationen und viele Anlaufstellen zu den webbasierten Techniken, so dass auch kompletten Neueinsteigern, die bisher nicht viel mit der Webentwicklung in Kontakt kamen, der Einstieg leicht fällt.

Es wird keine bestimmte Entwicklungsumgebung vorausgesetzt. Dem Entwickler steht es daher frei, ob er einen einfachen Texteditor oder eine integrierte Entwicklungsumgebung wie Eclipse nutzt. Es bietet sich allerdings an, einen Texteditor zu nutzen, der zu mindestens Code-Highlighting unterstützt, da dies beispielsweise die Erkennung von Fehlern im Code vereinfacht.

7.2 Möglichkeiten

Im Kapitel 6 dieser Arbeit wurde gezeigt, dass ein Webbrowser Zugriff auf die Geolocation API hat, wodurch auch die Verwendung eines ggf. verbauten GPS-Moduls innerhalb einer Applikation möglich ist. Des Weiteren

stehen Local Storage, Session Storage und eine Web SQL zur Verfügung. Auch die Möglichkeit eine Applikation offline zu verwenden besteht.

Zum jetzigen Zeitpunkt ist es in Android mit JavaScript möglich Single-Touch Events für den Webbrowser zu erstellen, allerdings keine Multi-Touch Events [Goo1112]. Ein Zugriff auf die Kamera via Webbrowser ist derzeit nicht möglich, jedoch arbeitet das W3C an der *HTML Media Capture* Spezifikation, mit der es in Zukunft funktionieren soll [W3C118]. Auch ist es nicht möglich, dass Androids Webbrowser Benachrichtigungen versendet. Für dieses Problem wurde ebenfalls eine W3C Arbeitsgruppe gegründet, die es in Zukunft ermöglichen soll, Benachrichtigungen über einen Webbrowser zu verschicken [W3C119].

Mehr Zugriff auf Hardwaresensoren und anderen Funktionen bekommt man, wenn man die Anwendung in einer APK-Datei kapselt. Mit *PhoneGap* (siehe 7.5) ist ein Zugriff auf den Beschleunigungssensor, die Kamera, den Kompass, die Kontakte, Dateien und Ortsangaben, das Netzwerk und Storage möglich. Die Kapselung ermöglicht zudem eine Installation der Anwendung mit eigenem Icon. Weiterhin lässt sie eine gekapselte Anwendung auch in den Android Market stellen, was sonst unmöglich wäre.

Wenn eine native Anwendung von einem Benutzer geschlossen wird, dann läuft diese in Android im Hintergrund weiter (siehe 4.2.1). Selbiges gilt für webbasierte Anwendungen. Dieses Verhalten lässt sich zum Beispiel mit folgendem Code beweisen:

```
<html>
<body>
<script type="text/javascript">
        setInterval(function() {
        var image = new Image();
        image.src = "images/" + new Date().getTime() + ".png";
        }, 1000);
</script>
</body>
</html>
```

Ein Aufruf der *setInterval()* Funktion in diesem Kontext bedeutet, dass eine Bilddatei sekündlich angefordert wird. Wird im Android Browser ein neues Tab geöffnet oder wird der Android Browser geschlossen, dann kann man auf dem Server weiterhin die Requests der Applikation sehen. Diese Art der Hintergrundprozesse ist beispielsweise auf iOS nicht möglich und eröffnet Entwickler auf Android größere Möglichkeiten, wenngleich bei einer solchen Applikation sichergestellt werden muss, dass nicht unnötig Daten im Hintergrund geladen werden oder zu viel Energie verbraucht wird.

Der größte Vorteil einer webbasierten Anwendung liegt in der Portierbarkeit. Das mobile Web konvergiert in Richtung WebKit und HTML5 (siehe 6.1). Eine webbasierte Anwendung ist in der Theorie daher leicht auf andere Betriebssysteme übertragbar.

Für die mobile Entwicklung stehen diverse Frameworks zur Verfügung. Im Wesentlichen kann man diese in *Markup-basierte* und *deklarative Frameworks* aufteilen. Während in einem Markup-basierten Framework die Benutzeroberfläche in purem HTML/CSS3 definiert wird, verfolgen deklarative Frameworks einen programmatischen Ansatz, d.h. die Elemente einer Benutzeroberfläche werden erst deklariert und anschließend programmatisch dieser hinzugefügt.

Wie mit *PhoneGap* schon angedeutet, gibt es auch *Bridge-Frameworks.* Diese Frameworks versuchen die Lücke zwischen webbasierten und nativen Anwendungen zu schließen, indem sie eine webbasierte Anwendung in ein Binärpaket, im Falle von Android eine APK-Datei, kapseln und Schnittstellen zum SDK und dessen API bereitstellen. Ein *Bridge-Framework* kann man speziell für Android mit einem *WebView* auch selber schreiben,

wobei die Schnittstellen zum SDK dann selbst implementiert werden müssen. Der Vorteil gegenüber *PhoneGap* besteht darin, dass die mit einem *WebView* erstellte Anwendung speichereffizienter ist (siehe 7.6).

Einen ganz anderen Ansatz verfolgt *Titanium Appcelerator Mobile,* mit welchem eine Anwendung ebenfalls mit Web-Technologien wie JavaScript erstellt und auch eine Binärdatei generiert wird. Allerdings wird aus dem JavaScript in der laufenden Anwendung nativer Code, so dass beispielsweise auch native UI-Elemente erzeugt werden. Damit die erstellten Applikationen auf den jeweiligen mobilen Betriebssystemen nativ aussehen und sich auch so anfühlen, wird im generierten Binärpaket ein JavaScript Interpreter mit ausgeliefert, welcher zur Laufzeit den eigenen Code in native Plattformaufrufe transformiert. *Titanium Appcelerator Mobile* ist aus diesem Grund nicht wirklich ein Web Framework, sondern viel mehr ein Kompatibilitätslayer bzw. ein Compiler. Dies bedeutet gleichzeitig, dass man abhängig von den Entwicklertools des Herstellers ist. Die Konsequenz daraus ist, dass beispielsweise kein richtiges Debuggen möglich ist, wie es für die anderen Frameworks in Kapitel 7.7 vorgestellt wird. *Titanium Appcelerator Mobile* ist kein echtes Web Framework und bildet extra eine Kategorie, in der auch Technologien wie *Adobe Air* zu finden sind. Aus diesem Grund wird auf ein extra Kapitel zu diesem Framework verzichtet.

Im Folgenden sollen einzelne Web-Frameworks und dessen Möglichkeiten detaillierter vorgestellt werden.

7.3 jQuery Mobile

jQuery Mobile ist ein kostenloses für Touch-Geräte optimiertes Markup-basiertes Framework, welches vom *jQuery Projekt Team* entwickelt wird und daher auch auf *jQuery* aufbaut. Es ist Verfügbar unter der *GPL2* und der *MIT* Lizenz. *jQuery* binhaltet die folgenden Funktionen [Wik119]:

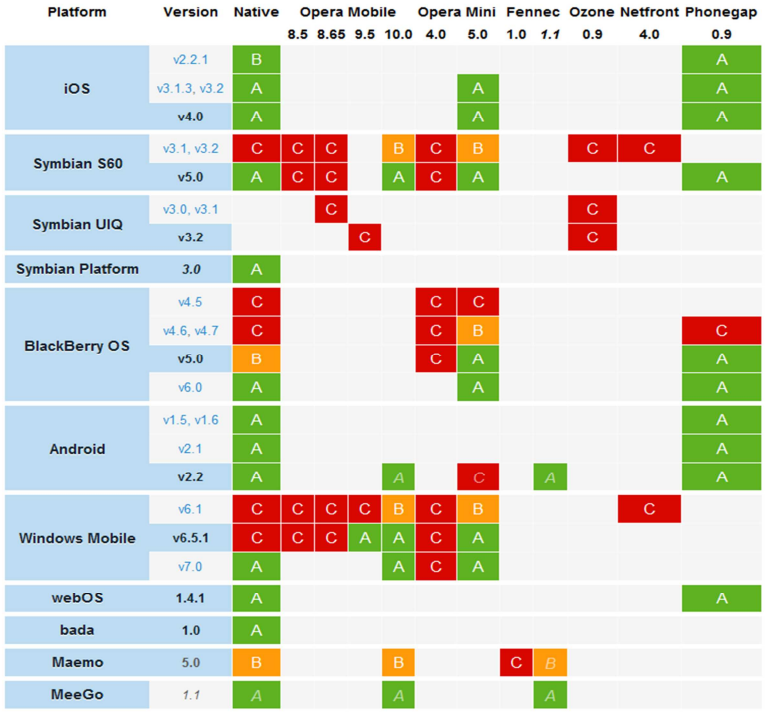
- Elementselektion im Document Object Model
- Document Object Model-Manipulation
- Erweitertes Event-System
- Hilfsfunktionen, wie zum Beispiel die each-Funktion
- Effekte und Animationen
- Ajax-Funktionalitäten
- Zahlreiche frei verfügbare Plug-Ins
- Erweiterbarkeit

jQuery ist die meistverwendete JavaScript-Bibliothek [QSu11], weshalb es vielen Entwicklern leicht fallen dürfte, mobile Anwendungen mit *jQuery Mobile* zu schreiben. Dieses ermöglicht ein einfaches und schnelles Erstellen einer Benutzeroberfläche inklusive Transition-Effekte. Mit *jQuery Mobile* werden mobile Webseiten erstellt, die aber auch mittels eines Wrappers zur Web Applikation gemacht werden können.

Es gibt noch keine stabile Version, allerdings schreitet die Entwicklung von *jQuery Mobile* rasch voran. Die erste Alpha-Version erschien am 26. Oktober 2010. Innerhalb der letzten fünf Monate erschienen drei Updates, wobei die aktuelle Version der Release Candidate 1 (v1.0 RC1) ist. Mit der ersten stabilen Version kann spätestens am Anfang des Jahres 2012 gerechnet werden.

Das *jQuery Projekt Team* stellt auf seiner Webseite eine Matrix bereit (vgl. Abb. 36), die aufzeigt, wie gut das Framework auf den unterschiedlichen mobilen Betriebssystemen und Browsern läuft. Die Klassifizierung findet mit den drei Buchstaben *A, B* und *C* statt, wobei A die beste und C die schlechteste Kompatibilität klassifiziert. Ein Entwickler kann daher schnell ablesen, ob und wie seine gewünschten Kombinationen unterstützt werden.

Android wird beispielsweise ab Version 1.5 mit einem *A* gekennzeichnet. Da *jQuery Mobile* vollständig kompatibel zu *PhoneGap* ist, findet man auch dieses in der Matrix wieder. Die Matrix ist für einen Entwickler als erste Anlaufstelle zu verstehen, denn geht man davon aus, dass das Ziel des Entwicklers eine schnelle und leichte Portierbarkeit bei gleichzeitig hoher Funktionalität ist, kann man hier alles auf einen Blick schon vor der Entwicklung ablesen, ohne im Nachhinein böse Überraschungen zu erleben.



Quelle: http://jquerymobile.com/gbs

Abbildung 36: jQuery Mobile Support Matrix

7.3.1 Support und Dokumentation

jQuery Mobile baut auf *jQuery* auf. *jQuery* ist die meistverwendete JavaScript-Bibliothek [QSu11] und hat daher eine große Community und eine sehr ausführliche Dokumentation. *jQuery Mobile* selbst besitzt eine noch nicht ganz so ausführliche Dokumentation und noch keine so große Community. Dies liegt allerdings daran, dass das Framework zum Zeitpunkt des Schreibens dieser Arbeit noch nicht mal ein Jahr alt ist. Aufgrund der großen Verbreitung von *jQuery* kann aber davon ausgegangen werden, dass das mobile Framework in Zukunft eine große Community haben wird. Ein weiterer Grund zu dieser Annahme ist, dass das Projekt Sponsoren wie *Adobe, Nokia, Mozilla* und *Palm* hat [The112] und viele Plattformen unterstützt. *Adobe* hat *jQuery Mobile* beispielsweise in Dreamweaver CS5.5 integriert und stellt eigene Tutorials zur Verfügung, was dem Framework weiteren Auftrieb verschaffen dürfte.

Die Dokumentation der unterstützten Plattformen ist wie im vorherigen Kapitel beschrieben hervorragend und ein großer Pluspunkt für das Framework. Für den Austausch der Entwickler untereinander stellt das *jQuery Projekt Team* ein Forum bereit, in dem Fragen in aller Regel zeitnah beantwortet werden.

Auf der Seite des Projektes ist eine Demo-Seite verfügbar, auf der man alle wichtigen Eigenschaften von *jQuery Mobile* live ansehen kann. Ein Kritikpunkt zu dieser Seite ist, dass nicht immer der dazugehörige Quellcode dabei steht, so dass man diesen dann unter Umständen in den Ressourcen mühselig selbst heraussuchen muss.

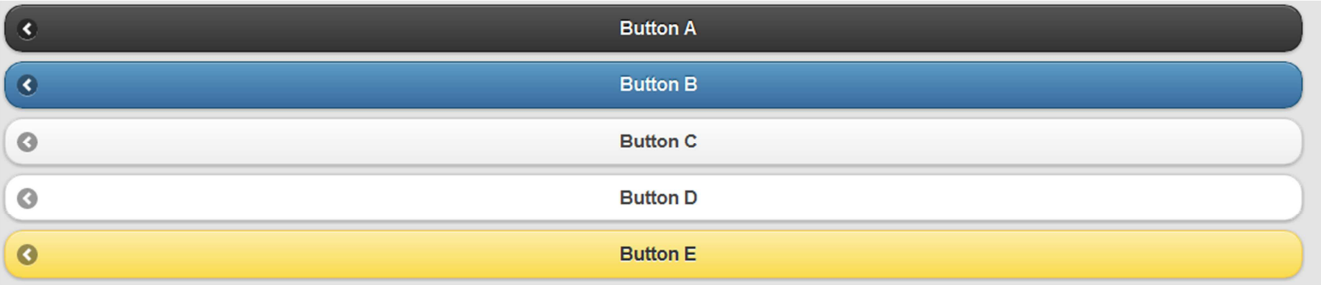
7.3.2 Framework-spezifische Möglichkeiten

jQuery Mobile ist, wie bereits erwähnt, ein Markup-basiertes Framework. Die Stärke des Frameworks ist daher die Tatsache, dass man für die Erstellung der Benutzeroberfläche keine Programmiersprache können muss. Nachdem das Framework heruntergeladen und integriert wurde, kann direkt mit der Erstellung der Oberfläche begonnen werden. Die vom *jQuery Projekt* bereitgestellte Demo-Seite, auf der man alle wichtigen UI-Elemente live ansehen kann, ist dabei eine große Hilfe.

Die Anatomie einer Seite in *jQuery Mobile* ist wie folgt:

```
<!DOCTYPE html>
<html>
	<head>
	<title>Titel</title>
	<meta name="viewport" content="width=device-width, initial-scale=1">
	<link rel="stylesheet" href=http://code.jquery.com/mobile/1.0rc1/jquery.mobile-1.0rc1.min.css />
	<script src="http://code.jquery.com/jquery-1.6.4.min.js"></script>
	<script src="http://code.jquery.com/mobile/1.0rc1/jquery.mobile-1.0rc1.min.js"></script>
</head>
<body>
	<div data-role="page">
		<div data-role="header">...</div>
		<div data-role="content">...</div>
		<div data-role="footer">...</div>
	</div>
</body>
</html>
```

Anhand des Attributs *data-role* wird die Rolle des jeweiligen HTML-Tags definiert. Im Codeausschnitt wurde innerhalb des body-Tags eine Seite definiert, welche aus einem Header, einem Inhalt und einem Footer besteht. Innerhalb dieser Tags können wiederum Rollen vergeben werden, die das Framework dann entsprechend interpretieren kann. Wenn die Applikation mehr als nur eine Seite enthalten soll, lassen sich in der gleichen Datei weitere Seiten hinzufügen, indem man einfach weitere div-Tags mit *data-role="page"* definiert. Des Weiteren stellt das Framework vordefinierte *Themes* zur Verfügung, die das Aussehen eines oder mehrerer Elemente beeinflussen. Möchte man beispielsweise ein *Theme* zu einem Button hinzufügen, findet man folgende Demo:



Quelle: http://jquerymobile.com/demos/1.0rc1/docs/api/themes.html
Abbildung 37: jQuery Mobile Button Themes

Gefällt einem die Ansicht des blauen Buttons, definiert man dieses mit dem *data-theme* Attribut:

```
<a href="#" data-role="button" data-theme="b">Button</a>
```

Ebenfalls durch ein Attribut lässt sich das im Screenshot zu sehende Icon des Buttons ändern.

```
<a href="#" data-role="button" data-theme="b" data-icon="search">Button</a>
```

Das Attribut *data-icon="search"* macht aus dem Pfeil eine Lupe.

Auf diese Art und Weise lassen sich alle UI-Elemente mit einfachen Attributen definieren. Selbst ein Transition-Effekt wird durch ein einfaches Attribut hinzugefügt. Durch die Demo-Seite lässt sich schnell das gewünschte Design finden und dieses in die eigene Anwendung übernehmen. *jQuery Mobile* erkennt dann während der Laufzeit alle Attribute und generiert daraus den gewünschten CSS3-Code.

Nachdem die Benutzeroberfläche einer Applikation fertiggestellt ist, muss man die Logik der Anwendung implementieren. Eine weitere große Stärke von *jQuery Mobile* liegt darin, dass man dafür nicht unbedingt JavaScript verwenden muss. Prinzipiell ist es vollkommen egal, welche Sprache dafür genutzt wird. Beispielsweise kann man die Logik auch in PHP implementieren. Der Nachteil wäre dann aber, dass man die Anwendung nur online nutzen kann. Die *Offline Application Cache API* (siehe 6.3.3) würde keinen großen Nutzen haben, da Android keinen PHP-Interpreter besitzt.

7.3.3 Beispiel: ToDo-Liste

In Kapitel 5.5.1 wurde eine ToDo-Liste nativ mit dem Android SDK entwickelt. Für einen Vergleich zwischen der nativen Entwicklung und den webbasierten Techniken soll mit *jQuery Mobile v1.0 RC1* ebenfalls eine ToDo-Liste entwickelt werden.

Die Applikation soll ToDo's bzw. Notizen in eine Liste persistent abspeichern können, so dass sie nach einem Neuaufruf der Anwendung immer noch vorhanden sind. Des Weiteren sollen die Notizen vom Benutzer auch bearbeitet und gelöscht werden können. Zusätzlich soll die Möglichkeit bestehen, ToDo's in die drei Kategorien *niedrig*, *normal* und *hoch* einzuordnen, womit eine Priorisierung der ToDo's erreicht wird.

In Kapitel 6.3.2.2 wurde die Web SQL anhand eines Beispiels, welches nicht für mobile Anwendungen optimiert ist, vorgestellt. Eine Stärke der webbasierten Techniken ist die Wiederverwendbarkeit des Quellcodes bzw. dessen Portierungsmöglichkeiten. Aus diesem Grund soll im Idealfall möglichst viel Code des Beispiels übernommen werden, um die Stärke auch nachzuweisen. Um die Daten persistent zu speichern, wird daher eine Web SQL verwendet.

Die *jQuery Mobile* basierte ToDo-Applikation wird aus drei Seiten bestehen. Die erste Seite ist die Übersicht aller abgespeicherten und priorisierten ToDo's, von denen die Titel zu sehen sind. Die Buttons für das Hinzufügen oder Löschen von Elementen sind ebenfalls auf der Seite enthalten. Ein Drücken auf den jeweiligen Titel wird den Inhalt des ToDo's anzeigen. Für das Hinzufügen, Anzeigen und Editieren von den Notizen wird eine extra Seite erstellt. In der nativen Anwendung wurde das Löschen so implementiert, dass durch ein langes Drücken auf den Titel ein Kontextmenü erscheint, in dem man die Löschoption auswählen kann. In *jQuery Mobile* wurde keine Möglichkeit gefunden, solche Kontextmenüs zu implementieren. Daher wird diese Funktion mittels eines Dialogs, welches fast ähnlich zu einem Kontextmenü ist, gelöst. Für diesen Dialog wird die dritte Seite innerhalb der Applikation benötigt.

Die Übersichtseite aller Notizen wird wie folgt in HTML definiert:

```
<div id="mainPage" data-role="page">
        <div data-role="header">
                <h1></h1>
                <a href="#addPage" id="reset" data-role="button" data-icon="plus">ToDo hinzufügen</a>
        </div>

        <div data-role="content">
                <ul id="todoList" data-role="listview" data-inset="true" data-filter="true" data-split-icon="delete">
                        <li id="hoch" data-role="list-divider">Hoch</li>
```

```
                    <li id="normal" data-role="list-divider">Normal</li>
                    <li id="niedrig" data-role="list-divider">Niedrig</li>
                </ul>
            </div>
</div>
```

Im Header der Seite wird ein Button mit Hilfe des *data-role* Attributs zum Hinzufügen neuer Elemente definiert. Mit dem Attribut *data-icon* stehen dem Entwickler die unterschiedlichsten Icons zur Auswahl:

Abbildung 38: jQuery Mobile Buttons

In diesem Fall passt ein *plus,* welches ein Button mit einem Pluszeichen generiert, zur Illustrierung des Hinzufügens sehr gut.

Alle Elemente werden im Content der Seite der gleichen Liste hinzugefügt, wobei es innerhalb dieser so genannte *list-divider* gibt, womit die Priorisierung der Elemente vorgenommen wird. Jeder Eintrag der Liste wird am rechten Rand einen Löschbutton besitzen, womit später der Dialog zum Löschen aufgerufen wird. Das Attribut *data-filter="true"* erzeugt ein Textfeld, mit welchem man die Einträge filtern kann. Diese Funktion steht zwar nicht in den oben genannten Anforderungen, doch zeigt sie an dieser Stelle die einfache und schnelle Verwendung bzw. Implementierung von sinnvollen Features, ohne dass sich ein Entwickler über die eigentliche Implementierung eines Filters Gedanken machen muss.

Das Definieren des Layouts und des Designs der beiden weiteren Seiten funktioniert nach dem gleichen Prinzip. Es müssen lediglich HTML-Elemente mit den entsprechenden Attributen hinzugefügt werden, woraus das Framework das Aussehen der Seite generiert. Das Erstellen einer Seite ist sehr intuitiv, wodurch sich in kürzester Zeit schöne Benutzeroberflächen ergeben.

Jedoch stößt man dabei hier und da an seltsame Eigenheiten von *jQuery Mobile*. Beispielsweise befindet sich im Header des oberen Codeausschnitts noch ein leerer *h1-Tag*. Auch wenn dem Header keine spezielle Überschrift gegeben wird, ist der leere Tag wichtig. Denn wird er entfernt, entfällt die Generierung des kompletten Headers. Das bedeutet, dass auch der Button zum Hinzufügen der Elemente nicht generiert wird, obwohl er in HTML definiert wurde. Eine weitere Eigenheit ist auch beim Definieren des Dialoges zum Löschen von Elementen aufgetreten. Das Definieren dieses Dialoges sieht folgendermaßen aus:

```
<div id="deleteDialog" data-role="page">
        <div data-role="header">
            <h1>ToDo-Liste</h1>
        </div>
        <div data-role="content" data-theme="b">
            <div align="center"><h4>Möchten Sie den Eintrag wirklich löschen?</h4></div>
            <a href="#mainPage" id="deleteTodo" data-role="button" data-icon="delete">Löschen</a>
            <a href="#mainPage" data-role="button" data-icon="back">Zurück</a>
        </div>
</div>
```

Der Dialog ist wie eine normale Seite definiert. Nur der aufrufende Verweis macht mit Hilfe des *data-rel="dialog"* Attributs aus einer Seite einen Dialog:

```
<a href="#deleteDialog" delId="'+row.id+'" data-rel="dialog" data-transition="pop"></a>
```

Dass ein Link eine *page* zu einem *dialog* macht, ist in der Dokumentation auch so beschrieben. Die Eigenheit hierbei ist jedoch, dass die beiden Sachen zwar identisch definiert werden, aber ein unterschiedliches *Theme*

als Standard besitzen, weshalb das Aussehen des Dialoges überhaupt nicht mehr zum Rest der Seite passt und wie ein Fremdkörper wirkt. Deswegen wurde, wie im oberen Codeausschnitt zu sehen, *data-theme="b"* zum *content* der Seite hinzugefügt. Dieses Theme entspricht dem Standardtheme, welches im Beispiel genutzt wird, so dass die Seiten jetzt wieder harmonisch zueinander wirken. Dieses Verhalten wirkt aus Entwicklersicht etwas inkonsistent. Zudem kostet es auch unnötig Zeit das passende Theme aus der Dokumentation herauszusuchen.

Dennoch konnte in relativ kurzer Zeit eine Benutzeroberfläche entworfen werden, die der Oberfläche der nativen Anwendung in puncto Aussehen überlegen ist.

Nachdem die Benutzeroberfläche erstellt wurde, erfolgt die Implementierung der eigentlichen Funktionalität. Dafür soll, wie bereits erwähnt, das Web SQL Beispiel aus Kapitel 6.3.2.2 als Vorbild dienen.

Die Erstellung und Verbindung zur Datenbank kann eins zu eins kopiert werden, wenngleich natürlich die Tabelle für die neuen Bedürfnisse angepasst werden muss:

```
var db = prepareDatabase();
var createSQL = 'CREATE TABLE todo_data (id INTEGER PRIMARY KEY,' +
                'ueberschrift TEXT NOT NULL,' +
                'beschreibung TEXT NOT NULL,' +
                'dringlichkeit INTEGER NOT NULL' +
            ')';

// überprüfen, ob der Browser SQL unterstützt
function getOpenDatabase() {
        try {
                if( !! window.openDatabase ) return window.openDatabase;
                else return undefined;
        } catch(e) {
                return undefined;
        }
}

// öffne die WebSQL Datenbank
function prepareDatabase() {
        var odb = getOpenDatabase();
        if(!odb) {
                return undefined;
        } else {
                var db = odb( 'ToDo_DB', '1.0', 'Datenbank für ToDo-Liste', 2 * 1024 * 1024 );
                db.transaction(function (t) {
                        t.executeSql( createSQL, [], function(t, r) {}, function(t, e) { });
                });
                return db;
        }
}
```

Für das Anzeigen aller ToDo-Elemente in der Übersicht wurde die *dispResults()* Funktion angepasst. Die eigentliche Funktionalität zum Auslesen der Elemente aus der Datenbank ist identisch, allerdings muss auf das neue Layout der Seite eingegangen werden (Auszug):

```
// zeigt die ToDo's in der Übersicht an
function dispResults() {
[...]
db.readTransaction(function(t) { // readTransaction setzt die Datenbank auf read-only
        t.executeSql('SELECT * FROM todo_data ORDER BY ueberschrift DESC)', [], function(t, r) {
                for( var i = 0; i < r.rows.length; i++ ) {
```

```
                        var row = r.rows.item(i);
                        var data = '<a href="#addPage"viewId="'+row.id+'" data-transition="slide">'+ row.ueberschrift
                                +'</a><a href="#deleteDialog" delId="'+row.id+'" data-rel="dialog"
                                data-transition="pop"></a>';
                if (row.dringlichkeit == 0) {
                        $("<li></li>").html(data).insertAfter($("#niedrig"));
                } else if (row.dringlichkeit == 1) {
                        $("<li></li>").html(data).insertAfter($("#normal"));
                } else {
                        $("<li></li>").html(data).insertAfter($("#hoch"));
                }
        }
        $("#todoList").listview("refresh");
});
[...]
```

In *jQuery Mobile* wurde kein Weg gefunden ein ähnliches Kontextmenü wie im nativen Beispiel zu implementieren. Aus diesem Grund wird zum Löschen eines Eintrages ein Löschbutton am rechten Ende eines Listeneintrages erzeugt. In *jQuery Mobile* lässt sich dies realisieren, indem man schlicht zwei Links nebeneinander definiert, wie in der Variable *data* zu sehen ist. Das Framework erkennt die zwei nebeneinander stehenden Links innerhalb eines Listeneintrages und generiert daraus dann eine so genannte *split button list*. Der erste Link ist dabei der eigentliche Inhalt des Listeneintrages, der Titel eines ToDo's. Der zweite Link definiert den Löschbutton des Elements. Nachdem die ToDo's in die jeweiligen Kategorien einsortiert wurden, wird die Liste mit *refresh* aktualisiert.

Das größte Problem während der Implementierung bestand darin, dass man in *jQuery Mobile* nicht ohne weiteres Parameter von einer zur anderen Seite übergeben kann. Um Einträge aktualisieren und löschen zu können, ist aber genau dies von Nöten. Gelöst wurde das Problem, indem für die Liste ein Handler implementiert wurde. Im letzten Codeausschnitt wurden die Links innerhalb eines Listeneintrages mit den Attributen *viewId* und *delId* versehen. Der Handler prüft, welches der beiden Attribute im ausgeführten Link vorhanden ist und steuert dann die weiteren Aktionen. Ist es *viewId*, soll die im Tag definierte ID des Eintrages angesehen und editiert werden können, wohingegen bei *delId* der Listeneintrag gelöscht werden soll. Zusätzlich wurden zwei Hilfsfelder definiert, indem die jeweilige Aktion und die zu bearbeitende bzw. zu löschende ID gespeichert werden:

```
<input id="inputAction" type="hidden" name="action" value="update" />
<input id="inputId" type="hidden" name="rowId" value="0" />
```

Je nach Aktion verändert sich *inputAction* auf *update, delete* oder *add*. Für die ersten beiden Operationen wird außerdem die ID des Eintrages in der Datenbank benötigt, weshalb diese im Feld *inputId* gespeichert wird. Die Zuweisungen steuert der Handler:

```
// steuert das Verhalten eines Klicks auf einen Eintrag in der Übersicht
$('#todoList').delegate('a', 'click', function() {
        var viewId = $(this).get(0).getAttribute('viewId');
        var delId = $(this).get(0).getAttribute('delId');
        if (viewId) {
                if(db) {
                        $("#inputAction").val("update");
                        $("#inputId").val(viewId);
                        db.readTransaction(function(t) { // readTransaction setzt die Datenbank auf read-only
                                t.executeSql('SELECT * FROM todo_data WHERE id = ?', [viewId], function(t, r) {
                                        for( var i = 0; i < r.rows.length; i++ ) {
                                                var row = r.rows.item(i);
```

```
                                        $("#todoUeberschrift").val(row.ueberschrift);
                                        $("#todoBeschreibung").val(row.beschreibung);
                                        $("#todoDringlichkeit").val(row.dringlichkeit).trigger("change");
                                    }
                                });
                            });
                    }
            } else if (delId) {
                    $("#inputAction").val("delete");
                    $("#inputId").val(delId);
            }
});
```

Mit Hilfe dieses Handlers ist es nun relativ einfach die Parameter einer anderen Seite zu übergeben. Denn diese müssen anschließend lediglich aus den Hilfsfeldern ausgelesen werden:

```
// klick auf den Lösch-Button
$("#deleteTodo").click(function() {
        var rowId = $("#inputId").val();
        dbGo(rowId);
});
```

Die Fallunterscheidung, ob etwas hinzugefügt, editiert oder gelöscht wird, geschieht in der Funktion *dbGo(id).* Diese ähnelt erneut stark der gleichnamigen Funktion im Web SQL Beispiel (Auszug):

```
// zuständig für das Hinzufügen, Updaten und Löschen eines Eintrages
function dbGo(id) {
        var action = $("#inputAction").val();
        [...]
switch(action) {
case 'add':
        [...]
case 'update':
        [...]
case 'delete':
        db.transaction(function(t) {
                t.executeSql('DELETE FROM todo_data WHERE id = ?', [id]);
        });
        break;
}
[...]
dispResults();
```

Bei einem Aufruf der ToDo-Liste ist im Android Browser noch dessen Adresszeile zu sehen. Um diese auszublenden, kann der Viewport wie folgt angepasst werden:

```
var nPageH = $(document).height();
var nViewH = window.outerHeight;
if (nViewH > nPageH) {
        nViewH = nViewH / window.devicePixelRatio;
        $('BODY').css('height',nViewH + 'px');
}
window.scrollTo(0,1);
```

Die folgenden Abbildungen zeigen die mit *jQuery Mobile* entwickelte ToDo-App als mobile Webseite.

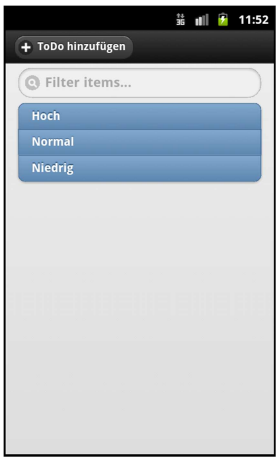


Abbildung 39: ToDo-List Start

Abbildung 40: Add/Edit ToDo

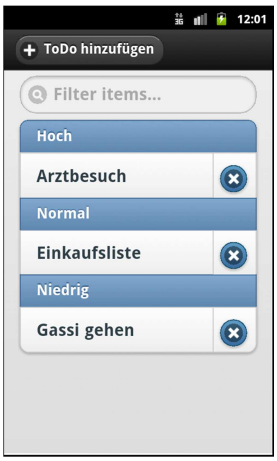


Abbildung 41: ToDo Übersicht

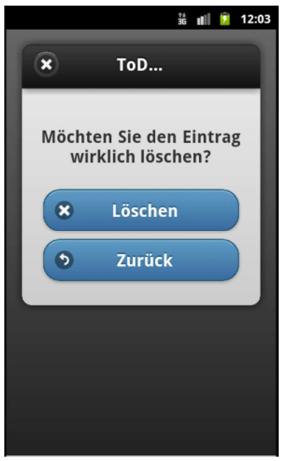


Abbildung 42: ToDo löschen

Das Beispiel befindet sich auf der CD im Ordner *webbasiert\ToDo-Liste*.

7.3.3.1 Offline Verwendung

Die ToDo-Liste soll vollständig offline verfügbar sein, weshalb die *Offline Application Cache API* (siehe 6.3.3) zum Einsatz kommt. Die Manifest-Datei *todo_liste.appcache*, die im HTML-Tag referenziert ist, wird folgendermaßen definiert:

```
CACHE MANIFEST
index.html
scripts/jquery-1.6.4.min.js
scripts/jquery.mobile-1.0rc1.min.js
css/jquery.mobile-1.0rc1.min.css
css/images/ajax-loader.png
css/images/icons-18-white.png
css/images/icons-36-white.png
```

Damit der MIME-Type des Manifests vom *Apache Web Server* korrekt erkannt wird, wird die Datei *.htaccess* mit folgendem Inhalt beigelegt:

```
AddType text/cache-manifest .appcache
```

Ob das Caching erfolgreich verläuft, kann in der Konsole der *Chrome Developer Tools* (siehe 7.6) überprüft werden:

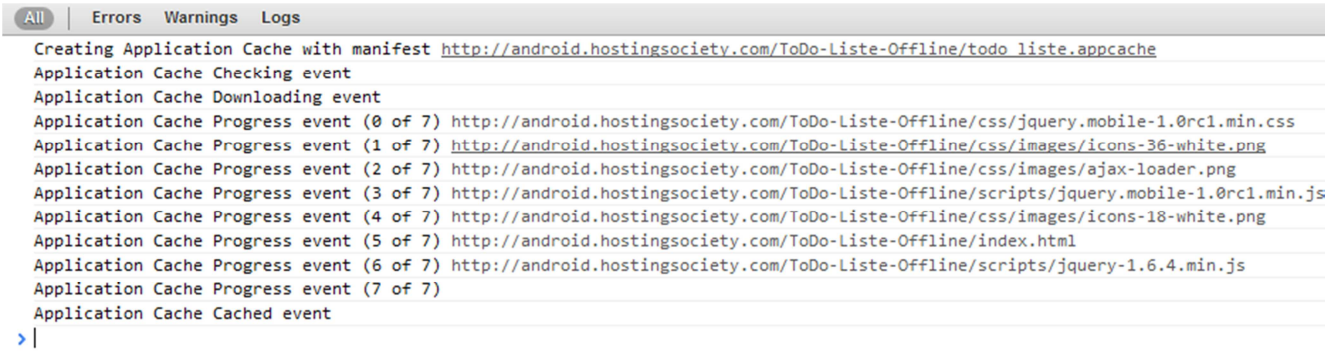


Abbildung 43: ToDo-List - Offline Application Cache API

In diesem Fall verlief das Caching erfolgreich, so dass die Anwendung nun vollständig offline verfügbar ist. Um die im Beispiel verwendete Web SQL muss man sich keine Gedanken machen, da diese ohnehin lokal auf dem Gerät gespeichert wird.

Die ToDo-List, die die *Offline Application Cache API* verwendet, befindet sich auf der CD im Ordner *webbasiert\ToDo-Liste-Offline*.

7.3.4 Fazit

Die Erstellung von Applikationen mit *jQuery Mobile* erweist sich als relativ einfach. Die Gestaltung der Benutzeroberfläche ist sehr intuitiv, was zusätzlich durch die Demo-Seite vereinfacht wird. Anfängern dürfte der Einstieg leicht fallen, da die Lernkurve extrem hoch ist. Erleichternd kommt hinzu, dass man bei der Auswahl der Logik auf keine bestimmte Programmiersprache festgelegt ist, wenngleich JavaScript die meisten Möglichkeiten bietet. Wäre die Logik der ToDo-Liste beispielsweise in PHP statt in JavaScript geschrieben worden, hätte man die Applikation nicht offline nutzen können.

Bei der Entwicklung stößt man hier und da an Eigenheiten des Frameworks. Auch Fehlen teilweise Funktionen wie die Übergabe eines Parameters von einer zur anderen Seite. Diese Eigenheiten bzw. das Nichtvorhandensein von Funktionen lassen sich jedoch durch Workarounds beseitigen, wenngleich es wünschenswert wäre, wenn diese Probleme gar nicht erst auftauchen würden. Dies liegt aber möglicherweise daran, dass *jQuery Mobile* bisher in keiner stabilen bzw. finalen Version vorliegt. Die Entwicklung von *jQuery Mobile* geschieht relativ zügig, so dass man damit rechnen kann, dass Schritt für Schritt Probleme entfernt und neue Features hinzugefügt werden.

Für das Testen der Anwendungen stehen nur ein HTC Desire und ein Nexus One zur Verfügung. Diese Geräte sind etwa 1,5-2 Jahre alt. Zum Zeitpunkt des Erscheinens der Geräte gab es das Framework noch nicht einmal. Dennoch ist die Performanz der ToDo-Liste als befriedigend zu bezeichnen, wenngleich die Anwendung nicht ganz so schnell und flüssig wie die native Version der App läuft. Insbesondere die CSS3-Effekte wie Transitions laufen nicht ganz ruckelfrei. Dennoch ist in gleicher Zeit eine wesentlich schönere Benutzeroberfläche im Vergleich zur nativen Version erstellt worden, weshalb kleine Ruckler zu verkraften sind.

7.4 Sencha Touch

Sencha Touch ist ein von der Firma *Sencha* entwickeltes deklaratives Framework, welches auf *Ext JS*, *Raphael* und *jQTouch* aufbaut und womit sich mobile Webseiten für touch-fähige Geräte entwickeln lassen. Mittels eines Wrappers lassen sich diese mobilen Webseiten auch zu einer Web App transformieren. Beispielsweise ist *Sencha Touch* vollständig kompatibel zu *PhoneGap*. Das Framework ist kostenfrei und steht unter der GPLv3 Lizenz. Sencha Touch ist kompatibel zu Android 2.1+, iOS 3+ und BackBerry 6+.

Für die Verwendung des Frameworks muss man zwingend JavaScript beherrschen. Die grafischen Elemente von *Sencha Touch* ähneln denen von iOS, wobei das Aussehen über *Themes* gesteuert werden kann. Das Framework wartet mit einer umfassenden Bibliothek an UI-Widgets, einer Verwaltung für Touch-Ereignisse samt Transitions und einem umfangreichen Datenpaket auf.

Das Framework liegt momentan in der Version 1.1 vor. Die Weiterentwicklung geschieht zügig, so dass vor kurzem die Version 2 angekündigt wurde [Sen11]. Ab Version 2 soll ein Wrappen der Anwendung für iOS und Android in *Sencha Touch* integriert sein, was *PhoneGap* de facto überflüssig macht. Außerdem verspricht *Sencha* einen enormen Performance-Schub. Das Layout-System wurde überarbeitet, so dass das Rendern wesentlich schneller von statten geht und Anwendungen schneller starten können. Auch Touch-Events sollen schnel-

ler erkannt werden, wodurch man beispielsweise ruckelfreier durch Listen scrollen kann. Zum Zeitpunkt des Verfassens dieser Arbeit liegt die Version 2 jedoch noch nicht vor, weshalb diese Ankündigungen nicht überprüft werden konnten.

7.4.1 Support und Dokumentation

Sencha Touch besitzt eine umfassende Dokumentation, wenngleich diese auch Löcher aufweist. Oft fehlt zum Beispiel die Angabe der Werte, die ein Attribut annehmen kann oder was eine Änderung dessen bewirkt. Stößt ein Entwickler dann auf Bugs, die nicht dokumentiert sind, kann man lange Zeit mit dem Debuggen und Tracen (siehe 7.7) von Fehlern verbringen.

Eine ausführliche Kompatibilitätsmatrix, wie sie für *jQuery Mobile* zu finden ist, sucht man für *Sencha Touch* vergebens. Allerdings gibt es dafür, ebenfalls wie bei *jQuery Mobile*, eine Demo-Seite, die *Sencha Kitchen-Sink* nennt. In dieser *Kitchen-Sink* kann man viele Funktionen und ganze Beispiele live betrachten, wobei der Quellcode bei Bedarf auch angesehen werden kann.

Für den Austausch der Entwickler untereinander stellt *Sencha* ein Forum zur Verfügung, bei dem über 280 000 Mitglieder registriert sind. Das Forum ist daher eine gute Anlaufstelle für Probleme, die man selber nicht lösen kann. Sind die Antworten nicht ausreichend befriedigend, gibt es für das Framework zahlreiche Tutorials und Videos, die gerade für Einsteiger eine enorme Hilfe darstellen.

Wenn man auf einen professionelleren und schnelleren Support angewiesen ist, kann man bei *Sencha* einen Support für 299$/Jahr kaufen. Bei einem Kauf dieser Option verspricht die Firma schnellere Antwortzeiten, Telefonsupport, Remote Unterstützung, sowie die Möglichkeit, dass Experten den Code prüfen und ggf. „tunen“.

7.4.2 Framework-spezifische Möglichkeiten

Da *Sencha Touch* ein deklaratives Framework ist, unterscheidet es sich signifikant vom Markup-basierten *jQuery Mobile*. Bei diesem wurde die Benutzeroberfläche erstellt, indem HTML geschrieben und den Tags Attribute gegeben wurden, aus denen *jQuery Mobile* den Rest generiert hat. *Sencha Touch* generiert, basierend auf JavaScript Objekte, sein eigenes Document Object Model (DOM). Daher werden grafische Oberflächen nicht durch HTML-Code vorgegeben. Alle Elemente werden durch JavaScript Objekte beschrieben, woraus *Sencha Touch* zur Laufzeit den entsprechenden HTML-Code generiert.

Sencha Touch bietet einige Funktionen, die es in *jQuery Mobile* so nicht gibt. Beispielsweise gibt es so genannte *Data Stores*, in denen Daten persistent gespeichert werden. Bei Verwendung dieser Data Stores muss man keine Kenntnisse von Web SQL oder Local Storage besitzen. Stattdessen könnte man folgendes abstrakte Data Model für die ToDo-App definieren:

```
Ext.regModel('ToDo', {
        idProperty: 'id',
        fields: [
                { name: 'id', type: 'int' },
                { name: 'ueberschrift', type: 'string' },
                { name: 'inhalt', type: 'string' }
         ],
        validations: [
                { type: 'presence', field: 'id' },
                { type: 'presence', field: 'ueberschrift', message: 'Bitte geben sie eine Überschrift ein!' }
        ]
});
```

Gleichzeitig kann man bestimmte Spalten auf deren Inhalte über die Funktion *validate()* überprüfen und bei einer leeren Überschrift eine Warnung ausgeben. Mit *Ext.Msg.alert(..)* ließe sich diese Warnung als Popup anzeigen – ein Feature, das man bei *jQuery Mobile* vergebens sucht, weshalb für die Löschbestätigung dort ein Dialog herhalten musste, der sich zwar ähnlich wie ein Popup verhält, aber im Gegensatz zu einem Popup die komplette Seite verdeckt.

Ein weiteres erwähnenswertes und exklusives Feature sind die Toolbars bzw. deren Verhalten in *Sencha Touch*. *Sencha Touch* ist das einzige Framework, bei dem es bisher möglich ist, Elemente wie eine Toolbar oder einen Footer so zu definieren, dass sie fest an ihrem Platz bleiben. In anderen Frameworks, wie zum Beispiel *jQuery Mobile,* scrollen die Elemente mit oder werden während des Scrollens ausgeblendet und erst wieder angezeigt, wenn das Scrollen beendet wurde.

7.4.3 Fazit

Sencha Touch bietet eine umfangreichere Funktionalität als seine Konkurrenten. Dennoch wurde für die webbasierte ToDo-Liste *jQuery Mobile* ausgewählt. Der Grund für diese Entscheidung war hauptsächlich die Anforderung, dass möglichst große Teile von einem in Kapitel 6 vorgestelltem Beispiel portiert werden sollen. In *Sencha Touch* ist ein eigenes Data Model integriert, welches einfach zu nutzen, aber für eine eins zu eins Portierung ungeeignet ist. Sicherlich hätte man auch das Data Model ignorieren und eine eigene Web SQL implementieren können, doch ist es in der Regel kontraproduktiv, wenn man Dinge anders macht, als wie es im Framework vorgesehen ist. Beispielsweise lassen sich in *Sencha Touch* die Datenelemente direkt als Liste ausgeben, so dass die Implementierung einer eigenen Web SQL ineffizienter wäre.

Wenn man mit JavaScript vertraut ist und der Begriff *objektorientiert* kein Fremdwort ist, fällt der Einstieg in *Sencha Touch* sehr leicht. Wie bei *jQuery Mobile* standen zum Testen von Anwendungen nur ein HTC Desire und ein Nexus One zu Verfügung. Die beiden Geräte kämpfen bei Applikationen, die mit *Sencha Touch* erstellt wurden, mit ähnlichen Problemen wie in *jQuery Mobile*. Insbesondere die CSS3-Effekte wie Transitions laufen nicht ganz ruckelfrei, wenngleich *Sencha Touch* einen minimal flüssigeren Eindruck macht. Dafür dauert das Starten von *Sencha Touch* Anwendungen aufreizend lange, während man beim Start von *jQuery Mobile* Anwendungen nahezu gar nicht warten muss.

Die Entwicklung mit *Sencha Touch* „fühlt“ sich anders an als mit *jQuery Mobile*. Während die Entwicklung mit Letzterem sehr an die Erstellung von Webseiten erinnert, fühlt sich der Umgang mit *Sencha Touch* eher wie das Programmieren mit Technologien wie *Java* oder *Adobe Flex* an. Aus diesem Grund entscheiden nicht nur die Features und die Anforderungen die Wahl des Frameworks, sondern auch die persönlichen Präferenzen eines Entwicklers. Wenn man vor der Wahl eines Frameworks steht, macht es daher Sinn zumindest kleine *Hello World* Anwendungen in allen Frameworks zu schreiben, um ein Gefühl für diese zu bekommen, damit man sich anschließend in dem gewählten Framework wohl fühlt.

7.5 PhoneGap

PhoneGap ist ein Open-Source *Bridge-Framework* von der Firma *Nitobi*. *PhoneGap* unterstützt derzeit iOS, Android, webOS, Symbian und Bada. *Bridge-Frameworks* wie *PhoneGap* versuchen die Lücke zwischen webbasierten und nativen Anwendungen zu schließen, indem sie eine webbasierte Anwendung in ein Binärpaket, im Falle von Android eine APK-Datei, kapseln und Schnittstellen zum SDK und dessen API bereitstellen. Dadurch wird es möglich Web Applikationen zu entwickeln, die man in den Android Market stellen kann, die ein eigenes Icon besitzen und stets offline Verfügbar sind.

Das Framework lässt sich mit *jQuery Mobile* und *Sencha Touch* kombinieren, indem die Logik in den beiden genannten Frameworks geschrieben wird, und Features, die nur ein *Bridge-Framework* bereitstellen kann, mit *PhoneGap* hinzugefügt werden. So ist mit *PhoneGap* unter Android ein Zugriff auf den Beschleunigungssensor, die Kamera, den Kompass, die Kontakte, Dateien und Ortsangaben, das Netzwerk und Storage möglich.

Das Framework befindet sich momentan in der Version 1.1 und wird stetig weiter entwickelt. Es besteht eine Roadmap, die bis in den Sommer 2012 reicht. Bis dahin soll *PhoneGap* beispielsweise Zugriff auf die Kalender API, die Messaging API und Bluetooth haben [Nit11].

7.5.1 Support und Dokumentation

Für *PhoneGap* gibt es einen Start Guide, der insbesondere für Anfänger sehr hilfreich ist. Neben einer API Dokumentation gibt es auch ein Wiki, in welchem hauptsächlich Tutorials zu finden sind. *PhoneGap* ermöglicht das Einbinden von Plug-Ins, deren Entwicklung ebenfalls im Wiki angesprochen wird. Die API Dokumentation ist recht ausführlich, wobei positiv anzumerken ist, dass zu vielen APIs ein Beispiel angegeben ist. Kritisch anzumerken ist, dass *PhoneGap* in den Tutorials keine bestimmten Berechtigungen erwähnt, die eine Funktion benötigt. Stattdessen sind die Tutorials so ausgelegt, dass stets sämtliche Berechtigungen vergeben werden.

Des Weiteren kann man kostenlos an einer Community teilnehmen, in dem man sich an einer Google Group beteiligt oder einem IRC Channel beitritt.

Weiteren Support bietet *Nitobi* gegen Aufpreis. Der kostenpflichtige Support ist in fünf Kategorien eingeteilt und kostet je nach Leistung 24,95$ bis 1995$ pro Monat. Im günstigsten Support-Paket erhält der Entwickler Zugriff auf Webinare und ein privates Forum. Ist man bereit für den Support mehr Geld auszugeben, erhält man dementsprechend auch mehr Leistung. So erhält man beispielsweise im teuersten Paket zusätzlich Telefonsupport, Remote Debugging, Plug-In Support, Email-Support, eine garantierte Erreichbarkeit, eine garantierte Antwortzeit und einige Dinge mehr.

7.5.2 Beispiel: ToDo-Liste

Die mit *jQuery Mobile* erstellte ToDo-Liste soll mittels *PhoneGap* in einer APK-Datei gewrapped werden, so dass aus der mobilen Webseite ein Web App wird, welche installiert werden kann und ein eigenes Icon besitzt.

Dafür muss zunächst das Android SDK installiert werden. Mit der Entwicklungsumgebung Eclipse (siehe 5.4.1) wird anschließend ein neues Projekt inklusive einer Activity namens *App* erstellt. Im root-Verzeichnis des Projektes werden die Ordner */libs* und */assets/www* angelegt. Im ersteren wird die Datei *phonegap.jar* herein kopiert, im letzteren die Datei *phonegap.js*, sowie das gesamte *jQuery Mobile* Projekt. Die *phonegap.js* muss in der *index.html* der ToDo-Liste importiert werden. Des Weiteren muss der xml-Ordner von *PhoneGap* in den */res* Ordner kopiert werden.

Nachdem diese Vorbereitungen getroffen wurden, muss der Code der Activity angepasst werden. Diese soll keine Activity-Klasse mehr erweitern, sondern eine DroidGap-Klasse, wofür *com.phonegap.** importiert werden muss. Ggf. muss die importierte *phonegap.jar* explizit dem Build-Path hinzugefügt werden, wenn das Paket von Eclipse unerkannt bleibt. Die Activity-Klasse wird nicht mehr benötigt, so dass deren Import gelöscht werden kann. Die Zeile *setContentView()* wird mit folgender ersetzt:

```
super.loadUrl("file:///android_asset/www/index.html");
```

Anschließend muss nur noch die *AndroidManifest.xml* editiert werden. In den *PhoneGap* Tutorials werden an dieser Stelle sämtliche Berechtigungen, die das Framework nutzen kann, freigegeben. Da das ToDo-App aber keine speziellen Berechtigungen benötigt, wird dies an dieser Stelle nicht getan. Nachteilig wäre diese Vorge-

hensweise, wenn man später spezielle *PhoneGap*-APIs nutzen möchte, da in der API-Referenz nicht beschrieben wird, welche Berechtigung die jeweilige Funktion nutzt.

Abschließend wird nur noch das Icon der Applikation angepasst. Der Code der ToDo-Liste kann auch noch etwas „getuned“ werden. Beispielsweise kann der Code zum Ausblenden der Adresszeile gelöscht werden, da dies von *PhoneGap* automatisch vorgenommen wird.

Die gewrappte Anwendung befindet sich auf der CD als Quellcode und als signierte APK-Datei im Order *webbasiert\ToDo-Liste-PhoneGap*.

7.5.3 Fazit

Die Nutzung von *PhoneGap* erweist sich als recht einfach. Das Bridge-Framework kommt erst so richtig zur Geltung, wenn es mit anderen Frameworks wie *jQuery Mobile* oder *Sencha Touch* kombiniert wird, da in diesen die Benutzeroberfläche erstellt wird. Mit *PhoneGap* kann man dann die schon fertigen Anwendungen in ein natives Binärpaket wrappen, so dass die Applikation vom Anwender beispielsweise in den Android Market gestellt und installiert werden kann. Die *PhoneGap*-APIs, die eine Schnittstelle zum Android SDK darstellen, sind ausreichend gut dokumentiert, wenngleich es wünschenswert wäre, wenn die jeweilig benötigten Rechte einer Funktion genannt werden würden. Zu beachten ist auch, dass das Debuggen der *PhoneGap*-APIs nur relativ schwer möglich ist, da dies weder mit der *Android Debug Bridge* (siehe 5.6), noch mit den *Chrome Developer Tools* (siehe 7.7) möglich ist. Aus diesem Grund ist es eher abzuraten, komplexe Anwendungen, die exzessiv von den APIs Gebrauch machen, in *PhoneGap* zu entwickeln.

Für einfachere Applikationen wie die ToDo-Liste stellt *PhoneGap* aber eine gute, einfache und schnelle Möglichkeit dar, die Apps dahingehend zu verändern, damit sie anschließend über den Android Market installiert werden können und in der Liste der installierten Apps gelistet werden.

Bei der gewrappten ToDo-Anwendung ist kein Nachteil zur ungekapselten Version erkennbar. Die Funktionalitäten, sowie Effekte wie Transitions, sind identisch.

7.6 WebView

Mit *PhoneGap* wurde eine Brücke zwischen den mobilen Webseiten und dem Android SDK geschlagen, so dass man die mobilen Webseiten zu Web Apps transformieren kann. Diese Funktionalität kann man in Android mit Hilfe eines *WebViews* auch selber programmieren.

Zunächst wird dafür ein neues Projekt erstellt. Anschließend muss dem Layout ein WebView hinzugefügt werden. Die *layout_height* und *layout_width* Eigenschaften des WebViews bekommen dabei den Wert *fill_parent*, damit sich das WebView über den gesamten Bildschirm erstreckt. Das komplette *jQuery Mobile* Projekt wird in den Ordner *assets/www* kopiert.

In einem WebView sind zunächst alle Optionen deaktiviert. Für die ToDo-Liste muss daher JavaScipt und die Datenbankfunktionalität aktiviert werden. Für die Datenbank muss der Pfad zum Speicherort definiert werden. Im Falle des Beispiels ist es */data/data/de.mahlert.todo_liste_webview/datenbank*. Des Weiteren muss die Größe der Datenbank definiert werden, da sie sonst nur 0 KB beträgt:

```
   private class EinWebChromeClient extends WebChromeClient {
        @Override
        public void onExceededDatabaseQuota (String url, String databaseIdentifier, long currentQuota, long esti-
matedSize, long totalUsedQuota, WebStorage.QuotaUpdater quotaUpdater) {
                quotaUpdater.updateQuota(2 * 1024 * 1024);
        }
```

```
}
```

Nach der ersten Ausführung der App stellt man fest, dass das WebView *trotz parent_fill* nicht den kompletten Bildschirm bedeckt. Es ist ein kleiner weißer Rand zu sehen, der für eventuelle Scrollbars freigehalten wird. Aus diesem Grund muss auch das Design der Scrollbars verändern werden. Die Veränderungen des WebView im Überblick:

```
WebView webview = (WebView) findViewById(R.id.webview);
webview.setWebViewClient(new EinWebViewClient());
webview.setWebChromeClient(new EinWebChromeClient());
webview.getSettings().setJavaScriptEnabled(true);
webview.getSettings().setDatabaseEnabled(true);
webview.getSettings().setDatabasePath("/data/data/de.mahlert.todo_liste_webview/datenbank");
webview.setScrollBarStyle(WebView.SCROLLBARS_OUTSIDE_OVERLAY);
webview.setScrollbarFadingEnabled(false);
webview.loadUrl("file:///android_asset/www/index.html");
```

Die mit einem WebView gewrappte Anwendung verbraucht 183 KB auf dem Endgerät, während die mit *PhoneGap* erstellte Anwendung ganze 471 KB verbraucht. Im laufenden Zustand sind allerdings keine Unterschiede zu erkennen. Das mit WebView gekapselte App „fühlt“ sich genauso an wie das mit *PhoneGap* erstellte App.

Die Erstellung mittels *PhoneGap* erweist sich als wesentlich einfacher. Bei einem WebView müssen viele Veränderungen vorgenommen werden, die man erst einmal verstehen muss. Wenn man seine Applikation für unterschiedliche Betriebssysteme wrappen will, müssen diese Änderungen für jedes System individuell verstanden und implementiert werden. Bei der Verwendung von *PhoneGap* sind diese Veränderungen nicht nötig, wodurch das Wrappen viel einfacher, schneller und intuitiver funktioniert. Allerdings ist die Implementierung eines WebViews ein einmaliger Aufwand, der sich auf Grund der besseren Speichereffizienz lohnen kann.

7.7 Debugging

Das Debugging bezieht sich auf mobile Anwendungen, die nicht innerhalb einer APK-Datei gekapselt sind. Wenn ein Entwickler beispielsweise *PhoneGap* verwendet und durch dessen Schnittstellen auf die Android APIs zugreift, ist diese Art des Debuggens nicht mehr möglich. Das Debuggen von Anwendungen, die mit einem *Bridge-Framework* erstellt wurden, ist daher eine ungleich komplexere Angelegenheit und sollte bei der Wahl der Art der Entwicklung berücksichtigt werden.

Chrome basiert wie der Android Browser auch auf WebKit und der JavaScript-Code wird auf die gleiche Weise interpretiert. Deswegen bietet es sich an die *Chrome Developer Tools* zu nutzen [The111], um webbasierte Anwendungen zu debuggen.

Die in den Chrome Developer Tools zur Verfügung gestellte Konsole ist oftmals schon ausreichend, um Probleme innerhalb einer Anwendung zu erkennen. In der Konsole kann man Reporting-Nachrichten ausgeben lassen, sowie manuell Code ausführen. Das folgende Beispiel soll die Reporting-Möglichkeiten verdeutlichen:

```
<script type="text/javascript">
        // gibt Nachrichten in der Konsole aus
        console.debug("debug message");
        console.info("info message");
        console.warn("warning message");
        console.error("error message");
        // wirft eine Exception
        throw new Error("Das ist eine Testexception");
</script>
```

Nach der Ausführung des Codes zeigt die Konsole alle Nachrichten, Fehler und Exceptions inklusive der Zeile, in der die Meldungen und Probleme in der Datei auftauchen, auf (vgl. Abb. 44). Möchte man einen Fehler im Kontext des Quellcodes sehen, landet man bei einem Klick auf die rechts zu sehende unterstrichende Datei in den Reiter *Resources* und in der Zeile des Codes (vgl. Abb. 45).

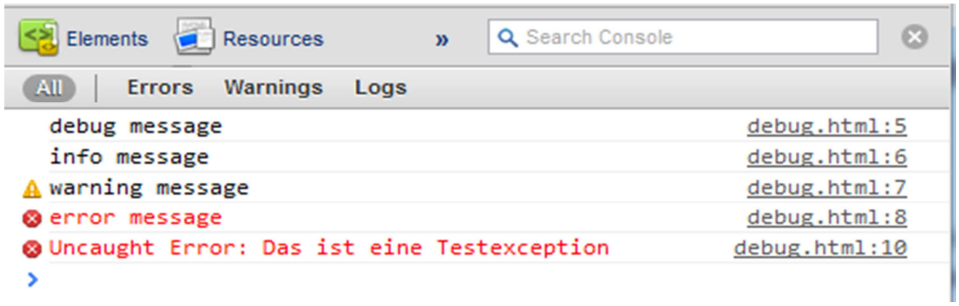


Abbildung 44: Chrome Dev Tools Konsole

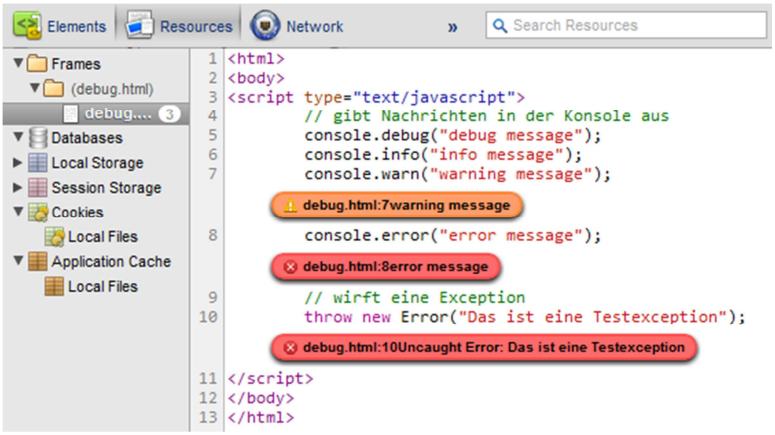


Abbildung 45: Chrome Dev Tools Resources

In dieser Ansicht werden Debug- und Infomeldungen nicht ausgegeben, so dass man Warnungen, Fehler und Exceptions auf einen Blick erkennen kann. Durch die farbliche Hervorhebung und den Kontext lassen sich Fehler schnell erkennen. Gute Logausgaben sind bei der Entwicklung komplexerer Anwendungen essentiell, da man sonst mühselig die Fehler im Code per Try-and-Error-Verfahren suchen müsste, sofern man sie nicht auf den ersten Blick erkennt. Des Weiteren bietet das linke Menü Möglichkeiten der Ansicht eventueller Datenbankeinträge, sofern die Anwendung Data Storage verwendet (siehe 6.3.2). Auch kann man beobachten, ob die *Application Cache API* (siehe 6.3.3 und 7.3.3.1), falls verwendet, wie gewünscht funktioniert.

Diese Art des Debuggens kann in vielen Fällen hilfreich sein und ist mit Logcat im Android SDK zu vergleichen, doch greift sie noch nicht weit genug. Deswegen kann man in den Chrome Developer Tools unter dem Reiter *Scripts* auch mit Hilfe von Breakpoints debuggen. Als Beispiel soll folgendes Script untersucht werden, welches eine Zufallszahl erzeugt:

```
<html>
<body>
<script type="text/javascript">
// erzeugt Zufallszahl
var x = Math.round(Math.random() * 100);
</script>
</body>
</html>
```

Die Steuerung des Debuggers funktioniert wie beim Debuggen in Eclipse mit drei Buttons:

Abbildung 46: Chrome Dev Tools Debug Steuerung

Der erste Pfeil bedeutet *Step over*. Ist der Breakpoint auf einen Funktionsaufruf gesetzt, dann lässt sich mit F10 die Funktion überspringen, während man mit F11, *Step Into*, die Funktion aufrufen und Zeile für Zeile durchgehen könnte. Mit Shift und F7, *Step Out*, kann zu den vorherigen Zuständen zurückgekehrt werden.

Im Beispiel wurde der Breakpoint auf die Wertzuweisung der Zufallszahl *x* gesetzt. Um den Wert der Zufallszahl sehen zu können, muss zunächst die Zeile ausgeführt werden. Wie in der nachfolgenden Abbildung zu sehen, kann man nach dem Benutzen von F10 den Wert der Zufallszahl ablesen.

Abbildung 47: Chrome Dev Tools Script Debugger

Auf diese Art und Weise lassen sich selbst komplexeste Anwendungen debuggen, da man mit Hilfe von Breakpoints den Zustand der Applikation zu einem beliebigen Zeitpunkt genau analysieren kann.

7.8 Reverse Engineering

Auch das Reverse Engineering bezieht sich auf Anwendungen, die nicht in einer APK-Datei gekapselt wurden. Das Reverse Engineering gekapselter Applikationen ist aber dennoch möglich und im Prinzip ein Mix aus Kapitel 5.7 und diesem hier.

HTML-Dokumente inklusive JavaScript lassen sich in Klartext anzeigen. Reverse Engineering ist in dem Fall gar nicht mehr nötig, da man die Daten einfach eins zu eins kopieren und übernehmen kann. Dieser Umstand ist für Entwickler oder Unternehmen, die Ihren Code schützen wollen, inakzeptabel. Es ist daher zu empfehlen einen so genannten *Obfuscator* zu nutzen, welcher den Quelltext in eine für den Menschen schwerer lesbare Form generiert.

Im Internet findet man viele verschiedene Obfuscator. Stellvertretend soll als Beispiel der Obfuscator von *Voormedia* [Voo11] genutzt werden. Dieser soll auf die folgende Zeile angewendet werden:

```
var x = Math.round(Math.random() * 100);
```

Das Beispiel generiert eine Zufallszahl und ist trivial, verdeutlicht aber die Funktionalität eines Obfuscators, denn der Output ist für den Menschen unlesbar:

```
<script language="JavaScript" type="text/javascript">
var
i,y,x="3c68746d6c3e0d0a3c626f64793e0d0a3c736372697074207479706 53d22746578742f6a6176617363726
97074223e0d0a09092f2f2065727a65756774205a7566616c6c737a61686c0d0a0909766172207820 3d204d617
4682e726f756e64284d6174682e72616e646f6d2829202a20313030293b0d0a3c2f7363726970743e0d0a3c2f6
26f64793e0d0a3c2f68746d6c3e";y='';for(i=0;i<x.length;i+=2){y+=unescape('%'+x.substr(i,2));}document.write(
y);
</script>
```

8 Native Entwicklung vs. webbasierte Technologien

In den bisherigen Kapiteln wurde auf die native und auf die webbasierte Entwicklung eingegangen. In diesem Kapitel sollen die gewonnenen Erkenntnisse zusammengefasst und analysiert werden, so dass Anwendungsfälle konstruiert werden können, aus denen hervor geht, wann welche Art der Umsetzung wann zu bevorzugen ist.

8.1 Vergleich der Voraussetzungen

Eine native Entwicklung geschieht aufgrund der Android Architektur in aller Regel mit dem Android SDK, weshalb ein Entwickler Java können muss. Die eigentliche native Programmiersprache wäre C / C++, welche maschinennahe und in der Regel sehr effizient ist [Wik115]. Die Entwicklung mit Hilfe des NDKs erhöht allerdings die Komplexität und die Fehleranfälligkeit der Anwendung. Aus diesem Grund empfiehlt Google selbst die Programmierung in Java und nicht in C / C++ [Goo112]. Dennoch können C / C++ Kenntnisse eines Entwicklers gefragt sein, da Teile der Anwendung, insbesondere leistungskritische Teile, in dieser Sprache implementiert werden können.

Für die Entwicklung mit webbasierten Techniken sind Kenntnisse in HTML5, CSS3 und JavaScript von Nöten. Es stehen mittlerweile einige Web-Frameworks für mobile Webseiten zur Verfügung, die sich in die Kategorien *Markup-basierte* und *deklarative Frameworks* einordnen lassen. *jQuery Mobile* ist ein Markup-basiertes Framework, in welchem die Benutzeroberfläche in purem HTML definiert wird. Dieser HTML-Code wird zur Laufzeit vom Framework ausgewertet, woraus die komplette Benutzeroberfläche inklusive CSS3-Effekte wie Transitions generiert wird. *Sencha Touch* ist ein deklaratives Framework, weshalb es sich signifikant von diesem Ansatz unterscheidet. Es generiert, basierend auf JavaScript Objekte, sein eigenes Document Object Model (DOM). Daher werden grafische Oberflächen nicht durch HTML-Code vorgegeben, sondern durch JavaScript Objekte beschrieben, woraus *Sencha Touch* zur Laufzeit den entsprechenden HTML-Code generiert. Durch diese Unterschiede innerhalb der Web-Frameworks entstehen unterschiedliche Voraussetzungen für einen Entwickler. Für die Entwicklung mit *Sencha Touch* muss ein Entwickler zwingend JavaScript beherrschen und objektorientiert programmieren können, während man bei *jQuery Mobile* hauptsächlich HTML verstehen muss. Des Weiteren muss die Logik in *jQuery Mobile* nicht objektorientiert sein und nicht unbedingt mit Hilfe von JavaScript implementiert werden. Stattdessen kann sich der Entwickler die Sprache nahezu beliebig aussuchen, denn denkbar sind sämtliche Web-Technologien wie PHP, Perl, Python usw.

Als eine dritte Art von Web-Frameworks gibt es so genannte Bridge-Frameworks wie z.B. *PhoneGap*, mit denen man Web Applikationen erstellen kann. Diese Frameworks versuchen die Lücke zwischen den Web-Frameworks und der nativen Entwicklung zu schließen, weshalb alle bisher genannten Voraussetzungen auf *PhoneGap* zu treffen. Zu diesen zählen daher HTML5, CSS3, JavaScript und Kenntnisse des Android SDKs bzw. Java.

Bevor die Wahl auf die native oder die webbasierte Entwicklung fällt, sollte ein Entwickler für sich erörtern, welche der genannten Vorrausetzungen und Präferenzen er mitbringt. Zu beachten ist dabei, dass die Vorrausetzungen für die webbasierte Entwicklung, wie erwähnt, je nach Wahl des Frameworks unterschiedlich sein können.

8.2 Vergleich der Möglichkeiten

Das Android SDK erlaubt den Zugriff auf die Hardware eines Gerätes. Bei leistungskritischen Anwendungen, etwa einem Navigationssystem, können Teile der Anwendung für eine bessere Performance auch das NDK nutzen. Der Kreativität eines Entwicklers sind bei einer nativen Entwicklung daher keine Grenzen gesetzt. Ein Beispiel, welches einen Teil dieses Potentials ausschöpft, ist die *IP Cam* (siehe 5.5.2), welche einen Socket öffnet und einen Stream der Kamera bereitstellt.

Entscheidet man sich für eine mobile Webseite, stößt man im Falle der *IP Cam* an die Grenzen des Machbaren. Ein Zugriff auf die Kamera via Webbrowser ist derzeit nicht möglich, jedoch arbeitet das W3C an der *HTML Media Capture* Spezifikation, mit der es in Zukunft funktionieren soll [W3C118]. Auch ist es nicht möglich, dass Androids Webbrowser Benachrichtigungen versendet. Für dieses Problem wurde ebenfalls eine W3C Arbeitsgruppe gegründet, die es in Zukunft ermöglichen soll, Benachrichtigungen über einen Webbrowser zu verschicken [W3C119]. Generell kann man sagen, dass alles das möglich ist, worauf ein Webbrowser Zugriff hat. Zu nennen sind dabei insbesondere Local Storage, Session Storage, Web SQL und die Geolocation API. Mit Hilfe von CSS3 lassen sich zudem schöne Benutzeroberflächen schnell erstellen, die der nativen Oberfläche, die nur mühsam zu erstellen ist, ebenbürtig ist.

Mehr Zugriff auf Hardwaresensoren und anderen Funktionen bekommt man, wenn man die Anwendung in einer APK-Datei kapselt. Dieser Vorgang transformiert eine mobile Webseite zu einer Web Applikation. Das Wrappen kann entweder mit Hilfe eines WebViews im Android SDK selbst programmiert werden oder es wird ein Bridge-Framework wie *PhoneGap* genutzt. Mit *PhoneGap* ist ein Zugriff auf den Beschleunigungssensor, den Kompass, die Kontakte, Dateien und Ortsangaben, das Netzwerk und Storage möglich.

Die *ToDo-Liste* wurde einmal nativ und einmal als mobile Webseite entwickelt, wobei in beiden Fällen darauf geachtet wurde, in etwa gleich viel Zeit für die Entwicklung zu benötigen. Zusätzlich wurde die mobile Webseite jeweils mittels WebView und *PhoneGap* zur Web Applikation transformiert.

Die native ToDo-Liste erfüllt die Anforderungen, allerdings besitzt sie keine schöne Benutzeroberfläche. Die mit *jQuery Mobile* erstellte ToDo-Liste hingegen hat eine schöne Benutzeroberfläche, die der nativen überlegen ist. Aktuelle Geräte standen zum Testen der Anwendung nicht zur Verfügung, weshalb dieses auf 1,5-2 Jahre alten Geräten stattfand. Die native Anwendung ist sehr flüssig und extrem schnell. Die Flüssigkeit und die Schnelligkeit der mobilen Webseite sind nur als befriedigend zu bezeichnen. Insbesondere CSS3-Effekte wie Transitions laufen nicht ganz ruckelfrei, dessen Ursache in den alten Geräten begründet ist. Dennoch kann man auf Grund der viel schöneren Oberfläche kleine Ruckler verzeihen, so dass die mobile Webseite der nativen Applikation überlegen ist. Die Entwicklung der mobilen Webseite war im Vergleich zur nativen Variante sogar so viel schneller fertig gestellt, dass weitere Features hinzugefügt werden konnten, weshalb sich in dieser die ToDo-Punkte in der Liste priorisieren und filtern lassen.

Die mobile Webseite hat allerdings den Nachteil, dass man sie nicht in den Android Market stellen kann, denn man kann diese nicht installieren, weshalb sie auch kein eigenes Icon besitzt und nicht unter den Applikationen gelistet wird. Deswegen wurde die mobile Webseite jeweils mittels WebView und *PhoneGap* zur Web Applikation transformiert. Beide Ansätze kapseln die Anwendung in einer APK-Datei, wodurch man diese anschließend installieren kann. Dadurch bekommt die Anwendung ein eigenes Icon und kann auch in den Android Market gestellt werden. Die mit einem WebView gewrappte Anwendung verbraucht 183 KB auf dem Endgerät, während die mit *PhoneGap* erstellte Anwendung ganze 471 KB verbraucht. Im laufenden Zustand sind allerdings keine Unterschiede zu erkennen. Auch läuft die Anwendung genauso gut wie im ungewrappten Zustand. Die Erstellung mittels *PhoneGap* erwies sich als wesentlich einfacher. Bei einem WebView müssen viele

Veränderungen vorgenommen werden, die man erst einmal verstehen muss. Wenn man seine Applikation für unterschiedliche Betriebssysteme wrappen will, müssen diese Änderungen für jedes System individuell verstanden und implementiert werden, wenngleich dies einen einmaligen Aufwand darstellt. Möchte man diesen Aufwand allerdings nicht betreiben und geht man davon aus, dass man die Anwendung auf möglichst viele Betriebssysteme portieren will, ist *PhoneGap* dem WebView trotz der schlechteren Speichereffizienz überlegen.

Steht man vor der Wahl, ob die Entwicklung einer Applikation nativ oder webbasiert stattfinden soll, spielt das Requirements Engineering eine enorm wichtige Rolle. Erst wenn man die Anforderungen im Detail kennt, kann man die Möglichkeiten der jeweiligen Frameworks ausloten. Auch sollte man sich einen genauen Zeitrahmen setzen, denn unterschiedliche Anforderungen lassen sich je nach Ansatz, wie im Beispiel der ToDo-Liste zu erkennen, unterschiedlich schnell implementieren.

Ebenfalls sollte man sich im Klaren sein, auf welchen Betriebssystemen die Applikation lauffähig sein soll. Die mit dem Android SDK erstellten Programme sind nur auf Android lauffähig. Ein Vorteil des NDKs ist, dass es die Möglichkeit eröffnet, C++ Bibliotheken einzubinden. Dies macht eine Portierung von und zu anderen Systemen leichter, da nur Teile der Anwendung neu geschrieben werden müssen. Eine leichte und schnelle Portierbarkeit bieten webbasierte Technologien. Dies wurde bewiesen, indem große Teile der ToDo-App Logik dem Web SQL Beispiel aus dem Kapitel 6.3.2.2 entnommen wurden.

8.3 Vergleich des Supports

Unter den getesteten Frameworks sind *Sencha Touch* und *PhoneGap* die Einzigen, bei denen ein kostenpflichtiger Support angeboten wird. Bei einem Kauf dieser Option verspricht die Firma *Sencha* schnellere Antwortzeiten, Telefonsupport, Remote Unterstützung, sowie die Möglichkeit, dass Experten den Code prüfen und ggf. „tunen“. *Nitobi* bietet für *PhoneGap* den kostenpflichtigen Support in fünf Stufen an, der je nach Stufe 24,95$ bis 1995$ pro Monat kostet. Im günstigsten Support-Paket erhält der Entwickler Zugriff auf Webinare und ein privates Forum, während im teuersten Paket zusätzlich Telefonsupport, Remote Debugging, Plug-In Support, Email-Support, eine garantierte Erreichbarkeit, eine garantierte Antwortzeit und einige Dinge mehr angeboten werden.

Für das Android SDK/NDK kann man zwar keinen kostenpflichtigen Support hinzukaufen, allerdings verfügt Android über eine sehr große Community. Genannt werden soll hier stellvertretend die Community *xda-developers* [xda11] mit über vier Millionen registrierten Mitgliedern, bei der neben Android auch Windows Phone und WebOS Entwickler unter den Mitgliedern vertreten sind. Diese sind zum Teil technisch hoch versiert, so dass dort unter anderem alternative Android-Images für diverse Mobiltelefone entwickelt werden.

Für die Frameworks gibt es auch Communities, allerdings sind diese längst nicht so groß wie die von Android. Gleichwohl muss erwähnt werden, dass es trotz dieser großen und teils technisch hoch versierten Android Communities keine garantierten Antworten gibt. Dementsprechend sind die kostenpflichtigen Angebote als Vorteil anzusehen.

8.4 Vergleich des Debuggings

Google stellt mit der *Android Debug Bridge* Möglichkeiten bereit, um native Anwendungen sowohl im Android Emulator als auch auf einem Endgerät debuggen zu können. Diese Art des Debuggens ist sehr mächtig und für komplexe Projekte hervorragend geeignet.

Webbasierte Anwendungen lassen sich mittels der *Chrome Developer Tools* debuggen [The111], da der Desktop-Browser Chrome wie der Android Browser auf WebKit basiert und der JavaScript-Code auf die gleiche Wei-

se interpretiert wird. Diese Art des Debuggens ist ebenfalls sehr mächtig. Allerdings stellte sich während des Testens heraus, dass Chrome sich auch einmal anders als der Android Browser verhalten kann. Wenn dieser Fall eintritt, steht der Entwickler vor einem Problem, da mobile Webseiten sich nicht ohne Weiteres auf einem Smartphone debuggen lassen.

Wenn ein Entwickler eine mobile Anwendung zu einer Web App transformiert und dabei weitere *PhoneGap* APIs verwendet, die auf die Android APIs zugreifen, ist kein Debuggen mehr möglich. Das Debuggen von Anwendungen, die mit einem *Bridge-Framework* erstellt wurden, ist daher eine ungleich komplexere Angelegenheit und sollte bei der Wahl der Art der Entwicklung berücksichtigt werden.

Zusammenfassend kann man sagen, dass sich native Anwendungen am besten Debuggen lassen. Bei komplexen Anwendungen, in denen schnell Bugs auftreten, könnte dies ein Argument für die native Entwicklung sein.

8.5 Vergleich des Reverse Engineerings

Ein Reverse Engineering ist sowohl bei nativen als auch bei webbasierten Anwendungen möglich, wenngleich der Aufwand des Reverse Engineerings bei nativen Applikationen wesentlich höher ist.

Gegen Reverse Engineering kann man sich in beiden Entwicklungsarten schützen, indem man Obfuscator einsetzt. Dieser Punkt spielt bei der Wahl der Entwicklung daher keine Rolle.

8.6 Entscheidungsfaktoren

Im Folgenden sollen sämtliche Faktoren, die die Wahl der Umsetzung beeinflussen, aufgezählt und beschrieben werden.

Die **Wünsche der Anwender** und die **Möglichkeiten der unterschiedlichen Methoden** grenzen die Wahl der Umsetzung ein. Aus diesem Grund spielt das Requirements Engineering eine enorm wichtige Rolle. Erst wenn man die Anforderungen im Detail kennt, kann man die Möglichkeiten der jeweiligen Frameworks ausloten. Dabei ist der Unterschied zwischen mobiler Webseite und Web Applikation zu beachten und der Umstand, dass sich mobile Webseiten schnell und einfach zu Web Applikationen transformieren lassen.

Ebenfalls sollte man sich im Klaren sein, über welchen **Zeitraum** sich ein Projekt erstreckt und die gefundenen Anforderungen nach **Qualität** priorisieren. In der Regel stehen Softwareentwicklungsprojekte unter einem gewissen Zeitdruck, weshalb nicht alle Anforderungen qualitativ gleichwertig implementiert werden können. Eine Erkenntnis dieser Arbeit ist, dass sich unterschiedliche Anforderungen in gleicher Zeit in jedem Framework unterschiedlich gut implementieren lassen. Weiß ein Entwickler welche Anforderungen besonders wichtig sind und kennt er die Stärken und Schwächen aller Frameworks, fällt es ihm leicht eine Wahl zu treffen.

Auch ein **Budget** ist in der Regel begrenzt, weshalb ein Entwickler beispielsweise im Vorhinein wissen muss, auf welchen Plattformen seine Applikation lauffähig sein soll bzw. ob und von welchem Betriebssystem er diese portieren will. Die **Portierung** ist maßgeblich für die Wahl der Umsetzung. In dieser Arbeit wurde gezeigt, dass es möglich ist, aus einer HTML5-Anwendung, die für Desktop-Browser geschrieben war, große Teile des Codes wiederzuverwenden, so dass daraus schnell eine mobile Webseite erstellt werden konnte, die auf den unterschiedlichsten mobilen Betriebssystemen lauffähig ist. Eine native Entwicklung für unterschiedliche Betriebssysteme hätte in diesem Fall mehr Zeit und ein größeres Budget verschlungen.

Es ist wichtig, dass ein Entwickler die **Endgeräte der Anwender** kennt, da die **Performanz** jedes Frameworks unterschiedlich ist. Eine nativ entwickelte Anwendung läuft in der Regel immer schneller als eine webbasierte Anwendung. In den Web-Frameworks gibt es insbesondere auf Geräten mit älterer Hardware Mängel bzgl. der

Performanz grafischer Effekte, welche neuere Geräte flüssiger darstellen können. Doch gibt es auch innerhalb der Web-Frameworks Unterschiede. Zum Beispiel haben Transition-Effekte in *Sencha Touch* einen flüssigeren Eindruck gemacht als in *jQuery Mobile*. Dafür dauert in *Sencha Touch* das Starten einer Anwendung aufreizend lange, welches in *jQuery Mobile* richtig schnell von statten geht. An dieser Stelle ist jedoch zu beachten, dass die Web-Frameworks relativ neu sind. Beispielsweise ist *jQuery Mobile* zum Zeitpunkt des Verfassens dieser Arbeit noch nicht einmal ein Jahr alt und existiert bisher erst als Release Candidate 1. Allerdings geht die Entwicklung der Web-Frameworks zügig voran. Zudem entwickelt sich die im Endgerät verbaute Hardware auch immer weiter, so dass mittlerweile schon Dual-Core Prozessoren in Smartphones verbaut werden. Daher ist davon auszugehen, dass der Faktor Performanz in Zukunft eine immer geringere Rolle spielen wird.

Gleichwohl beschränken derzeit noch **leistungskritische Anwendungen** die Wahl auf eine native Entwicklung mit dem Android SDK oder gar dem NDK. Die im NDK verwendete Sprache C++ ist maschinennahe und in der Regel sehr effizient [Wik115].

Ein wichtiger Punkt für die Wahl der Umsetzung kann auch das **Debuggen** sein. Das Debuggen von nativen Anwendungen ist dem von webbasierten Anwendungen überlegen. Es ist daher zu raten, große und komplexe Applikationen, die besonders anfällig für Bugs sind, nativ zu entwickeln.

Während der Entwicklung können auch der **Support** und die **Community** eines Frameworks wichtig werden. Da Android unter den mobilen Betriebssystemen im Jahr 2011 eine Vormachtstellung mit einem Marktanteil von fast 50 Prozent einnimmt [Can15] und zudem quelloffen ist, gibt es auch abseits der offiziellen Dokumentation eine große Community und viele Foren, bei denen man Hilfe suchen kann. Die Web-Frameworks können u.a. auch auf Grund ihres geringen Alters keine so großen Communities vorweisen. Dafür bieten diese zum Teil einen kostenpflichtig Support, so dass man einen direkten Kontakt zu den Entwicklern der Frameworks bekommt. Vor der Wahl eines Frameworks sollte man sich daher die Stärke der Community und die Möglichkeit eines kostenpflichtigen Supports, sowie dessen Preise und Leistungen, anschauen.

Die **Verbreitung einer Applikation** spielt insofern eine wichtige Rolle, als dass ein Entwickler seine Anwendung in der Regel einer breiten Masse zugänglich machen will. Eine ideale Plattform für diesen Zweck ist im Falle von Android der Android Market. Mobile Webseiten lassen sich nicht in den Market stellen. Jedoch lassen sich diese auf unterschiedliche Weise zu Web Applikationen transformieren. Dies lässt sich unter Android mit Hilfe eines WebViews selbst programmieren, doch braucht man dann mehr Kenntnisse vom Android SDK und von Java oder von einem Bridge-Framework wie *PhoneGap*, mit welchem dies schnell und einfach möglich ist.

Die **persönlichen Fähigkeiten eines Entwicklers** haben einen großen Einfluss auf die Wahl der Umsetzung. Hat dieser schon eine jahrelange Erfahrung in Java und fühlt sich überhaupt nicht wohl, wenn er in JavaScript programmieren muss, wird die Wahl wohl immer auf die native Entwicklung fallen. Dennoch sollte sich solch ein Entwickler Markup-basierte Web-Frameworks wie *jQuery Mobile* anschauen, da man in diesem die Logik, anders als im deklarativen *Sencha Touch*, nicht zwingend in JavaScript schreiben muss.

Die **Persistenz**, welche in der Motivation dieser Arbeit als Entscheidungsfaktor vermutet wurde, hat keinen Einfluss auf die Wahl der Entwicklung. In der Arbeit wurde gezeigt, dass webbasierte Technologien gleich mehrere Methoden zum Speichern von Daten anbieten.

Auch die in der Motivation erwähnte **Konnektivität** hat sich als kein entscheidender Faktor erwiesen. Zwar sind mobile Webseiten auf den ersten Blick nur erreichbar, wenn eine Internetverbindung besteht, doch kann man diese mit Hilfe der *Offline Application Cache API* lokal speichern und offline verfügbar machen.

8.7 Anwendungsfälle

Aus den im vorherigen Kapitel beschriebenen Entscheidungsfaktoren lässt sich eine Vielzahl von Anwendungsfällen konstruieren, von denen zumindest ein paar erwähnt werden sollen.

Anwendungsfall 1: IP Cam – Es soll eine Applikation geschrieben werden, die über ein Socket den Stream der Kamera bereitstellt (siehe 5.5.2).
Entscheidung: Es wird die native Entwicklung gewählt. Mobile Webseiten können generell ausgeschlossen werden, weil mit diesen kein Zugriff auf die Kamera besteht. Die Entwicklung einer Web Applikation wäre prinzipiell möglich, doch ist die Anwendung relativ komplex und ein Debuggen einer Web Applikationen, die auf Android APIs zugreift, nur schwer möglich bzw. nicht ausreichend genug.

Anwendungsfall 2: Es soll eine Anwendung geschrieben werden, in der man ToDo-Punkte abspeichert. Die Anwendung soll nur für den persönlichen Gebrauch sein und wird nur im heimischen drahtlosem Netzwerk genutzt.
Entscheidung: Es wird eine mobile Webseite erststellt, da die Anwendung mit Hilfe der Web-Frameworks innerhalb kürzester Zeit fertig gestellt werden kann. Insbesondere die Erstellung der Benutzeroberfläche geschieht wesentlich schneller als die mit dem Android SDK. Eine Web Applikation ist nicht notwendig, da die Applikation nur für den persönlichen Gebrauch gedacht ist und daher keine Verbreitung über den Android Market stattfinden muss.

Anwendungsfall 3: Nun soll die im Anwendungsfall 2 beschriebene ToDo-App auch unterwegs verfügbar sein, wobei mit Verbindungsabbrüchen zu rechnen ist.
Entscheidung: Die Entscheidung fällt weiterhin auf die mobile Webseite. Die Entwicklung geht genauso schnell von statten. Um die App auch bei Verbindungsabbrüchen nutzen zu können, wird sie lediglich durch die *Offline Applikcation Cache API* erweitert.

Anwendungsfall 4: Die in Anwendungsfall 2 beschriebene ToDo-App soll nun installiert werden können, damit die Anwendung ein Icon erhält und man eine Verknüpfung auf den Homescreen legen kann.
Entscheidung: Es wird eine Web App entwickelt. Die Vorteile der mobilen Webseite werden ergänzt durch die Möglichkeiten des Wrappens der Anwendung. Dadurch kann man ein Icon hinzufügen und die Anwendung als APK-Datei kompilieren und diese anschließend installieren.

Anwendungsfall 5: Es soll ein Navigations-App für PKWs entwickelt werden.
Entscheidung: Da die Anwendung leistungskritisch ist, wird sie mit den nativen Entwicklungstools realisiert.

Anwendungsfall 6: Es soll eine Anwendung geschrieben werden, die die aktuelle Position als Längen- und als Breitengrad ausgibt. Dem Programmierer steht der Code des Beispiels zur Geolocation API aus Kapitel 6.3.4 zur Verfügung.
Entscheidung: Da der Programmierer den Code des Geolocation Beispiels hat, wird eine mobile Webseite entwickelt. Die Portierung des Beispiels ist mit Hilfe eines Web-Frameworks sehr einfach und zügig zu erreichen. Zusätzlich wird die *Offline Application Cache API* genutzt, damit die Anwendung unterwegs ggf. auch ohne Internetverbindung auskommt.

Anwendungsfall 7: Ein Auftraggeber möchte eine Applikation, mit der sich die Heizung seines Hauses steuern lässt. Er besitzt ein älteres Smartphone, auf dem Android 1.5 läuft.
Entscheidung: Die Anwendung wird nativ entwickelt, da Web-Frameworks insbesondere auf Geräten mit älterer Hardware Mängel bzgl. der Performanz grafischer Effekte aufweisen. Eine webbasierte Anwendung würde daher auf dem Gerät des Auftraggebers nicht flüssig laufen.

Anwendungsfall 8: Ein Sportverein möchte für seine Mitglieder und Fans eine App haben, über die aktuelle Meldungen, Termine, Fotos, Kontaktdaten und Anfahrtspläne kommuniziert werden können. Ein Vereinsmitglied betreibt bereits die Webseite des Vereins in Eigenregie. Es ist weder großes Knowhow noch Budget vorhanden.

Entscheidung: Die Entscheidung fällt auf eine mobile Webseite, für dessen Wartung das bisherige Knowhow ausreicht. Die mobile Webseite ist zudem plattformunabhängig, so dass diese auf nahezu jedem Endgerät eines Mitglieds und Fans lauffähig ist. Eine native Entwicklung für alle unterschiedlichen Betriebssysteme der Mitglieder und Fans würde länger dauern und ein größeres Budget erfordern. Je nach Anzahl der Mitglieder und Fans könnte es sich unter Umständen lohnen das Knowhow von Bridge-Frameworks anzueignen, damit die mobile Webseite zu einer Web-App transformiert werden kann. Diese könnte anschließend beispielsweise in den Android Market gestellt werden, so dass die Verbreitung der App bei einer großen Mitglieder- und Fananzahl vereinfacht wird.

9 Fazit und Ausblick

Leistungskritische Anwendungen erfordern zum Zeitpunkt des Verfassens noch immer eine native Entwicklung. Google versucht Android stetig weiterzuentwickeln. Beispielsweise haben Google und Intel vor kurzem angekündigt, dass Android ab 2012 auch auf Intel Atom Prozessoren lauffähig sein wird [tel11], was eine offizielle Öffnung der x86-Plattform für Android entspricht. Dadurch besteht eine nicht geringe Wahrscheinlichkeit, dass der Android Emulator in zukünftigen Versionen signifikant performanter sein wird. Dies würde die ohnehin schon hervorragenden Debugging-Möglichkeiten der nativen Entwicklung weiterhin verbessern. Wenn man eine komplexe Anwendung, die eine hohe Anfälligkeit für Bugs besitzt, schreibt, ist auf Grund der in der nativen Entwicklung gegebenen Debugging-Möglichkeiten zu raten, die Anwendung auch nativ zu entwickeln.

Dennoch hat die Arbeit gezeigt, dass webbasierte Techniken eine echte Alternative zur nativen Entwicklung darstellen. Es existieren sogar Anwendungsfälle, in denen eine webbasierte Entwicklung wesentlich effektiver ist. Gleichwohl erfordern anspruchsvolle webbasierte Benutzeroberflächen Geräte mit aktueller Hardware, da deren Nutzung ansonsten nicht flüssig ist. Im selben Atemzug ist zu erwähnen, dass die Web-Frameworks allesamt noch recht jung sind und deren Entwicklung, ebenso wie die Entwicklung der Hardware an sich, zügig voran geht. Beispielsweise hat *Sencha* vor kurzem ein Update angekündigt, welches einen enormen Performance-Schub verspricht [Sen11]. Auch arbeitet das W3C an neuen und weiteren Spezifikationen, durch die ein Webbrowser zukünftig Zugriff auf die Kamera [W3C118] erhält oder Benachrichtigungen versenden kann [W3C119]. Auf Grund dieser sich immer weiter verbessernden Voraussetzungen ist davon auszugehen, dass die webbasierte Entwicklung eine immer größere Rolle einnimmt und immer mehr Entwickler auf diesen Zug aufspringen werden.

Die Arbeit hat sowohl die Unterschiede als auch die Vor- und Nachteile der Umsetzung nativer mobiler Anwendungen im Vergleich zu webbasierten Technologien evaluiert, so dass sie eine Entscheidungshilfe bei der Wahl der Umsetzung gibt. Das Verstehen der Entscheidungsfaktoren ist wichtig, um den Nutzen für die Anwender und somit letztendlich den Unternehmenserfolg maximieren zu können.

10 Literaturverzeichnis

10.1 Literatur

[Bec09] Arno Becker, Marcus Pant: *Android - Grundlagen der Programmierung*. , München, 2009.

[För11] Klaus Förster, Bernd Öggl: *HTML5 Guidelines for Web Developers*. Addison-Wesley Professional, Amsterdam, 2011.

[Ful11] Steve Fulton, Jeff Fulton: *HTML5 Canvas*. O'Reilly Media, Inc., Sebastopol, 2011.

[Gil10] Zoe Mickley Gillenwater: *Stunning CSS3*. New Riders, Berkeley, 2010.

[Hog10] Brian P. Hogan: *HTML5 and CSS3: Develop with Tomorrow's Standards Today*. Pragmatic Bookshelf, Raleigh, 2010.

[Hol11] Anthony T. Holdener III: *HTML5 Geolocation*. O'Reilly Media, Inc., Sebastopol, 2011.

[Law11] Bruce Lawson, Remy Sharp: *Introducing HTML5*. New Riders, Berkeley, 2011.

[Lou01] Dirk Louis: *C++ - leicht, klar, sofort*. Markt+Technik, München, 2001.

[Wem11] Faithe Wempen: *HTML5 Step by Step*. Microsoft Press Corp., Sebastopol, 2011.

10.2 Online-Quellen

[And11] Android-Hilfe.de: Warum ist der Android Emulator so langsam? http://www.android-hilfe.de/android-app-entwicklung/88133-warum-ist-der-android-emulator-so-langsam.html, Version vom: 16. 09 2011.

[App11] Apple Inc.: Safari on iOS, Mac OS X, and Windows http://developer.apple.com/devcenter/safari/index.action, Version vom: 23. 08 2011.

[App111] Apple Inc.: Configuring Web Applications http://developer.apple.com/library/safari/#documentation/AppleApplications/Reference/SafariWebContent/ConfiguringWebApplications/ConfiguringWebApplications.html#//apple_ref/doc/uid/TP40002051-CH3-SW3, Version vom: 24. 09 2011.

[Can15] Canalys: Android takes almost 50% share of worldwide smart phone market http://www.canalys.com/newsroom/android-takes-almost-50-share-worldwide-smart-phone-market, Version vom: 15. 08 2011.

[Dev11] Alexis Deveria: When can I use IndexedDB? http://caniuse.com/indexeddb, Version vom: 11. 09 2011.

[dex11] dex2jar Project: dex2jar http://code.google.com/p/dex2jar, Version vom: 09. 09 2011.

[Dro11] DroidDraw: DroidDraw http://www.droiddraw.org, Version vom: 12. 09 2011.

[Fin11] Wendy Finn: Android Market vs iPhone App Store http://www.brighthub.com/mobile/google-android/articles/74976.aspx, Version vom: 13. 09 2011.

[Goo09] Google Inc.: Sensor | Android Developer
http://developer.android.com/reference/android/hardware/Sensor.html, Version vom: 24. 09 2009.

[Goo11] Google Inc.: FAQ for web developers
http://www.google.com/chrome/intl/en/webmasters-faq.html, Version vom: 23. 08 2011.

[Goo111] Google Inc.: Download the Android NDK
http://developer.android.com/sdk/ndk/index.html, Version vom: 15. 08 2011.

[Goo1110] Google Inc.: ADT Plugin for Eclipse
http://developer.android.com/sdk/eclipse-adt.html, Version vom: 15. 09 2011.

[Goo1111] Google Inc.: Best Practices for Web Apps
http://developer.android.com/guide/webapps/best-practices.html, Version vom: 24. 09 2011.

[Goo1112] Google Inc.: Android Browser does not support multitouch events
http://code.google.com/p/android/issues/detail?id=11909, Version vom: 25. 09 2011.

[Goo112] Google Inc.: What is the NDK?
http://developer.android.com/sdk/ndk/overview.html, Version vom: 15. 08 2011.

[Goo113] Google Inc.: Installing the SDK
http://developer.android.com/sdk/installing.html, Version vom: 14. 09 2011.

[Goo114] Google Inc.: What is Android?
http://developer.android.com/guide/basics/what-is-android.html, Version vom: 13. 09 2011.

[Goo115] Google Inc.: Developer Resources
http://developer.android.com/resources/index.html, Version vom: 13. 09 2011.

[Goo116] Google Inc.: Android Community
http://source.android.com/community/, Version vom: 13. 09 2011.

[Goo117] Google Inc.: Camera | Android Developers
http://developer.android.com/reference/android/hardware/Camera.html, Version vom: 05. 09 2011.

[Goo118] Google Inc.: Android Debug Bridge
http://developer.android.com/guide/developing/tools/adb.html, Version vom: 15. 09 2011.

[Goo119] Google Inc.: Debugging
http://developer.android.com/guide/developing/debugging/index.html, Version vom: 15. 09 2011.

[JD11] JD: JD | Java Decompiler
http://java.decompiler.free.fr, Version vom: 14. 09 2011.

[Laf11] Eric Lafortune: ProGuard
http://proguard.sourceforge.net, Version vom: 08. 09 2011.

[mak11] makandra GmbH: Übersicht aller Job-Angebote auf webentwickler-jobs.de
http://www.webentwickler-jobs.de, Version vom: 25. 09 2011.

[Man11] Manager Magazin New Media GmbH: Internet Explorer: Ungeliebter Marktführer
http://www.manager-magazin.de/unternehmen/it/0,2828,661933,00.html, Version vom: 23. 08 2011.

[NBA11] NBAndroid: NBAndroid
http://www.nbandroid.org, Version vom: 12. 09 2011.

[Nit11] Nitobi: PhoneGap roadmap-planning
http://wiki.phonegap.com/w/page/28291160/roadmap-planning, Version vom: 01. 10 2011.

[Pew11] Pew Research Center: 35% of American Adults Own a Smartphone
http://pewresearch.org/pubs/2054/smartphone-ownership-demographics-iphone-blackberry-android, Version vom: 23. 08 2011.

[QSu11] Q-Success: Usage of JavaScript libraries for websites
http://w3techs.com/technologies/overview/javascript_library/all, Version vom: 30. 09 2011.

[Sen11] Sencha Inc.: Previewing Sencha Touch 2: Native Packaging and Performance
http://www.sencha.com/blog/sencha-touch-2-what-to-expect, Version vom: 30. 09 2011.

[Sim11] Simple Project: Simple
http://www.simpleframework.org, Version vom: 15. 09 2011.

[SPI11] SPIEGEL ONLINE GmbH: SPIEGEL ONLINE
http://www.spiegel.de, Version vom: 24. 09 2011.

[Squ11] Squidoo: SQLite: The Hammer For Software Problems
http://www.squidoo.com/sqlitehammer, Version vom: 27. 09 2011.

[Ste11] StepStone GmbH: Java-Entwickler: Anwendungen für überall
http://www.it-jobs.stepstone.de/content/de/de/b2c_Java_Entwickler_Anwendungen_fuer_ueberall.cfm, Version vom: 12. 09 2011.

[tel11] teltarif.de Onlineverlag GmbH: 2012 soll offizielle Android-Version für Intel-Chips kommen
http://www.teltarif.de/intel-atom-android-google-betriebssystem/news/43956.html, Version vom: 15. 09 2011.

[The11] The Android Open Source Project: Bytecode for the Dalvik VM
http://www.netmite.com/android/mydroid/dalvik/docs/dalvik-bytecode.html, Version vom: 14. 09 2011.

[The111] The Chromium Project: Google Chrome Developer Tools
http://www.chromium.org/devtools, Version vom: 25. 09 2011.

[The112] The jQuery Project: jQuery Mobile Framework
http://jquerymobile.com, Version vom: 30. 09 2011.

[TUM11] TUM - Lehrstuhl Informatik II: Einführung in die Informatik 1
http://www2.in.tum.de/hp/Main?nid=168, Version vom: 12. 09 2011.

[TUM111] TU München: TUM in Rankings
http://portal.mytum.de/cop/ranking, Version vom: 12. 09 2011.

[Voo11] Voormedia: Obfuscate HTML scrambler
http://www.voormedia.com/en/tools/html-obfuscate-scrambler.php, Version vom: 24. 09 2011.

[W3C11] W3C: Geolocation API Specification
http://dev.w3.org/geo/api/spec-source.html, Version vom: 24. 08 2011.

[W3C111] W3C: The canvas element
http://www.w3.org/TR/html5/the-canvas-element.html, Version vom: 24. 08 2011.

[W3C112] W3C: Web Storage
http://dev.w3.org/html5/webstorage/, Version vom: 24. 08 2011.

[W3C113] W3C: Web SQL Database
http://www.w3.org/TR/webdatabase, Version vom: 01. 09 2011.

[W3C114] W3C: Indexed Database API
http://www.w3.org/TR/IndexedDB, Version vom: 01. 09 2011.

[W3C115] W3C: CSS current work & how to participate
http://www.w3.org/Style/CSS/current-work, Version vom: 10. 09 2011.

[W3C116] W3C: W3C Confirms May 2011 for HTML5 Last Call, Targets 2014 for HTML5 Standard
http://www.w3.org/2011/02/htmlwg-pr.html, Version vom: 20. 09 2011.

[W3C117] W3C: Mobile Web Application Best Practices
http://www.w3.org/TR/mwabp/#bp-viewport, Version vom: 24. 09 2011.

[W3C118] W3C: HTML Media Capture
http://www.w3.org/TR/capture-api, Version vom: 25. 09 2011.

[W3C119] W3C: Web Notification Working Group Charter
http://www.w3.org/2010/06/notification-charter, Version vom: 25. 09 2011.

[W3r11] W3resource: JavaScript version and supported browser versions
http://www.w3resource.com/javascript/introduction/javascript-version.php, Version vom: 23. 08 2011.

[Wik111] Wikimedia Foundation Inc.: World Wide Web Consortium
http://de.wikipedia.org/wiki/World_Wide_Web_Consortium, Version vom: 24. 08 2011.

[Wik112] Wikimedia Foundation Inc.: Indexed Database API
Indexed Database API, Version vom: 01. 09 2011.

[Wik113] Wikimedia Foundation Inc.: Cascading Style Sheets
http://de.wikipedia.org/wiki/Cascading_Style_Sheets, Version vom: 03. 09 2011.

[Wik114] Wikimedia Foundation Inc.: Java (Programmiersprache)
http://de.wikipedia.org/wiki/Java_(Programmiersprache), Version vom: 04. 09 2011.

[Wik115] Wikimedia Foundation Inc.: C++
http://de.wikipedia.org/wiki/C++, Version vom: 12. 09 2011.

[Wik116] Wikimedia Foundation Inc.: Android (Betriebssystem)
http://de.wikipedia.org/wiki/Android_(Betriebssystem), Version vom: 12. 09 2011.

[Wik117] Wikimedia Foundation Inc.: Motion JPEG
http://de.wikipedia.org/wiki/Motion_JPEG, Version vom: 14. 09 2011.

[Wik118] Wikimedia Foundation Inc.: Obfuscator
http://de.wikipedia.org/wiki/Obfuscator, Version vom: 15. 09 2011.

[Wik119] Wikimedia Foundation Inc.: jQuery
http://de.wikipedia.org/wiki/JQuery, Version vom: 30. 09 2011.

[Wik23] Wikimedia Foundation Inc.: Hypertext Markup Language
http://de.wikipedia.org/wiki/Hypertext_Markup_Language, Version vom: 23. 08 2011.

[Wiś11] Ryszard Wiśniewski: android-apktool
http://code.google.com/p/android-apktool/, Version vom: 14. 09 2011.

[xda11] xda-developers: Android and Windows Mobile Developers
http://www.xda-developers.com, Version vom: 12. 09 2011.

11 Abbildungsverzeichnis