

## Mobile Entwicklung // Mobiles Lernen

### STUDIENARBEIT

Studiengang Angewandte Informatik

an der Dualen Hochschule Baden-Württemberg Stuttgart Campus Horb

Betreuer: Prof. Dr. Olaf Herden

**Tobias Maier**

1. Juni 2012

<http://studicast.tobi-maier.de>

[studicast@tobi-maier.de](mailto:studicast@tobi-maier.de)



## Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit mit dem Thema

**„Mobile Entwicklung // Mobiles Lernen“**

selbstständig verfasst und keine anderen als die angegebenen  
Quellen und Hilfsmittel verwendet habe.

Tobias Maier

Böblingen, 1. Juni 2012

## Danksagung

Mein Dank gilt in erster Linie meinem Professor Herrn Olaf Herden für die Ermöglichung dieser Arbeit und die Betreuung während der Umsetzungsphase. Er brachte mich außerdem auf die Idee, am Wettbewerb BW goes Mobile teilzunehmen. Vielen Dank für die Unterstützung hierbei und die Begleitung während des Wettbewerbs.

Weiterhin danke ich meinem Arbeitgeber, der SPIRIT/21, für die Möglichkeit, auch die eine oder andere freie Minute im Büro für das Projekt nutzen zu können.

Vielen Dank an die MFG Medien- und Filmgesellschaft Baden-Württemberg für die Ausschreibung des Wettbewerbs BW goes Mobile und die Möglichkeit für mich, mit meiner Studienarbeit daran teilzunehmen.

## Zusammenfassung

Im Rahmen dieser Studienarbeit werden zunächst die verschiedenen Plattformen mobiler Endgeräte vorgestellt. Ausführlich wird auf die Möglichkeiten der Softwareentwicklung für diese Betriebssysteme eingegangen: Vor- und Nachteile von Native Apps und Web Apps, das Konzept der Hybrid Apps, die Plattformunabhängigkeit mit den nativen Features der Plattformen zu kombinieren, sowie der Ansatz von Generated Apps.

Anschließend wird das Framework PhoneGap praktisch an einigen Anwendungsbeispielen getestet. Da es durch seine Performance nicht restlos überzeugt, werden ähnliche Programmierbeispiele anschließend mit dem Titanium SDK durchgeführt, einem weiteren Framework zur hybriden App-Entwicklung. Da sich damit allerdings nicht alle benötigten Funktionalitäten abdecken lassen, wird für die Umsetzung des Praxisprojekts PhoneGap eingesetzt.

Im zweiten Teil der Arbeit wird das Projekt StudiCast, ein Konzept zum mobilen Lernen, vorgestellt und praktisch umgesetzt. Dabei geht es darum, kurz zusammengefasste Lerninhalte aus Schule und Studium selbst mit einer App aufzunehmen, um so jederzeit und überall durch wiederholtes Anhören lernen zu können. Diese Podcasts können außerdem über eine Community mit anderen geteilt werden.

## Inhalt

<b>1.</b>	<b>Einleitung .....</b>	<b>7</b>
<b>2.</b>	<b>Mobile Entwicklung .....</b>	<b>8</b>
2.1.	Mobile Endgeräte.....	8
2.1.1.	Historie.....	8
2.1.2.	Plattformen im Vergleich .....	8
2.2.	Formen der App-Entwicklung .....	10
2.2.1.	Native Apps.....	10
2.2.2.	Web Apps .....	11
2.2.3.	Hybrid Apps .....	13
2.2.4.	Generated Apps.....	15
2.3.	App-Entwicklung mit PhoneGap .....	16
2.3.1.	Installation und Einrichtung .....	16
2.3.2.	Hallo Welt .....	17
2.3.3.	AudioRecording .....	24
2.3.4.	Arbeiten mit dem Dateisystem.....	25
2.3.5.	Oberflächengestaltung .....	28
2.4.	Zwischenbetrachtung.....	29
2.5.	App-Entwicklung mit Titanium .....	29
2.5.1.	Entwicklungsumgebung und Projekterstellung .....	30
2.5.2.	Dateisystem und Netzwerk.....	33
2.5.3.	AudioRecording .....	34
2.6.	Technologieauswahl.....	35
<b>3.</b>	<b>Mobiles Lernen .....</b>	<b>36</b>
3.1.	StudiCast .....	36
3.2.	Funktionsumfang .....	37
3.3.	Datenhaltung .....	37
<b>4.</b>	<b>Technische Umsetzung.....</b>	<b>39</b>
4.1.	Server .....	40
4.1.1.	Dateisystem .....	40
4.1.2.	MySQL.....	41
4.1.3.	Zugriffslogik.....	43
4.1.4.	Anwendungslogik / API .....	43
4.1.5.	Web-Interface .....	45

---

4.2.	App .....	46
4.2.1.	Dateisystem .....	46
4.2.2.	SQLite .....	46
4.2.3.	HTTPClient .....	47
4.2.4.	Anwendungslogik .....	47
4.2.5.	Smartphone- und Tablet-GUI .....	47
<b>5.</b>	<b>Fazit .....</b>	<b>49</b>
<b>6.</b>	<b>Verzeichnisse.....</b>	<b>50</b>
6.1.	Abbildungsverzeichnis .....	50
6.2.	Tabellenverzeichnis .....	50
6.3.	Literaturverzeichnis.....	51

## 1. Einleitung

Smartphones und Tablet-PCs sind seit Jahren unaufhaltsam auf dem Vormarsch. Ob Mobiltelefon, Kalender, E-Mail-Client, Messenger, Zeitung, Navigationsgerät, Spielzeug oder Statussymbol – die mobilen Endgeräte erfüllen sehr vielfältige Aufgaben. Von einem Trend oder gar einem Hype kann längst keine Rede mehr sein, vielmehr von einer kompletten Weiterentwicklung der Informationsgesellschaft hin zur mobilen Welt.

Die Entwicklung von Apps ist mit diesem Aufschwung zu einem Bestandteil der Softwareentwicklung geworden und wird in diesem Bereich in den kommenden Jahren wohl immer gewichtiger.

Im Rahmen dieser Studienarbeit soll zunächst das Thema Mobile Entwicklung, das heißt Softwareentwicklung für mobile Endgeräte, durchleuchtet werden. Im ersten Teil geht es deshalb um die eben genannten Geräte sowie die unterschiedlichen Plattformen, die derzeit am Markt gehandelt werden. Anschließend werden die Möglichkeiten vorgestellt, Anwendungen für diese Plattformen zu entwickeln, besonders auf die Vor- und Nachteile von Native Apps und Web Apps sowie Mischformen wird detailliert eingegangen.

Im zweiten Teil folgt ein Konzept zum Mobilem Lernen. Damit sollen Smartphones und Tablets als sinnvolles Hilfsmittel für Schüler und Studenten etabliert werden, die sie beim Lernprozess unterstützen können. Die Idee StudiPodcasts wird praktisch umgesetzt und durch diese Studienarbeit dokumentiert.

## 2. Mobile Entwicklung

### 2.1. Mobile Endgeräte

Genau genommen zählen laut Bundesamt für Sicherheit in der Informatik [1] sämtliche Mobiltelefone, Smartphones, PDAs, Laptops und Tablet-PCs zu den mobilen Endgeräten – eben alle Geräte, die man als Benutzer einfach mit sich führen kann. Landläufig sind damit jedoch nur Smartphones und Tablets gemeint. Und genau darauf beschränkt sich auch diese Studienarbeit.

#### 2.1.1. Historie

Die Entstehung heutiger Smartphones wird auf den Nokia Communicator zurückgeführt. Diese Serie von Geräten wurde seit 1996 von Nokia vertrieben. Daraus bzw. dafür entwickelte Nokia das Betriebssystem Symbian OS, das eine Vorreiterrolle unter den Smartphone-Betriebssystemen einnehmen konnte. Allerdings wurden die damaligen Smartphones noch klassisch mit Tasten oder Touch-Stiften bedient.

Die Multitouch-Oberfläche, die heute sehr eng mit dem Bild des Smartphones verbunden ist, begann ihren Siegeszug im Januar 2007, als Apple sein erstes iPhone auf den Markt brachte. Die Technologie existierte zwar schon seit einigen Jahren, erreichte aber mangels geeigneter Geräte keine breite Öffentlichkeit. Parallel zu Apple drängten etliche weitere Hersteller von Smartphones bzw. Betriebssystemen auf den Markt. Relativ früh war hierunter bereits Microsoft mit Windows Mobile und später Windows Phone. Seit 2009 spielt außerdem das auf Linux basierende Android eine wachsende Rolle im Wettstreit der mobilen Betriebssysteme.

#### 2.1.2. Plattformen im Vergleich

##### Apple iOS

Apple iOS ist das Betriebssystem aller mobilen Apple-Produkte, dazu gehören das iPhone, der iPod touch und das iPad. iOS hieß bis 2010 iPhone OS. Es basiert auf dem ebenfalls von Apple vertriebenen Mac OS X. Entwickler haben die Möglichkeit, Apps in den Sprachen C und Objective-C zu programmieren.

## **bada**

bada ist das hauseigene Betriebssystem von Samsung. Es wird allerdings nicht so konsequent mit den Geräten von Samsung ausgeliefert wie beispielsweise iOS von Apple. Entsprechend gering ist auch die Verbreitung der Plattform. Apps können in C und C++ geschrieben werden.

## **BlackBerry OS**

Das BlackBerry OS wird von der Firma RIM vertrieben und ist für den Einsatz in BlackBerrys optimiert. Damit kommt es vor allem im Unternehmensumfeld sehr häufig zum Einsatz. Anwendungen für BlackBerry können in Java entwickelt werden.

## **Android**

Android ist der derzeit populärste Linux-Vertreter unter den mobilen Betriebssystemen. Es wird von der Open Handset Alliance entwickelt, hinter der vor allem Google steckt. Nokias MeeGo oder webOS von Palm sind weitere Beispiele für mobile Linux-Derivate. Android war zunächst für den Einsatz in Smartphones optimiert. Ein zweiter Versionsbaum deckte Tablets ab. Ab Version 3 wurden sie zu einer Version sowohl für Smartphones als auch für Tablets vereinigt. Für die App-Entwicklung können die Sprachen Java, C und C++ verwendet werden.

## **Symbian**

Symbian ist der Nachfolger von Symbian OS. Nokia stellte die Weiterentwicklung jedoch 2011 ein, da die Technologie zu weit hinter den Konkurrenten zurücklag. Für die letzten Versionen können Apps in den Sprachen Java, C und C++ geschrieben werden.

## **Windows Phone 7**

Windows Phone 7 ist das aktuelle Microsoft-Betriebssystem für Smartphones. Es tritt mit dem Versuch an, die vom Desktop gewohnte Windows-Umgebung auf das Smartphone zu transferieren. Apps für Windows Phone werden in der „Microsoft-Sprache“ C# programmiert.

## 2.2. Formen der App-Entwicklung

### 2.2.1. Native Apps

Native Apps sind Anwendungen, die speziell für ein bestimmtes Betriebssystem in einer dafür vorgesehenen Programmiersprache geschrieben werden. Beispielsweise wäre eine Native App für ein iOS-Gerät typischerweise in Objective-C oder eine App für Android in Java programmiert. Vor allem in den Anfangszeiten der mobilen Entwicklung kam diese Variante fast ausschließlich zum Einsatz. Mittlerweile überwiegen jedoch in vielen Einsatzszenarien die Vorteile der anderen (weiter unten beschriebenen) Verfahren.

Der größte Vorteil von Native Apps ist, dass sie auf sämtliche Funktionen des Betriebssystems bzw. des Geräts zugreifen können und damit für den Entwickler die meisten Möglichkeiten bieten. Da sie in der Sprache programmiert sind, für die das Betriebssystem optimiert ist, werden sie außerdem im Vergleich zu den meisten anderen Entwicklungsverfahren sehr schnell ausgeführt. Native Apps kommen also vor allem dort zum Einsatz, wo einerseits eine hohe Performanz gefragt ist und andererseits Funktionen benötigt werden, auf die nur nativ zugegriffen werden kann.

Im Umkehrschluss ist ein häufig bemängelter Nachteil der Native Apps, dass sie für ein einziges Betriebssystem entwickelt bzw. optimiert werden. Soll eine Anwendung auf verschiedenen Plattformen zum Einsatz kommen, muss sie zumindest zu einem gewissen Teil mehrfach entwickelt werden. Das hat zur Folge, dass anschließend auch mehrere Versionen der App gepflegt und weiterentwickelt werden müssen.

Ein weiterer Nachteil ist, dass der Entwickler die entsprechende Sprache beherrschen muss und sich an die Richtlinien der jeweiligen Plattform bzw. dessen Anbieter halten muss. So muss man beispielsweise als Entwickler bei Apple registriert sein (was unter anderem mit Kosten verbunden ist), um Native Apps für iOS zu programmieren. Speziell bei der App-Entwicklung für iOS braucht der Entwickler außerdem einen Mac, um die Entwicklungsumgebung Xcode nutzen zu können. Die Entwicklung für andere Betriebssysteme ist nicht ganz so restriktiv geregelt. Dennoch muss sich der Programmierer mit den einzelnen Spezifikationen jeder Plattform auskennen.

## 2.2.2. Web Apps

Web Apps sind in der Welt der App-Entwicklung das Gegenstück zu Native Apps. Der Begriff wurde 2007 von Steve Jobs geprägt, der damals bereits den Paradigmenwechsel von Native auf Web Apps vorhersagte oder vielmehr vorantreiben wollte. Das Prinzip ist eng verbunden mit dem sogenannten Mobilem Internet, dessen großer Aufschwung in Verbindung mit Smartphones und Tablets ebenfalls zu den großen Leistungen Steve Jobs gerechnet wird.

Dabei wird keine App mehr auf dem Gerät installiert, wie man es klassisch von Software auf dem Computer kennt, sondern die komplette Anwendung in eine Website verpackt – inklusive Datenhaltung, Logik und Darstellung. Die App wird in ihrem Aussehen und ihrer Benutzbarkeit so gut wie möglich an das mobile Endgerät, also beispielsweise ein Smartphone, angepasst, sodass der Benutzer kaum einen Unterschied zu einer Native App feststellt. Der Begriff der Userexperience konnte sich hierfür durchsetzen.

Und gemeint damit ist auch nicht, dass eine Website für Smartphones optimiert wird, was etwa die Seitenformate oder Größe und Anordnung von Bildern angeht, sondern tatsächlich das Look & Feel einer App annimmt, inklusive typischer Farbgestaltung, Buttons mit entsprechender gewohnter Funktionalität usw.

Programmiert werden Web Apps in klassischen Sprachen der Webentwicklung, das heißt HTML, JavaScript und CSS. Dabei lohnt sich der Einsatz spezieller Frameworks. Bekannte Beispiele hierfür sind iWebKit (für iOS-Geräte), Sencha Touch oder jQuery Mobile (die relativ junge mobile Ausgabe des sehr weit verbreiteten JavaScript-Frameworks jQuery). Die Frameworks dienen vor allem der Abstraktion. So muss sich der Entwickler nicht bei jeder neuen Web App darum kümmern, Buttons und Layouts entsprechend der gewünschten Zielplattform zu gestalten, sondern überlässt dies dem Framework.

Hierin liegt nun schon der erste große Vorteil von Web Apps gegenüber Native Apps: Die Programmiersprache. HTML, JavaScript und CSS sind absolute Grundlagen für jeden Programmierer. Sie sind so weit verbreitet, so gut dokumentiert und so leicht erlernbar, dass damit eigentlich jeder Entwickler Apps schreiben kann. Unterstützt durch die vorhandenen Frameworks kann hier sehr viel Zeit und Aufwand eingespart werden.

Ein weiterer Hauptvorteil ist die Plattformunabhängigkeit. Der eigentliche Code wird nicht für ein einziges Betriebssystem geschrieben – selbst die Anpassung von Oberflächen an das jeweilige Design des Smartphones übernimmt meistens ein Framework. Damit muss eine App nur ein einziges Mal programmiert werden und kann fast alle Plattformen abdecken. Bei entsprechender Herangehensweise kann mit demselben Code sogar noch eine klassische Website für „herkömmliche“ Browser realisiert werden.

Hinzu kommt, dass Web Apps nicht auf dem Smartphone oder Tablet installiert werden müssen, sondern nur wie jede andere Website geladen werden, was in etwa dem gewohnten Startvorgang einer Native App entspricht. Gleichzeitig müssen Web Apps nicht über die jeweiligen AppStores der Plattform-Betreiber vertrieben werden, sondern können von beliebigen Seiten aus dem Internet aus gestartet werden.

Dahinter verbirgt sich allerdings auch eine Schwachstelle, mit der die einzelnen Betriebssysteme unterschiedlich gut umgehen. Unter Android gibt es beispielsweise ein Menü, in dem alle installierten Apps zu finden sind und von wo aus man jede Anwendung starten kann. Web Apps werden dort nicht zu finden sein. Man kann sie höchstens mit einer Art Shortcut auf dem Homescreen ablegen, nicht jedoch im Menü. Auch das Anlegen eines solchen Shortcuts ist nicht mit einem Klick getan. Unter iOS ist dieses Problem besser gelöst. Dort gibt es erstens das „Android-App-Menü“ nicht und zweitens können Shortcuts auf eine Web App mit minimalem Aufwand direkt aus dem Browser heraus angelegt werden.

Als Nachteil wird von manchen Entwicklern genannt, dass Web Apps ständig Zugriff auf das Internet benötigen. Das ist nicht richtig. Aktuelle Webtechnologien sind in der Lage, dieses Problem zu umgehen. Mit HTML5 wurde beispielsweise der Local Storage eingeführt. Damit kann eine Web App offline ausgeführt werden und Daten so lange auf dem Gerät zwischenspeichern, bis sie wieder Zugriff auf das Internet erhält.

Ein tatsächlicher Nachteil ist jedoch der stark eingeschränkte Zugriff auf Funktionen des Betriebssystems bzw. des Geräts. So ist es mit einer Native App beispielsweise problemlos möglich, das Gerät vibrieren zu lassen, was mit einer reinen Web App nicht funktioniert. Einige Schnittstellen wie der Abruf von GPS-Daten, das Auslösen eines Anrufs oder einer E-Mail werden zwar über den Browser zur Verfügung gestellt und können somit per JavaScript genutzt werden. An die Möglichkeiten einer Native

App kommt eine Web App damit allerdings nie heran. Und genau das ist die größte Schwachstelle und für etliche Anwendungsszenarien ein Ausschlusskriterium.

Weiterhin spricht gegen den Einsatz von Web Apps die vergleichsweise geringere Performanz gegenüber Native Apps. Da Web Apps interpretiert werden und nicht als kompilierte Programme in Maschinencode vorliegen, können sie an etlichen Stellen nicht die Ausführungsgeschwindigkeit von Native Apps erreichen.

Heiko Behrens [2] benutzt in seinem Vortrag für den Vergleich von Native und Web Apps ein hervorragend passendes Bild aus dem Film Avatar: Die Native Apps sind in diesem Beispiel die Ureinwohner (was dem Ursprung des Wortes auch gar nicht so fern liegt). Sie haben ihre eigene Sprache (Programmiersprache), ihre eigene Kultur (Programmierparadigmen, Vorgehensweisen, Best Practices) und ein eigenes Ökosystem (Zusammenspiel von Komponenten, Distributionskanäle usw.). Sie sind für einen Fremden damit schwer zu verstehen, aber sie haben sich auf ihrer Plattform als geeignete Lebensform durchgesetzt und kommen somit am besten mit den gegebenen Ressourcen und Möglichkeiten zurecht.

Die andere Seite, das sind im übertragenen Sinne die Web Apps, sind die Invasoren („not-so-native“). Sie sind lediglich an der Wirtschaftlichkeit interessiert (Plattformunabhängigkeit, Vereinheitlichung, geringer Entwicklungsaufwand, Gewinnmaximierung usw.). Damit streben sie nicht die Lösung an, die für das jeweilige System die beste ist, sondern die aus wirtschaftlichen Bestrebungen heraus am effizientesten ist. Auf lange Sicht werden sich damit die Web Apps durchsetzen.

### 2.2.3. Hybrid Apps

Hybrid Apps sind ein völlig neuer Ansatz, die Vorteile von Native und Web Apps zu verbinden und möglichst viele Nachteile auszublenden. Das Prinzip ist einfach: Der Entwickler programmiert eine Web App und bettet diese in eine rudimentäre Native Browser-App ein, die als Schnittstelle zwischen Web App und Betriebssystem dient und als Native App Zugriff auf alle Systemfunktionalitäten erhält.

Vorreiter im Bereich der Hybrid Apps ist PhoneGap, ein von Nitobi Software initiiertes OpenSource-Projekt, das mittlerweile von Adobe aufgekauft wurde und vermarktet bzw. weiterentwickelt wird. PhoneGap ist ein Framework, das dem Entwickler eine ganze Menge an Schnittstellen zur Verfügung stellt. So kann beispielsweise mit ein-

fachen JavaScript-Aufrufen auf das Dateisystem eines Gerätes, den Kompass oder die Kamera zugegriffen werden, was eine reine Web App nicht könnte. Den Nativen Teil der App, also den Minibrowser mit Schnittstellen, liefert PhoneGap für etliche Plattformen mit. Dazu gehören aktuell iOS, Android, Blackberry OS, WebOS, Windows Phone 7, Symbian und Bada. Manche Features können jedoch nicht mit allen Betriebssystemen oder Geräten genutzt werden.

Das Ergebnis einer solchen Hybrid App ist schon sehr gut. Der Entwickler braucht quasi keinerlei Erfahrung außer den üblichen Webtechnologien. Die Einarbeitung in die PhoneGap-API ist dank guter Dokumentation relativ einfach. Damit kann schnell etwas produziert werden, das gut aussieht und funktioniert. Dem Benutzer wird bei genauem Hinschauen allerdings auffallen, dass es sich nicht um eine Native App handelt. Ebenso lässt – wie die eigenen Versuche zeigen – die Geschwindigkeit zu wünschen übrig. Das Ziel ist dennoch erreicht: PhoneGap vereint alle Vorteile der Web App-Entwicklung mit der Funktionsvielfalt von Native Apps und der Möglichkeit, die App mit eigenem Icon selbst unter Android im entsprechenden Menü zu finden. Lediglich der Performanz-Nachteil bleibt.

Ein zweites sehr interessantes Beispiel für Hybrid Apps ist der Ansatz von Appcelerator, für den manche Quellen bereits eine eigene Kategorie „Interpretierte Apps“ eingeführt haben. Ähnlich wie PhoneGap stellt Appcelerator ein Framework bereit, das jedoch deutlich umfangreicher und noch nicht so gut dokumentiert ist. Entwickelt wird nicht mehr deklarativ in HTML und CSS, sondern nur noch programmatisch per JavaScript. [3]

Titanium Studio, die hauseigene Entwicklungsumgebung, bastelt daraus dann eine fertige App. Dabei wird im Hintergrund weiterhin die Web App ausgeführt, allerdings in einem unsichtbaren Browser, der mit dem Host, also dem jeweiligen Betriebssystem kommuniziert. An dieser Stelle werden aus dem JavaScript-Code nun native Oberflächen generiert. Der Benutzer erhält eine App, die komplett zu seiner Userexperience passt. Die ersten Versuche bestätigen außerdem eine deutlich bessere Ausführungsgeschwindigkeit, soll heißen die Oberfläche ruckelt nicht wie bei vorangegangenen Tests mit Web Apps und PhoneGap.

Die aktuelle Version Appcelerator Titanium Mobile 1.8.1 unterstützt iOS und Android, wobei für Android einige Einschränkungen gelten.

Zwei weitere Vertreter der Interpretierten bzw. Hybrid Apps sind MonoTouch und rhomobile. Beide stellen eine komplette Laufzeitumgebung auf dem Gerät zur Verfügung – MonoTouch für .NET und rhomobile für ruby. Der Entwickler hat dadurch die Möglichkeit, Apps in seiner gewohnten Sprache, z.B. C#, zu schreiben. MonoTouch gibt es bislang jedoch nur für iOS und Android. Der Vorteil dabei ist ganz klar, dass in C# geschriebene Apps mit Hilfe von MonoTouch sowohl auf iOS- und Android-Geräten als auch nativ auf Windows Phone 7 sowie als Desktop-Anwendung auf jedem Windows-Rechner eingesetzt werden können.

#### 2.2.4. Generated Apps

Bei Generated Apps handelt es sich bisher noch um eine eher unbekannte Nische der App-Entwicklung. Es geht darum – ähnlich wie bei den Hybrid Apps – Code in einer einfachen, bekannten Sprache zu schreiben und diesen automatisch in eine andere Sprache umwandeln zu lassen.

Ein OpenSource-Beispiel dafür ist XMLVM [3]. Dieser Generator nimmt Quellcode in C#, Ruby oder Java entgegen und wandelt ihn über Zwischencode in eine nahezu beliebige Zielsprache (z.B. Objective-C) um. Der Vorteil ist wiederum, dass der Entwickler mit seiner „eigenen“ Programmiersprache arbeiten kann und die App mittels Generator theoretisch in Native Apps für verschiedene Plattformen umwandeln kann. Was bei diesem Prinzip nicht berücksichtigt wird, sind die Feinheiten der unterschiedlichen Plattformen. Der Code sollte zwar funktional in generiertem Objective-C dasselbe Ergebnis liefern wie in Java oder C#, allerdings ist er nicht für das entsprechende Gerät optimiert und kann deshalb die jeweiligen Stärken auch nicht voll ausschöpfen.

Ein weiterer Nachteil ist, dass der generierte Code für Menschen unlesbar ist. Es ist also unmöglich, einen gemeinsamen Basisanteil einer Anwendung in einer Sprache zu schreiben, diesen dann in die anderen benötigten Sprachen per Generator zu portieren und die individuellen Anpassungen oder Optimierungen an den generierten Codederivaten vorzunehmen.

Ein relativ junges Framework, das an dieser Stelle als weiteres Beispiel erwähnt werden soll, ist Netbiscuits Tactile. Dabei handelt es sich um ein HTML5-Framework, das Oberflächen für verschiedene mobile Plattformen generieren kann. Der Entwick-

ler schreibt hierfür Code in der proprietären, auf XML basierenden Auszeichnungssprache Biscuit ML [4]. Diese Art von Framework erzeugt allerdings keine „Generated Native App“, sondern generiert wiederum eine Web App.

Der Einsatz von Generated Apps sollte gut durchdacht werden. Für wirkliche Experten kann es dann interessant werden, wenn ein eigener, selbst entwickelter Generator zum Einsatz kommt. Dieser könnte Aufgaben wie das Gestalten von Oberflächen übernehmen und als Quellsprache z.B. mit XML arbeiten.

## 2.3. App-Entwicklung mit PhoneGap

In diesem Kapitel werden erste Entwicklungsversuche mit dem Framework PhoneGap beschrieben. Es wird bei null begonnen, das heißt mit dem Versuch, überhaupt erst einmal ein „Hallo Welt“ auf ein Smartphone zu zaubern. Später wird das Beispiel weiter ausgebaut. Besonderes Augenmerk soll dabei auf Funktionen gelegt werden, die für die StudiPodcast-App im zweiten Teil der Studienarbeit wichtig sein werden, wie beispielsweise Audioaufnahme oder das Speichern von Dateien.

### 2.3.1. Installation und Einrichtung

Die Entwicklung soll auf einem relativ schwachen, älteren Laptop (IBM ThinkPad R52, 1,8GHz, 2GB RAM, Windows XP SP3) stattfinden. Das Beispielprojekt soll auf Android 2.3.3 laufen. Für die praktischen Tests steht neben dem Android-Emulator ein HTC Desire HD mit Android 2.3.3 zur Verfügung.

Auf der Website von PhoneGap ist unter dem Punkt „Getting Started“ detailliert beschrieben, wie die Entwicklungsumgebung aufgesetzt wird. Für die Entwicklung für Android wird hierzu zunächst Eclipse Classic benötigt. Download und Installation funktionieren problemlos, ebenso die anschließende Installation des Android SDKs (Android Software Development Kit). Als Plug-In für Eclipse wird das ADT (Android Development Tools) Plug-In nachgeladen.

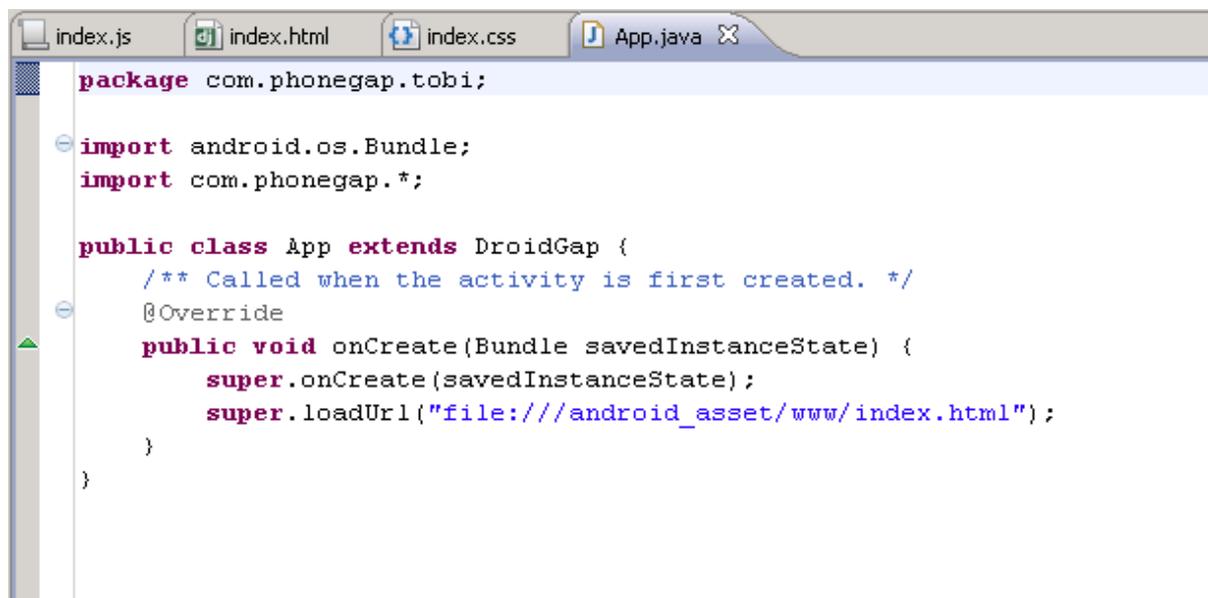
Das PhoneGap-Framework selbst ist ein 10,1MB großes ZIP-Archiv. Es trägt die Versionsnummer 1.4.1 mit Datum vom 2. Februar 2012. Das Archiv enthält neben etlichen Guides einen Ordner lib, in dem sich wiederum ein Ordner für jede unterstützte Plattform befindet.

### 2.3.2. Hallo Welt

Der PhoneGap-Dokumentation weiter folgend, wird in Eclipse ein neues „Android Project“ angelegt. Um das PhoneGap-Framework nutzen zu können, müssen im Wurzelverzeichnis des neuen Projekts die beiden Ordner `/libs` und `/assets/www` erstellt werden. Die Inhalte des `android`-Ordners des PhoneGap-Downloads werden anschließend in das Projekt kopiert:

- `phonegap.js` nach `/assets/www`
- `phonegap.jar` nach `/libs`
- der komplette `xml`-Ordner nach `/res`

Als Nächstes wird die Hauptdatei `App.java` im `src`-Ordner geöffnet und bearbeitet: Die Vererbung der Klasse `App` wird von `Activity` auf `DroidGap` geändert, die Zeile `„import com.phonegap.*“` muss ergänzt werden und die überflüssig gewordenen Imports können entfernt werden. Abbildung 1 zeigt die Datei `App.java` nach den Änderungen.



```
package com.phonegap.tobi;

import android.os.Bundle;
import com.phonegap.*;

public class App extends DroidGap {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.loadUrl("file:///android_asset/www/index.html");
    }
}
```

Abbildung 1: `App.java` nach der Bearbeitung

In Eclipse muss PhoneGap nun noch dem Buildpath hinzugefügt werden. Das geht per Rechtsklick auf den Ordner `/libs`, dort unter `Build Path` den Punkt `Configure Build Path` auswählen und im Reiter `Libraries` die Datei `phonegap-1.4.1.jar` hinzufügen. Anschließend sollten keine Probleme mehr auftreten.

Als letzte Vorbereitung muss die Datei AndroidManifest.xml bearbeitet werden. Hier werden die Berechtigungen für die App gesetzt. Die Datei sollte so aussehen:

```
<?xml version="1.0" encoding="utf-8"?>

<manifest
  xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.phonegap.tobi"
  android:versionCode="1"
  android:versionName="1.0">

  <supports-screens
    android:largeScreens="true"
    android:normalScreens="true"
    android:smallScreens="true"
    android:resizeable="true"
    android:anyDensity="true" />

  <uses-permission
    android:name="android.permission.CAMERA" />
  <uses-permission
    android:name="android.permission.VIBRATE" />
  <uses-permission
    android:name="android.permission.ACCESS_COARSE_LOCATION" />
  <uses-permission
    android:name="android.permission.ACCESS_FINE_LOCATION" />
  <uses-permission
    android:name=
      "android.permission.ACCESS_LOCATION_EXTRA_COMMANDS" />
  <uses-permission
    android:name="android.permission.READ_PHONE_STATE" />
  <uses-permission
    android:name="android.permission.INTERNET" />
  <uses-permission
    android:name="android.permission.RECEIVE_SMS" />
  <uses-permission
    android:name="android.permission.RECORD_AUDIO" />
  <uses-permission
    android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
  <uses-permission
    android:name="android.permission.READ_CONTACTS" />
  <uses-permission
    android:name="android.permission.WRITE_CONTACTS" />
  <uses-permission
    android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
  <uses-permission
    android:name="android.permission.ACCESS_NETWORK_STATE" />
  <uses-permission
    android:name="android.permission.GET_ACCOUNTS" />
  <uses-permission
    android:name="android.permission.BROADCAST_STICKY" />

  <uses-sdk android:minSdkVersion="10" />
```

```
<application
  android:icon="@drawable/ic_launcher"
  android:label="@string/app_name">

  <activity
    android:name=".App"
    android:label="@string/app_name"
    android:configChanges="orientation|keyboardHidden">

    <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category an-
droid:name="android.intent.category.LAUNCHER" />
    </intent-filter>
  </activity>

  <activity
    android:name="com.phonegap.DroidGap"
    android:label="@string/app_name"
    android:configChanges="orientation|keyboardHidden">

    <intent-filter>
    </intent-filter>
  </activity>

</application>
</manifest>
```

An dieser Stelle ist es ratsam, das komplette Projekt zu kopieren, um eine Art Vorlage für spätere Projekte zu haben.

Der eigentliche Inhalt der ersten „Hallo Welt“-App ist bezeichnend klein. In dem Verzeichnis /assets/www wird eine Datei index.html erstellt, die lediglich den folgenden Inhalt enthält:

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Hallo Welt</title>
    <script
      type="text/javascript"
      charset="utf-8"
      src="phonegap-1.4.1.js">
    </script>
  </head>
  <body>
    <h1>Hallo Welt</h1>
  </body>
</html>
```

Um die Aufbereitung des Inhalts soll sich vorerst PhoneGap kümmern. Jetzt wird es Zeit, die ersten Ergebnisse zu betrachten. Dafür gibt es grundsätzlich zwei Wege: den Android Emulator auf dem Computer oder das Smartphone. Im ersten Test soll es der Emulator sein. Dazu sind beim ersten Anlauf mehrere Schritte notwendig.

Als Erstes müssen, falls noch nicht geschehen, einige Komponenten des Android SDK nachgeladen werden. Der Android SDK Manager (in Eclipse erreichbar über das Window-Menü) ist ein sehr bequemes Werkzeug hierfür. Abbildung 2 zeigt die Oberfläche des SDK Managers.

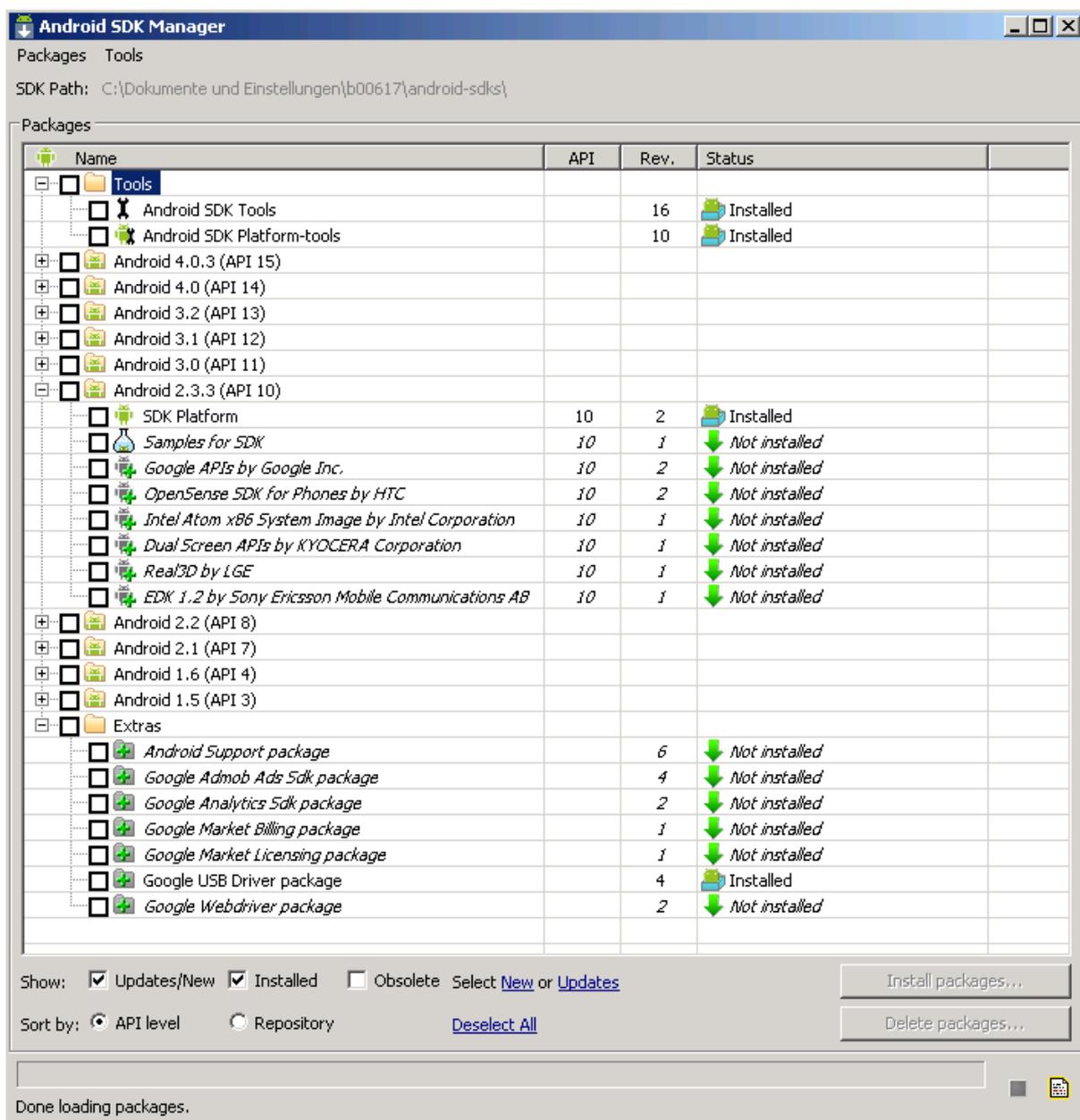
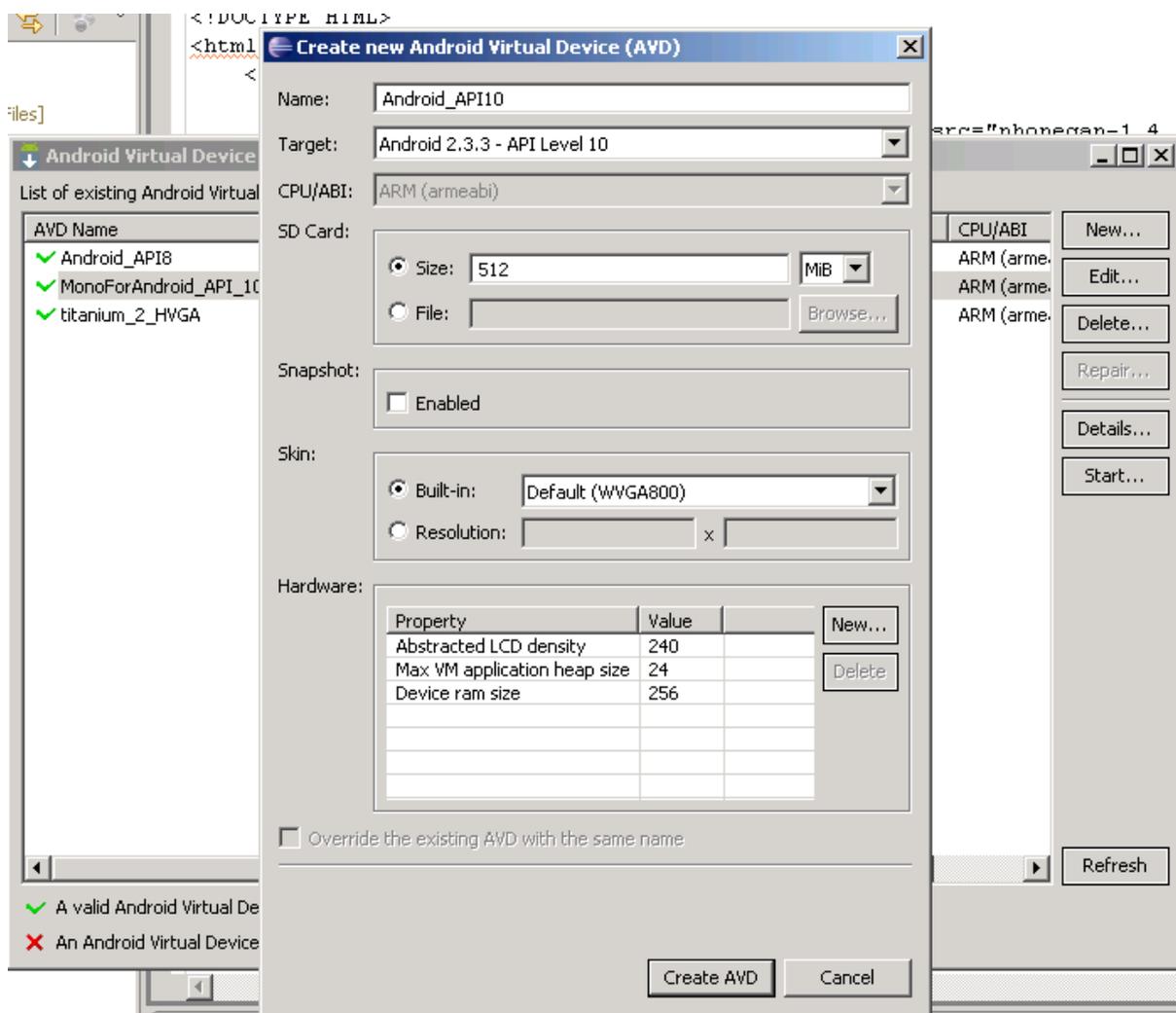


Abbildung 2: Android SDK Manager

Da die Applikation später auf einem Smartphone mit Android 2.3.3 getestet werden soll, empfiehlt es sich, auch den Simulator mit der Version 2.3.3 (API10) auszustatten. Wichtig ist, dass die Komponente SDK Platform installiert ist. Weiterhin ist es für die späteren Tests mit dem Smartphone sinnvoll, gleich das Google USB Driver package aus der Kategorie Extras mitzuintallieren.

Ist die Installation abgeschlossen, muss das eigentliche virtuelle Gerät noch angelegt werden. Dafür eignet sich der AVD (Android Virtual Device) Manager (in Eclipse ebenfalls erreichbar über das Window-Menü). Der AVD Manager ist in Abbildung 3 zu sehen.



**Abbildung 3: Android Virtual Device Manager**

Hier wird nun ein neues Virtual Device angelegt. Der Name kann beliebig gewählt werden, allerdings dürfen dabei keine Punkte oder Leerzeichen verwendet werden (deshalb hier die Bezeichnung Android\_API10 und nicht etwa Android 2.3.3). Unter Target stehen die zuvor installierten Versionen zur Verfügung. Bei SD Card kann ein

Limit für den Speicher des simulierten Geräts festgelegt werden. Die restlichen Werte sind Standardwerte.

Damit sollten nun alle Einstellungen erledigt sein und die Ausführung des Projektes kann per Run-Menü gestartet werden. Bei der oben genannten Entwicklungsumgebung mit schwachem Rechner geschieht jedoch zunächst nichts und erst nach einigen Minuten öffnet sich zuerst der Android-Simulator und darauf dann nach weiterer Wartezeit die „Hallo Welt“-App.

Bei den weiteren Versuchen stellt sich schnell heraus, dass der Simulator zwar nicht jedes Mal neu gestartet werden muss, das Starten der App und die Reaktionszeiten des Simulators jedoch trotzdem völlig inakzeptabel sind.

Als Nächstes soll die App nun also direkt aufs Smartphone geladen und dort ausgeführt werden. In den Einstellungen des Smartphones muss dafür erst das USB-Debuggen aktiviert werden (Einstellungen > Anwendungen > Entwicklung). Per USB-Kabel wird die Verbindung mit dem Rechner hergestellt.

Das Gerät wird von Windows unter den beschriebenen Systemkomponenten jedoch nicht automatisch erkannt. Dafür muss auf dem Rechner noch der passende Treiber von HTC installiert werden. Laut verschiedener Entwicklerforen im Internet ist das jedoch ein HTC-spezifisches Problem. Im Rahmen dieser Arbeit konnte diese Aussage nicht verifiziert werden. Dafür stößt der aufmerksame Suchende im Internet auf eine modifizierte Version der HTC-Software, die lediglich den benötigten Treiber enthält und nicht die komplette HTC-Sync-Suite.

Ist der Treiber installiert und das Smartphone per Kabel verbunden, genügt laut PhoneGap-Dokumentation ein Klick auf das Run-Menü und dort unter Run as auf den Punkt Android Application. In der verwendeten Umgebung ist dafür jedoch ein Zwischenschritt nötig.

Unter dem Punkt Run Configurations öffnet sich ein neues Fenster (Abbildung 4), in dem die Android Application zuerst angelegt werden muss. Vor dem Starten auf dem Smartphone ist es notwendig, im Reiter Target Manual auszuwählen.

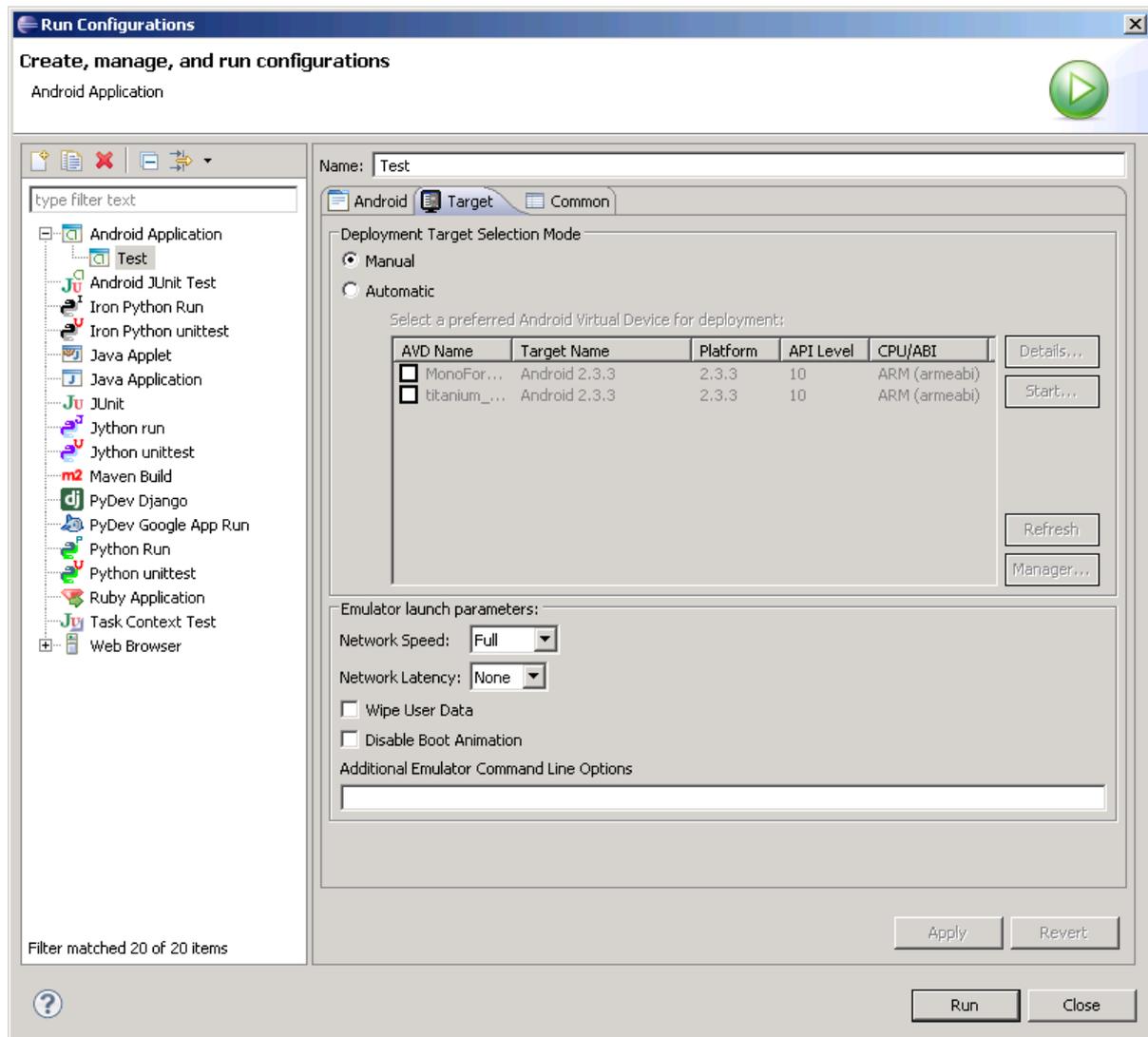


Abbildung 4: Eclipse Run Configurations

Nach Betätigen des Run-Buttons öffnet sich der sogenannte Android Device Chooser (Abbildung 5). Wenn alles richtig eingerichtet wurde, taucht hier nun tatsächlich das verbundene Smartphone auf und kann bequem ausgewählt werden.

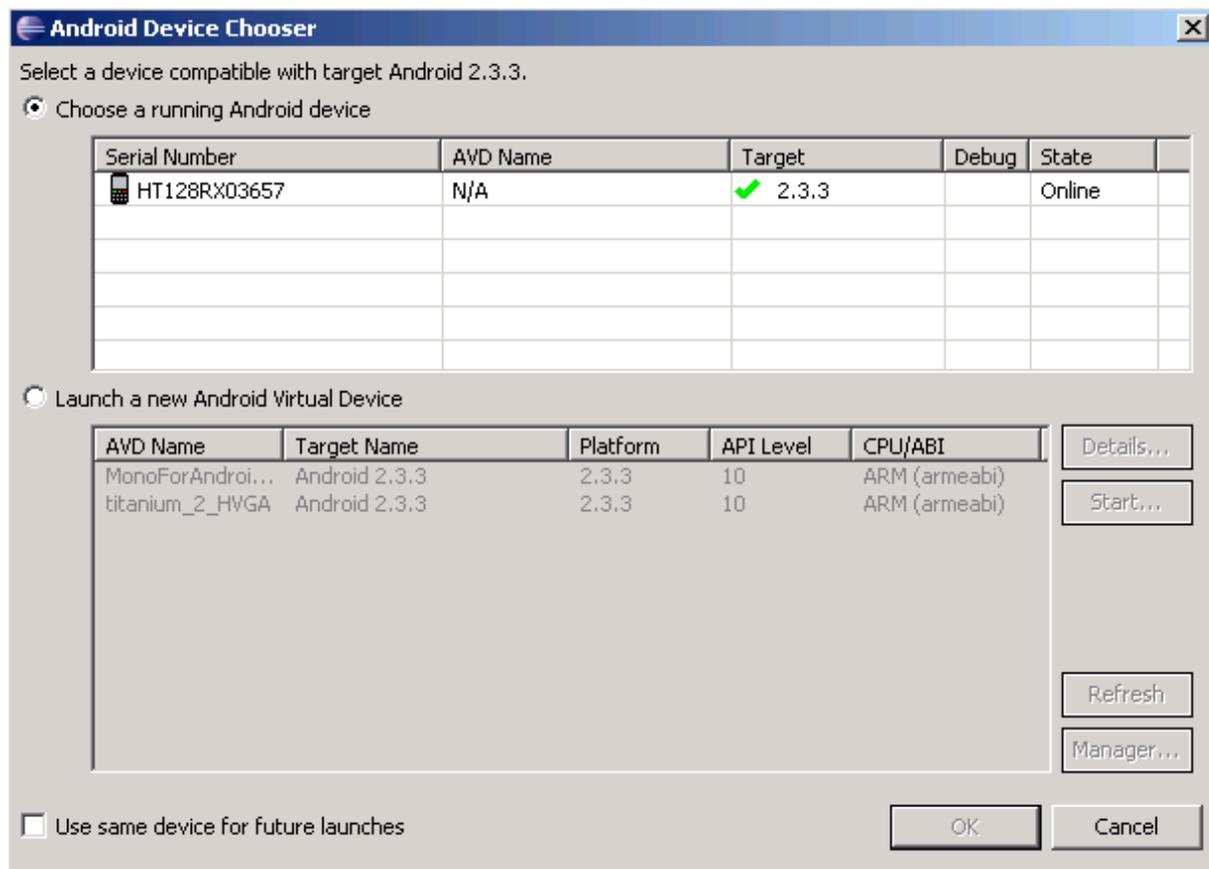


Abbildung 5: Android Device Chooser

Eclipse arbeitet noch einen Moment, dann startet die „Hallo Welt“-App direkt auf dem Smartphone. Sie sieht genau gleich aus wie auf dem Simulator.

### 2.3.3. AudioRecording

Die erste wichtige Funktionalität, die mit PhoneGap untersucht werden soll, ist die Aufnahme von Sprache. Die PhoneGap-API bietet hierfür ein Media-Objekt mit den entsprechenden Methoden und einem umfassenden Beispiel. Für das Starten einer Aufnahme genügen zwei Zeilen JavaScript-Code:

```
record_media = new Media(src, onSuccess, onError);
record_media.startRecord();
```

Beendet wird die Aufnahme damit:

```
record_media.stopRecord();
```

Im ersten Versuch wird die Aufnahme direkt beim Aufruf der App gestartet (anstatt dem „Hallo Welt“-Alert). Gleichzeitig wird ein Timer gesetzt, der nach zehn Sekunden

die Aufnahme automatisch wieder beendet. Hier der vollständige Beispielcode (ähnlich dem Standard-Beispiel der PhoneGap-Dokumentation):

```
function recordAudio() {
    var media = new Media("myrecord.mp3", onSuccess, onError);
    media.startRecord();

    var recTime = 0;
    var recInterval = setInterval(function() {
        recTime = recTime + 1;
        if (recTime >= 10) {
            clearInterval(recInterval);
            media.stopRecord();
        }
    }, 1000);
}

function onSuccess() {
    alert("audio success");
}

function onError(error) {
    alert(error.message);
}
```

Die App startet beim Testen erneut ohne Probleme und gibt nach zehn Sekunden wie erwartet die Meldung „audio success“ aus. Das Ergebnis kann mit einer beliebigen Browser-App überprüft werden: im Wurzelverzeichnis der SD-Karte befindet sich eine Datei „myrecord.mp3“, die beim Abspielen exakt wiedergibt, was während der Testaufnahme gesprochen wurde. Erstes Fazit: Erfolg!

### 2.3.4. Arbeiten mit dem Dateisystem

Die oben beschriebene Beispielanwendung soll nun um die nächste Funktion erweitert werden: Die erzeugte mp3-Datei soll nach Beenden der Aufnahme in ein anderes Verzeichnis verschoben und umbenannt werden.

PhoneGap hält für diese Aufgabe eine eigene File-API bereit, die sich jedoch nicht so unkompliziert nutzen lässt wie das Media-Objekt. Zunächst muss sich der Entwickler an das grundlegende Konzept der asynchronen Aufrufe gewöhnen. Da gibt es beispielsweise ein Objekt namens FileEntry, das eine Datei im Dateisystem repräsentiert. Dieses Objekt besitzt eine Methode getParent, welche ein DirectoryEntry zurückgeben soll. Das DirectoryEntry-Objekt entspricht dem Ordner, in dem sich die Datei befindet.

Der Aufruf:

```
var parentDirectory = entry.getParent();
```

führt jedoch nicht zu dem gewünschten Ergebnis, da die Methode keinen Rückgabewert besitzt, sondern das Ergebnis asynchron an eine success-Methode übergeben wird. Das Beispiel muss korrekterweise so lauten:

```
var parentDirectory;
entry.getParent(success, fail);

function success(parentDir) {
    parentDirectory = parentDir;
}

function fail(error) {
    // Fehlerausgabe
}
```

Dieser asynchrone Aufruf hat zur Folge, dass mit dem nachfolgenden Programmablauf erst dann fortgefahren werden kann, wenn ein benötigter Wert mittels success-Methode zurückgegeben wurde. Es entsteht also unter Umständen eine ziemlich lange Verkettung von Funktionen, was den Code so stark zerstückelt, dass er schon wieder unübersichtlich wird.

Nachfolgend wird das Beispiel von oben weitergeführt und durch Kommentare erklärt. Die erste Funktion `saveRecord()` wird direkt nach dem Befehl `media.stopRecord()`; des bereits vorhandenen `AudioRecord`-Beispiels aufgerufen.

```
var temp_fileEntry = null;
var target_directory = null;
var record_filename = 'ownrecordname';

function saveRecord() {
    // Kontrolle, ob temporäre Aufnahme-Datei vorhanden ist.
    // Wenn ja: asynchroner Aufruf der Funktion saveRecordResolved
    window.resolveLocalFileSystemURI(
        "file:///sdcard/myrecord.mp3",
        saveRecordResolved
    );
}
```

```
function saveRecordResolved(fileEntry) {
    // fileEntry entspricht der temporären Datei ("myrecord.mp3")
    // Zwischenspeichern des fileEntries in Variable
    temp_fileEntry = fileEntry;
    // Aufruf des Root-Verzeichnisses und Übergabe an
    // Funktion saveRecordFolder
    window.resolveLocalFileSystemURI(
        "file:///sdcard",
        saveRecordFolder
    );
}

function saveRecordFolder(directoryEntry) {
    // directoryEntry entspricht Root-Verzeichnis des Smartphones
    // Aufruf des Ordners „studipodcast“ bzw. Anlegen des Ordners,
    // falls nicht vorhanden
    // Übergabe an Funktion saveRecordMove
    directoryEntry.getDirectory(
        "studipodcast",
        {create: true, exclusive: false},
        saveRecordMove
    );
}

function saveRecordMove(targetEntry) {
    // targetEntry entspricht dem Verzeichnis "/studipodcast"
    // Zwischenspeichern in Variable
    target_directory = targetEntry;
    // Verschieben der temporären Datei ins Zielverzeichnis
    // und umbenennen
    temp_fileEntry.moveTo(
        targetEntry,
        record_filename+'.mp3',
        saveRecordSuccess,
        saveRecordFail
    );
}

function saveRecordSuccess(entry) {
    // entry entspricht verschobener, umbenannter Datei
    // Erfolgsmeldung ausgeben
    alert("moved successfully");
}

function saveRecordFail(error) {
    // Fehler beim Verschieben des Datei
    // Fehlerausgabe
    alert(error.message);
}
```

Auch dieser Code lässt sich auf dem Smartphone ausführen. Die Tests zeigen, dass die aufgenommene Datei tatsächlich in das gewünschte Verzeichnis verschoben und umbenannt wird. Eine offene Frage für den zweiten Teil der Arbeit ist allerdings,

ob der Code für andere Plattformen angepasst werden müsste, oder ob der Pfad `file:///sdcard` genauso auch unter iOS funktioniert. Das kann an dieser Stelle aus Zeitgründen noch nicht evaluiert werden.

### 2.3.5. Oberflächengestaltung

Jede Anwendung steht und fällt mit ihrer Oberfläche. PhoneGap sieht vor, für die Entwicklung von Oberflächen ein weiteres Framework einzubinden. Prominente Vertreter sind hier Sencha Touch und jQueryMobile. Die Wahl für diese Arbeit fällt aus Gründen der Gewohnheit auf letzteres.

Für einen Einsatz von jQueryMobile werden drei Dateien benötigt: die JavaScript-Dateien von jQuery (hier Version 1.7.1) und jQueryMobile (hier 1.0.1) sowie die dazugehörige CSS-Datei (ebenfalls 1.0.1). Alle drei werden in das Verzeichnis `/assets/www` kopiert und in der `index.html`-Datei zusätzlich zu den anderen Script-Dateien eingebunden:

```
<link rel="stylesheet" href="jquery.mobile-1.0.1.min.css" />
<script type="text/javascript" src="jquery-1.7.1.min.js">
</script>
<script type="text/javascript" src="jquery.mobile-1.0.1.min.js">
</script>
```

Der Aufbau einer App-Oberfläche mit jQueryMobile ist denkbar einfach. Folgender Code (in der `index.html`) genügt für die erste Testseite:

```
<body>
  <div data-role="page" id="home">
    <div data-role="header">
      <h1>StudiPodcast</h1>
    </div>
    <div data-role="content">
      <p>Hallo Welt</p>
    </div>
  </div>
</body>
```

Mittels JavaScript und CSS wird das HTML-Gerüst in eine ansprechende, relativ nativ wirkende Oberfläche gerendert. Als letzter Schritt soll nun ein Button eingefügt und mit der oben entwickelten Aufnahme-Funktionalität verbunden werden.

Im content-Bereich genügt hierfür die Zeile:

```
<a data-role="button" id="btn_startRecord">Aufnahme starten</a>
```

Im JavaScript wird eine entsprechende Funktion in gewohnter jQuery-Manier zum Beispiel über die ID des Buttons an dessen click-Event gebunden:

```
$("#btn_startRecord").click(function() {  
    startRecord();  
});
```

## 2.4. Zwischenbetrachtung

Im theoretischen Grundlagenteil dieser Studienarbeit wurde ausführlich das Thema App-Entwicklung bzw. die verschiedenen Möglichkeiten untersucht. Vor allem die Vor- und Nachteile von Native Apps und Web Apps wurden aufgewogen und mit dem Ansatz von Hybrid Apps versucht, die Stärken beider Modelle zu verbinden. Dabei sind die beiden Hauptziele, plattformunabhängig zu sein und gleichzeitig auf möglichst viele native Funktionen der einzelnen Plattformen zugreifen zu können.

Anschließend wurde mit PhoneGap ein Framework praktisch getestet, das genau dieses Prinzip umzusetzen versucht. Bisher konnten damit auch alle Aufgabenstellungen mit relativ wenig Aufwand bewältigt werden. Einzige Schwachstelle ist, dass die Oberfläche nicht komplett wie eine Native App aussieht und die Performance vor allem bei den Übergängen einzelner Bildschirme (Transitions) zu wünschen übrig lässt. Das kann allerdings auch an jQueryMobile liegen und muss im nachfolgenden Teil der Arbeit eventuell noch eingehender untersucht werden, bevor PhoneGap als Framework für die zu entwickelnde StudiPodcast-App eingesetzt werden kann.

## 2.5. App-Entwicklung mit Titanium

Für einen direkten Vergleich mit PhoneGap soll das Titanium Mobile SDK von Appcelerator herangezogen werden. Wie bereits in Kapitel 2.2.3 beschrieben, verfolgt es einen ähnlichen Ansatz wie PhoneGap. Die generierte WebApp wird allerdings im Hintergrund ausgeführt und durch eine native Benutzeroberfläche erweitert, wodurch der Eindruck einer kompletten Native App entsteht. Die ersten Versuche zeigen direkt, dass das bei PhoneGap beklagte Ruckeln der Oberfläche hier kein

Problem darstellt. Anschließend wird untersucht, ob auch die restlichen benötigten Funktionalitäten wie Audioaufnahme, Dateimanagement und Kommunikation mit einem Server mittels Titanium realisierbar sind.

### 2.5.1. Entwicklungsumgebung und Projekterstellung

Appcelerator bietet eine eigene Entwicklungsumgebung für die Arbeit mit dem Titanium Mobile SDK zum Download an. Nach kostenloser Registrierung kann Titanium Studio von der Website des Unternehmens heruntergeladen werden. Die Oberfläche ist stark im Stil von Eclipse gehalten.

Nach der Auswahl von File > New > Titanium Mobile Project öffnet sich ein Fenster zur Konfiguration des gewünschten Projekts (siehe Abbildung 6).

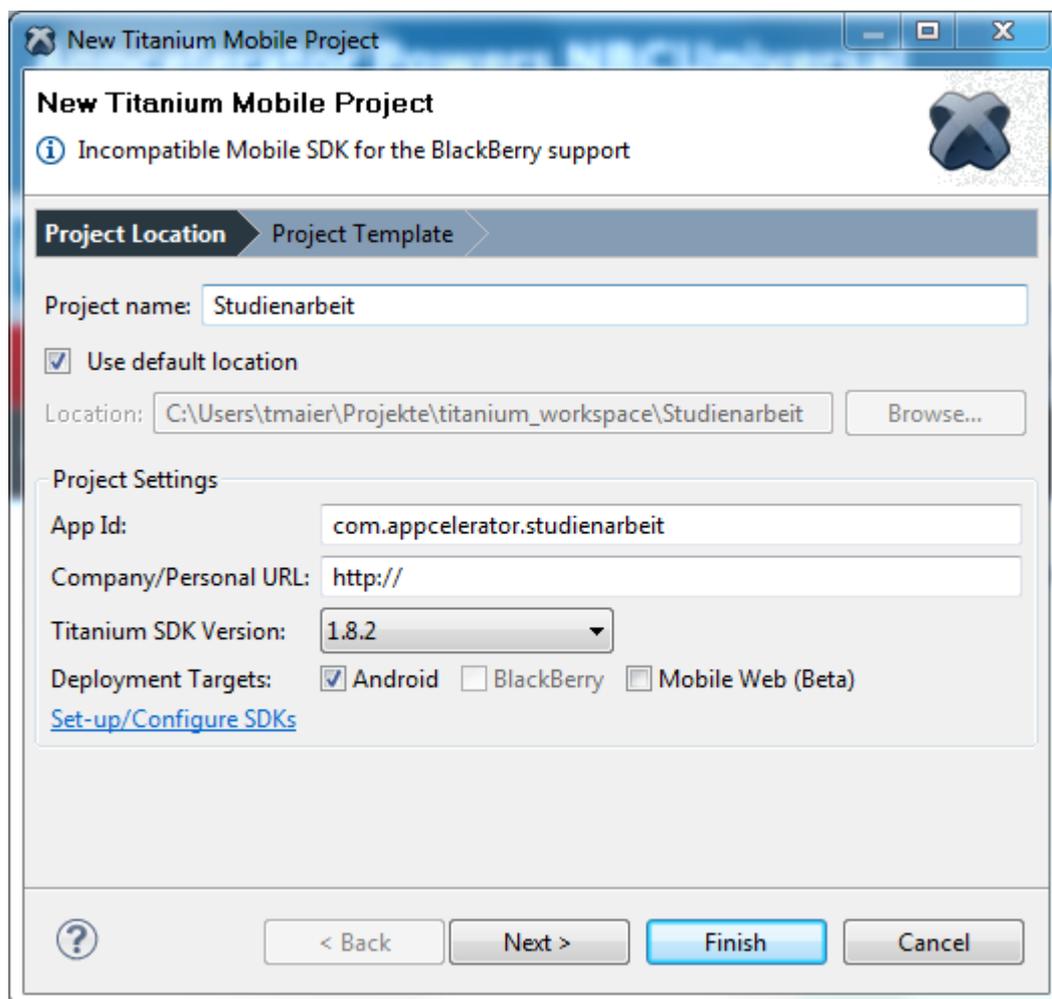
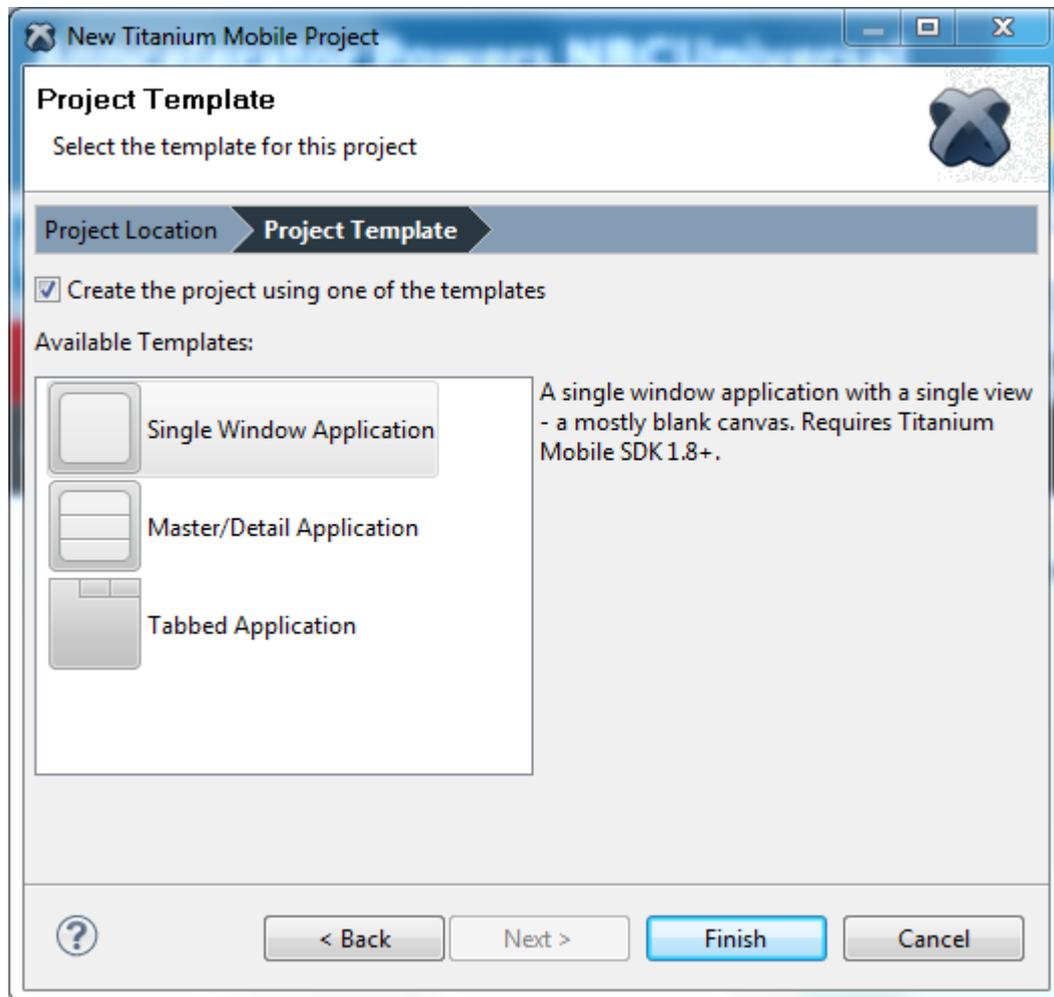


Abbildung 6: Anlegen eines neuen Projekts in Titanium Studio (1)

Nach Ausfüllen der Pflichtfelder (Project name und App Id) und Wahl der zu unterstützenden Plattformen (in diesem Beispiel nur Android) kann mit dem Button Next zum nächsten Fenster gesprungen werden (siehe Abbildung 7).



**Abbildung 7: Anlegen eines neuen Projekts in Titanium Studio (2)**

In diesem Auswahlbereich bietet der Editor die Möglichkeit, eine Vorlage für das neue Projekt auszuwählen. Für den Moment soll es eine Single Window Application sein. Durch Betätigen des Finish-Buttons wird das Projekt angelegt.

Die nachfolgende Abbildung 8 zeigt Titanium Studio direkt nachdem das Projekt angelegt wurde. Die einzelnen Eigenschaften wie die App Id oder die Deployment Targets können in der Datei tiapp.xml auch nachträglich noch geändert werden.

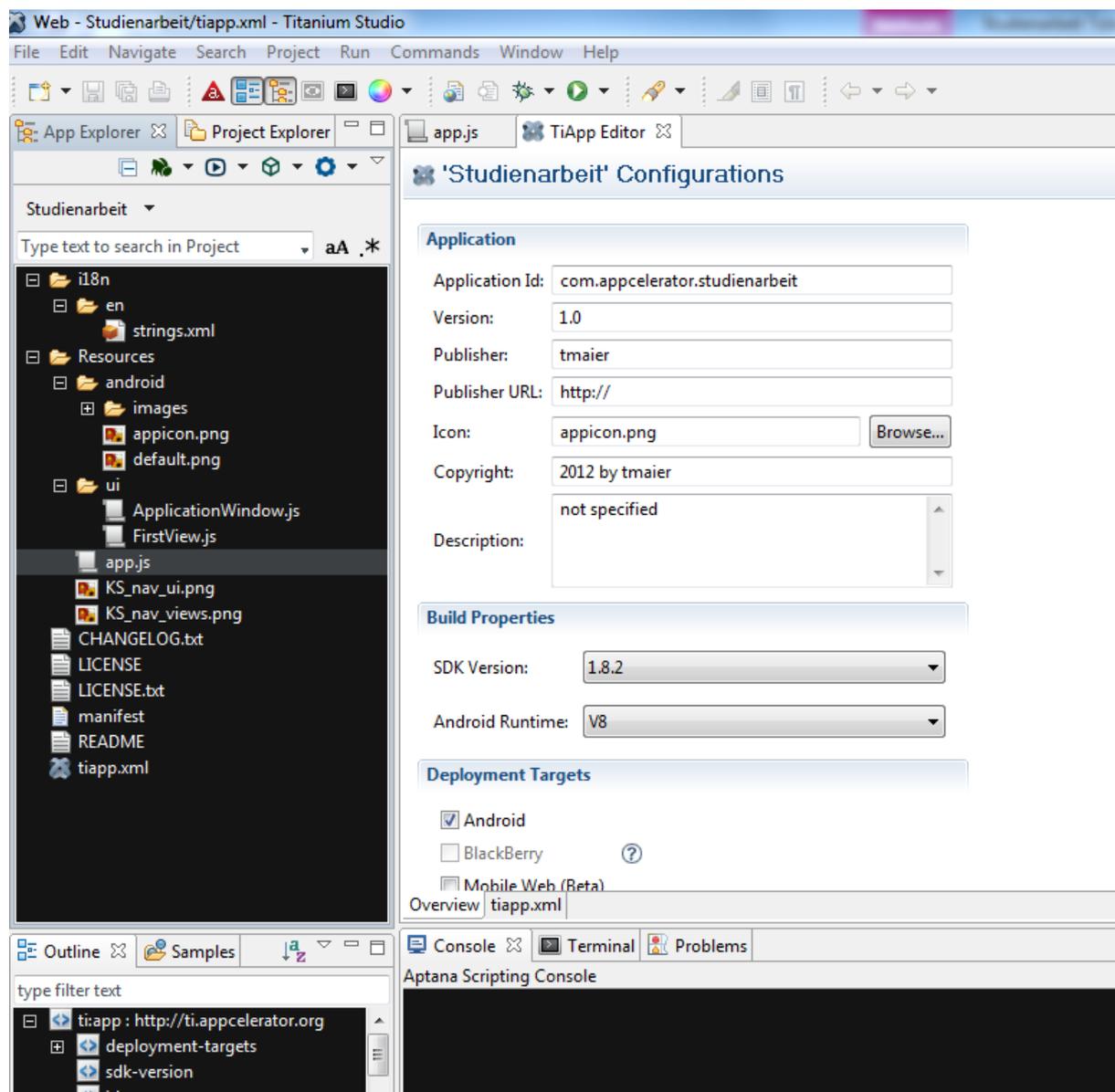


Abbildung 8: Projektstruktur in Titanium Studio

Das Projekt besteht aus zwei Hauptordnern: i18n und Resources. Der Ordner i18n (eine in der Softwareentwicklung übliche Abkürzung für internationalisation) enthält die Datei strings.xml, welche für die Übersetzung von Strings in verschiedene Sprachen vorgesehen ist. Im Ordner Resources befinden sich die Unterverzeichnisse android, welches alle Grafiken enthält, die für Android-Smartphones und -Tablets als Appicon benötigt werden, sowie ui, in dem sich die JavaScript-Dateien für die Oberfläche befinden.

Die Datei app.js ist so etwas wie der Ausgangspunkt für die App. Sie wird nach dem Öffnen der App als Erstes ausgeführt. Der nachfolgende Code entstammt der Standard-app.js aus dem oben erstellten Beispiel.

```
if (Ti.version < 1.8 ) {
    alert('Sorry - this application template requires
          Titanium Mobile SDK 1.8 or later');
}
else if (Ti.Platform.osname === 'mobileweb') {
    alert('Mobile web is not yet supported by this template');
}
else {
    //require and open top level UI component
    var ApplicationWindow = require('ui/ApplicationWindow');
    new ApplicationWindow().open();
}
```

In diesem Codeabschnitt wird zuerst überprüft, dass es sich um eine passende Version des Titanium SDKs sowie eine unterstützte Plattform handelt. Anschließend wird eine Instanz des `ApplicationWindow` erzeugt und „geöffnet“. Dieses `ApplicationWindow` entstammt wiederum der Datei `ApplicationWindow.js` aus dem `ui`-Ordner.

Der richtige Ort, um erste Änderungen vorzunehmen, ist eben diese Datei. An dieser Stelle werden die einzelnen „Hello World“-Versuche und das grundsätzliche Arbeiten jedoch nicht näher erläutert, sondern großzügig übersprungen. Auffällig im Vergleich zu PhoneGap ist allerdings, dass komplett in JavaScript geschrieben wird, ohne HTML und CSS.

## 2.5.2. Dateisystem und Netzwerk

Titanium bietet zwei sehr interessante Klassen: `Filesystem` und `Network`. Mit `Filesystem` lassen sich alle Anforderungen an das Dateisystem abdecken, wie etwa das Erstellen oder Öffnen von Dateien. Das Beispiel aus Kapitel 2.3.4 soll hier noch einmal mit Titanium umgesetzt werden. Dabei geht es darum, eine eben aufgenommene Datei `myrecord.mp3` in ein anderes Verzeichnis zu verschieben und umzubenennen.

```
function move () {
    // Kontrolle, ob temporäre Aufnahme-Datei vorhanden ist.
    var file = Ti.Filesystem.getFile("myrecord.mp3");
    if (file.exists()) {
        // Datei verschieben und umbenennen.
        file.move("studipodcast/ownrecordname.mp3");
    }
}
```

Durch den asynchronen Funktionsaufbau von PhoneGap war für diese Aufgabe eine Verkettung von insgesamt sechs Funktionen und zusätzlichen Variablen notwendig. Titanium löst diese Aufgabe mit einer einzigen vergleichsweise sehr kurzen Funktion.

Die Network-Klasse ist im Titanium SDK für die Kommunikation mit einem Server zuständig. In Verbindung mit Filesystem lässt sich folgendes einfaches Beispiel realisieren:

```
function download(url) {
    // HTTP-Request aufbauen
    var client = Ti.Network.createHTTPClient({

        // Diese Funktion wird mit der Antwort des Requests aufgerufen
        onload : function(e) {
            var file = Ti.Filesystem.getFile('downloaded.mp3');
            file.write(this.responseData);
        },

        // Timeout des Requests
        timeout : 5000
    });

    // HTTP-Request absenden
    client.open("GET", url);
    client.send();
}
```

Die oben beschriebene Funktion setzt einen HTTP-Request zu einer übergebenen URL ab. Das Beispiel geht davon aus, dass sich hinter der URL eine mp3-Datei befindet. Der Inhalt dieser Datei (`this.responseData`) wird in eine lokale Datei `downloaded.mp3` gespeichert.

### 2.5.3. AudioRecording

Während der Entstehung dieser Studienarbeit wurden mehrere neue Versionen des Titanium Mobile SDK veröffentlicht (1.8.1 bis aktuell 2.0.1). Das Framework enthält in allen Versionen eine Klasse `Media` bzw. `Media.AudioRecorder`. Allerdings funktioniert dieses Feature nur unter iOS (laut Dokumentation iPhone und iOS [3]). Praktische Versuche haben das Nicht-Funktionieren unter Android bestätigt. Im Internet kursieren verschiedene (kostenpflichtige wie kostenlose) Module, um eben diese Lücke zu schließen, allerdings sind sie alle nicht kompatibel mit den oben genannten Versionen des Frameworks.

## 2.6. Technologieauswahl

Da die Aufgabenstellung dieser Arbeit eine Cross-Platform-Lösung verlangt und eine reine Web App aufgrund der benötigten nativen Features wie AudioRecording nicht in Frage kommt, bleibt nur der hybride Ansatz übrig. Die derzeit größten Vertreter in dieser Kategorie – PhoneGap und Titanium – wurden hinsichtlich ihrer Eignung für die anstehende Entwicklungsaufgabe getestet und verglichen.

In der Kategorie Benutzerfreundlichkeit gewinnt Titanium klar gegen PhoneGap. Es wurden zwar im Rahmen dieser Studienarbeit keine messbaren Werte erhoben, jedoch an verschiedenen Stellen auf die teilweise sehr unsauberen Transitions in PhoneGap-Apps hingewiesen. Die von Titanium erzeugten Oberflächen arbeiten hier deutlich zuverlässiger und wirken im direkten Vergleich auch wesentlich nativer.

Der Entwicklungsaufwand mit dem Titanium Mobile SDK ist geringer als mit PhoneGap. Ein Hauptgrund hierfür sind die asynchronen Aufrufe in PhoneGap, die sehr aufwändig wirken und den Code unnötig unübersichtlich machen. Das oben angeführte Beispiel des Dateiverschiebens belegt dies (Kapitel 2.3.4 und 2.5.2).

Ein absolutes Ausschlusskriterium für Titanium ist jedoch die Tatsache, dass unter Android damit keine Audioaufnahme möglich ist. Damit fällt die Wahl der Technologie auf PhoneGap.

### 3. Mobiles Lernen

Dieser Abschnitt befasst sich mit dem Konzept StudiPodcasts bzw. StudiCast. Die technische Realisierung ist im darauffolgenden Abschnitt beschrieben.

#### 3.1. StudiCast

„Lernen ganz nebenbei – beim Autofahren, während der Hausarbeit, beim Sport oder sogar während dem Computerspielen. Das scheitert bei den meisten Schülern und Studenten daran, dass Lernen meistens mit Lesen verbunden ist. Genau hier setzt die neue App für Mobiles Lernen an. Es soll ein Internet- und App-gestütztes Netzwerk entstehen, in dem „studentische Podcasts“ ausgetauscht werden können. Lernen funktioniert dann nicht über Lesen, sondern auch übers Hören.

Jeder Nutzer kann selbst Inhalte beitragen, indem er oder sie Zusammenfassungen von Lerntexten, Definitionen von schwierigen Begriffen, oder Übersetzungen bei Fremdsprachen vorliest und per App oder Browser in das Netzwerk einspielt. Im besten Fall geben die Lehrer und Dozenten sogar ihre Einwilligung, dass ihre Vorlesungen direkt mitgeschnitten werden dürfen, oder geben ihre Vorlesungsskripte und Veröffentlichungen entsprechend frei.

Die Benutzung der Plattform soll dem Community-Gedanken folgen. Über ein Punktesystem ist geregelt, dass jeder Einsteiger eine gewisse Menge an Hörproben sofort hören darf. Danach wird erwartet, dass jeder, der konsumiert, auch etwas zum Netzwerk beiträgt und selbst Inhalte einstellt. Je höher diese von den Hörern bewertet werden, desto mehr Punkte bekommt er dafür und kann diese selbst wieder in Podcasts eintauschen.

Die eingestellten Podcasts werden wie bereits beschrieben von den Hörern bewertet. Vom Autor können sie mit Metatags (Schulfach/Studiengang, Thema, Schlagwörter, etc.) versehen werden. Über die App können dann bequem die entsprechenden Beiträge zu einem bestimmten Thema gefunden und überall und zu jeder Zeit gehört und somit unterbewusst gelernt werden.“

Diese kurze Beschreibung umreißt die Idee hinter StudiCast sehr deutlich. Der Text wurde in dieser Form im Winter 2011 als Bewerbung für den Wettbewerb BW goes Mobile der MFG Medien- und Filmgesellschaft Baden-Württemberg eingereicht.

Das Konzept hat sich anschließend während der Umsetzungsphase weiterentwickelt und einige Veränderungen erfahren. So wurde beispielsweise aus dem ursprünglichen Arbeitstitel StudiPodcasts der vorläufige Produktname StudiCast.

### 3.2. Funktionsumfang

Aus dem oben beschriebenen Konzept leiten sich etliche Funktionen und Features ab, die nachfolgend kurz zusammengefasst und klassifiziert werden. Im Rahmen der Studienarbeit lassen sich nicht alle Ideen umsetzen, deshalb liegt der Fokus zunächst auf einer stabilen Grundfunktionalität.

Zu den absoluten Basisfunktionen der App gehört die Aufnahme und Wiedergabe von Podcasts. Damit verbunden ist eine ansprechende, gut strukturierte Oberfläche, die geeignet ist aufgenommene Podcasts mit Metatags zu versehen und später wieder zu finden.

Der erweiterte Funktionsumfang besteht aus der Kommunikation mit einem Server. Die Server-Seite muss über eine Benutzerverwaltung sowie eine Dateiverwaltung und entsprechende Webschnittstellen verfügen. Die App muss über diese Schnittstellen einen Benutzer registrieren, sich als Benutzer authentifizieren sowie Podcasts inklusive Metainformationen hoch- und herunterladen können.

Diese Funktionen werden im Rahmen der Studienarbeit umgesetzt. Darüber hinaus wird die App so entworfen, dass weitere Features wie die Kommentierung und Bewertung von Podcasts durch eine Community oder das Benutzer-Punkte-System in einer späteren Version ergänzt werden können.

### 3.3. Datenhaltung

Eine wichtige Überlegung für jedes App-Projekt ist die Art der Datenhaltung. Dabei spielen zwei Fragen eine Rolle: Speicherung in einer Datenbank oder im Dateisystem? Und: Speicherung auf einem zentralen Server oder auf jedem einzelnen Endgerät?

Für die Beantwortung der ersten Frage muss zunächst klar sein, welche Arten von Daten gespeichert werden sollen. Das sind bei StudiCast natürlich hauptsächlich Au-

diodaten, aber auch die dazugehörigen Metainformationen (Titel, Beschreibung) und Benutzerinformationen (Name, Email-Adresse).

Für diese Kombination eignen sich eigenen Erfahrungen und den verschiedensten Diskussionen zu diesem Thema zufolge am ehesten eine Mischform: Die Audiodaten werden als mp3-Dateien im Dateisystem abgelegt (sowohl auf der Server- als auch der Clientseite) und die Metatags in einer Datenbank gespeichert (MySQL auf der Serverseite und SQLite auf der Clientseite).

Die Frage nach dem physikalischen Speicherort ist ebenfalls schnell geklärt: Da das Lernen mit StudiCast überall möglich sein soll – auch beim Joggen im Wald und beim Fahren mit der Bahn – müssen die Podcasts auch überall zur Verfügung stehen, unabhängig vom mobilen Internet. Es muss also eine lokale Speicherung der Daten erfolgen.

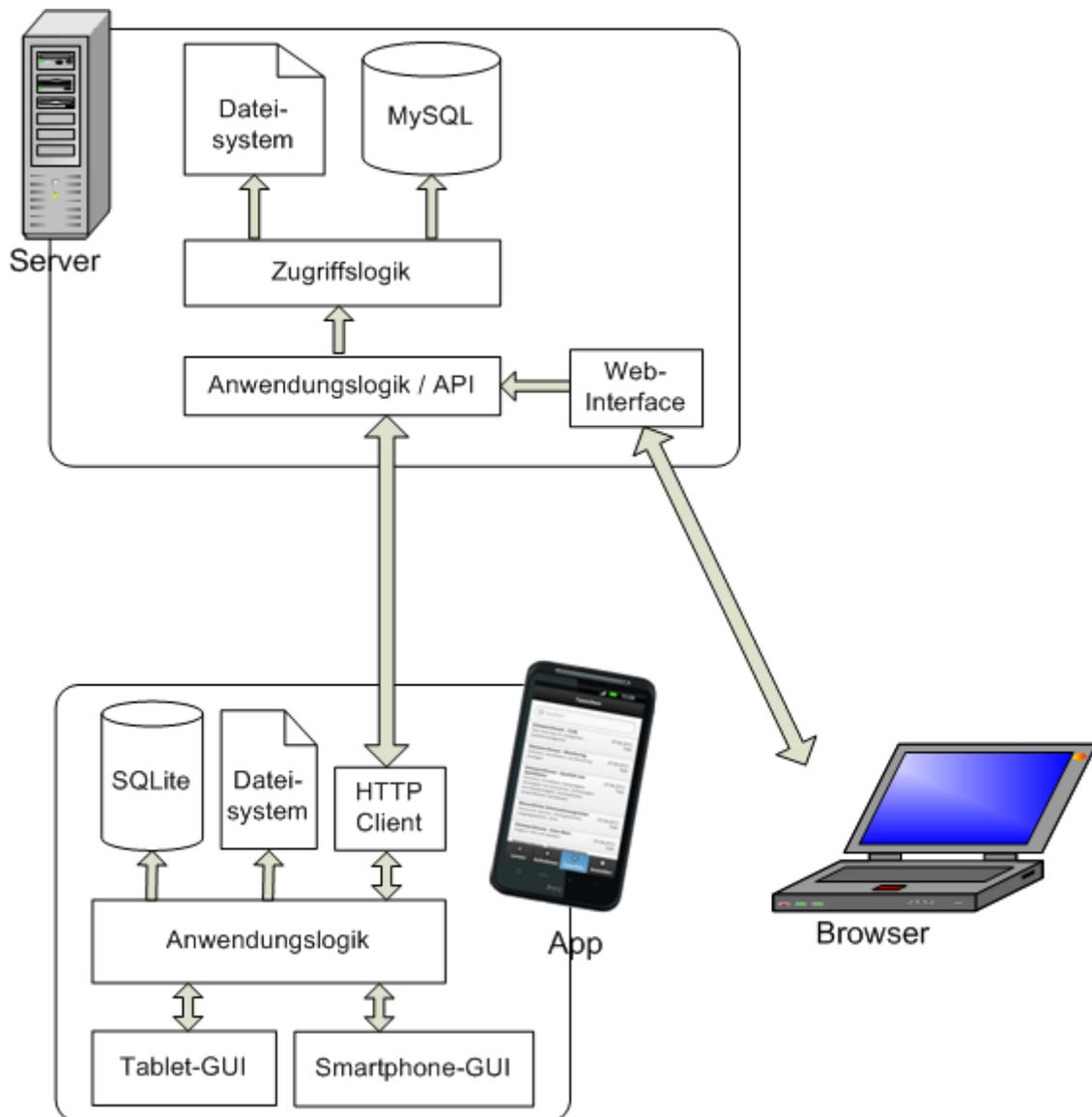
Um zu überprüfen, mit welchen Datenmengen zu rechnen ist, werden verschiedene Probeaufnahmen mit dem ersten Prototypen der App getätigt. Die folgende Tabelle 1 zeigt die Messergebnisse.

	<b>Dauer der Aufnahme</b>	<b>Größe der mp3-Datei</b>
<b>1</b>	2:49 min	269 KB
<b>2</b>	1:36 min	154 KB
<b>3</b>	2:11 min	210 KB
<b>4</b>	0:49 min	81 KB
<b>5</b>	2:27 min	234 KB
<b>☉</b>	<b>~1:58 min</b>	<b>~190KB</b>

**Tabelle 1: Probemessungen Aufnahmedauer und Dateigröße**

Eine durchschnittliche Aufnahmedauer von knapp zwei Minuten erscheint realistisch. Bei der dazugehörigen durchschnittlichen Dateigröße von ca. 190 KB würden die Audiodateien auf einem Endgerät bei angenommenen 100 Aufnahmen rund 19 MB Speicherplatz belegen. Hinzu kommen die Daten in der Datenbank, die jedoch im Vergleich vernachlässigbar klein sind. Ein Speicherbedarf von rund 20 MB sollte für die App also genügen und ist in Zeiten von Gigabyte-SD-Karten sicherlich kein Problem.

## 4. Technische Umsetzung



**Abbildung 9: Struktur der einzelnen Komponenten von StudiCast**

Abbildung 9 zeigt die verschiedenen Komponenten von StudiCast und ihre Zusammenhänge in der angedachten Endausbaustufe. Nachfolgend werden sie ausführlich vorgestellt.

## 4.1. Server

Der Server kann ein beliebiger Webserver mit PHP und SQL-Datenbank sein. Für die hier vorgestellte Umsetzung kommt ein Apache 2.0 mit PHP 5.2.12 und MySQL 5.1 zum Einsatz. Schematisch besteht die Serverseite aus fünf Teilkomponenten.

### 4.1.1. Dateisystem

Mit Dateisystem ist nicht das komplette Verzeichnis des Webserver gemeint, sondern lediglich ein einzelner Ordner „data“. In diesem Ordner werden alle Podcasts in Form von mp3-Dateien gespeichert. Die Dateien sind generisch benannt (bspw. 4fa141f3e5ee4.mp3). Dadurch sind sie von ihrem eigentlichen Titel unabhängig und es entstehen keine Namenskonflikte. Den Zugriff auf die Dateien regelt eine Klasse der Zugriffslogik mit Hilfe der Metadaten (inklusive Dateipfad) aus der Datenbank (siehe Kapitel 4.1.3).

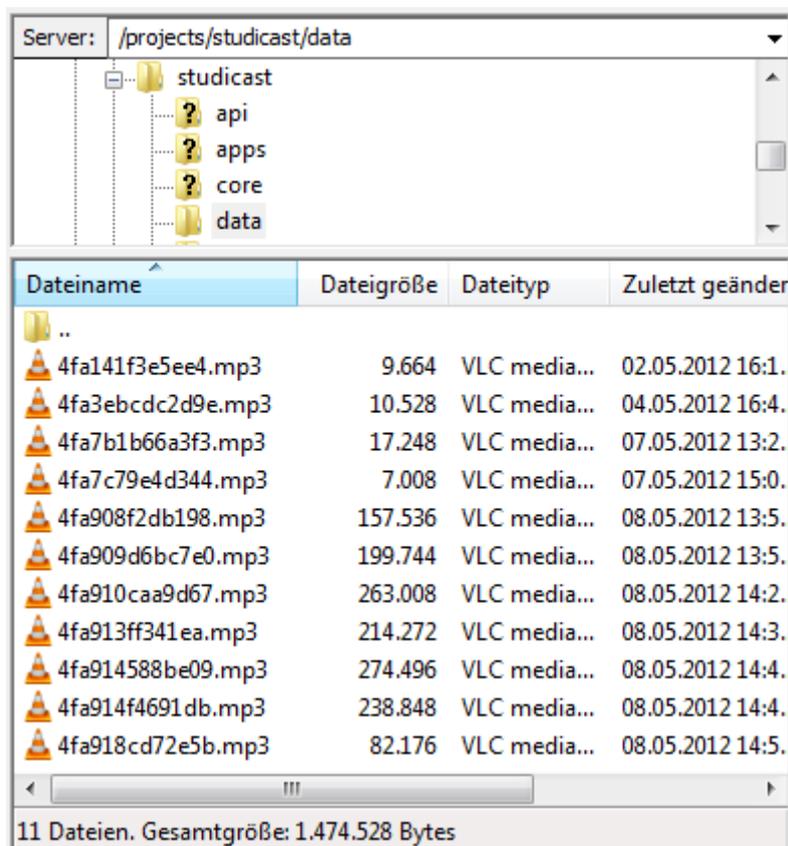


Abbildung 10: Das data-Verzeichnis des Servers

In der ersten Ausbaustufe von StudiCast werden die mp3-Dateien alle direkt in das Verzeichnis data gespeichert (siehe Abbildung 10), da während der Aufbau- und

Testphase nicht mit riesigen Datenmengen zu rechnen ist. Die Anzahl der Dateien und deren Größe sollte allerdings beobachtet werden. In einem späteren Update muss dieser Prozess dann automatisiert werden.

Denkbar ist folgender Ansatz: Ein Cronjob überprüft automatisiert, wie viele Dateien sich direkt in data befinden. Sind es mehr als ein vordefinierter Grenzwert, bspw. 1000 Dateien, wird ein Unterverzeichnis erstellt und die 1000 Dateien dorthin verschoben, inklusive Anpassung der Pfade in der Datenbank. Dadurch bleibt das Dateisystem einigermaßen übersichtlich. Außerdem lassen sich – unter der Annahme, dass es innerhalb der erstellten Unterverzeichnisse keine Änderungen mehr gibt – einfachere Backups durchführen.

#### 4.1.2. MySQL

Die Datenbank besteht in der Basisversion aus drei Relationen: podcast, user und user\_login (siehe Abbildung 11).

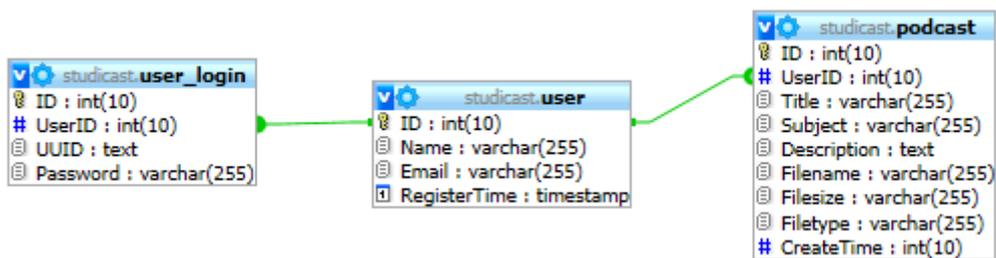


Abbildung 11: Einfaches Datenbankmodell des Servers

Die Relation user ist relativ selbsterklärend. Hier erhält jeder Benutzer nach Registrierung einen Eintrag mit dem gewählten Namen und seiner Email-Adresse. Die ID wird automatisch vergeben. Für spätere Auswertungen wird außerdem in das Feld RegisterTime der Zeitpunkt der Registrierung gespeichert.

Ebenso automatisch erhält jeder Benutzer einen Eintrag in die Tabelle user\_login. Hier wird neben der wiederum von der Datenbank vergebenen ID und der referenzierten UserID noch ein Wert UUID gespeichert. Dabei handelt es sich um die eindeutige Geräte-ID des Smartphones, über die sich ein Benutzer nach der Registrierung jederzeit identifizieren lässt. Das Feld Password bleibt in diesem Fall leer. Es wird für die später zu implementierende Web-Oberfläche benötigt. Wird ein Account darüber angelegt, existiert keine UUID, dafür benötigt der Benutzer ein Passwort, um

sich per Browser zu authentifizieren. Das Datenbankmodell sieht vor, dass ein Benutzer später mit einem Account mehrere Zugangsmöglichkeiten haben kann, etwa die App auf dem Smartphone, die App auf dem Tablet und den Webzugang.

Die Relation `podcast` erhält für jede hochgeladene Aufnahme einen Eintrag. Dieser besteht aus der automatisch vergebenen ID, einer UserID, welche den Podcast mit seinem Autor verknüpft, den Metainformationen Title (Titel der Aufnahme), Subject (Studiengang oder Schulfach) und Description (Beschreibung) sowie den vom System vergebenen Werten Filename (der Dateiname bzw. Pfad im Dateisystem), Filesize (Dateigröße) und Filetype (Dateityp). Letzterer wird separat gespeichert, da StudiCast eventuell in einer späteren Version verschiedene Dateiformate unterstützen soll. Das Feld `CreateTime` wird nicht automatisch von der Datenbank beschrieben, sondern ebenfalls aus den Metadaten, die zusammen mit der Datei hochgeladen werden. Damit enthält es nicht den Zeitpunkt des Uploads, sondern der Zeitpunkt der Aufnahme.

Eine mögliche Erweiterung dieses Basismodells für eine spätere Ausbaustufe von StudiCast ist in der nachfolgenden Abbildung 12 dargestellt.

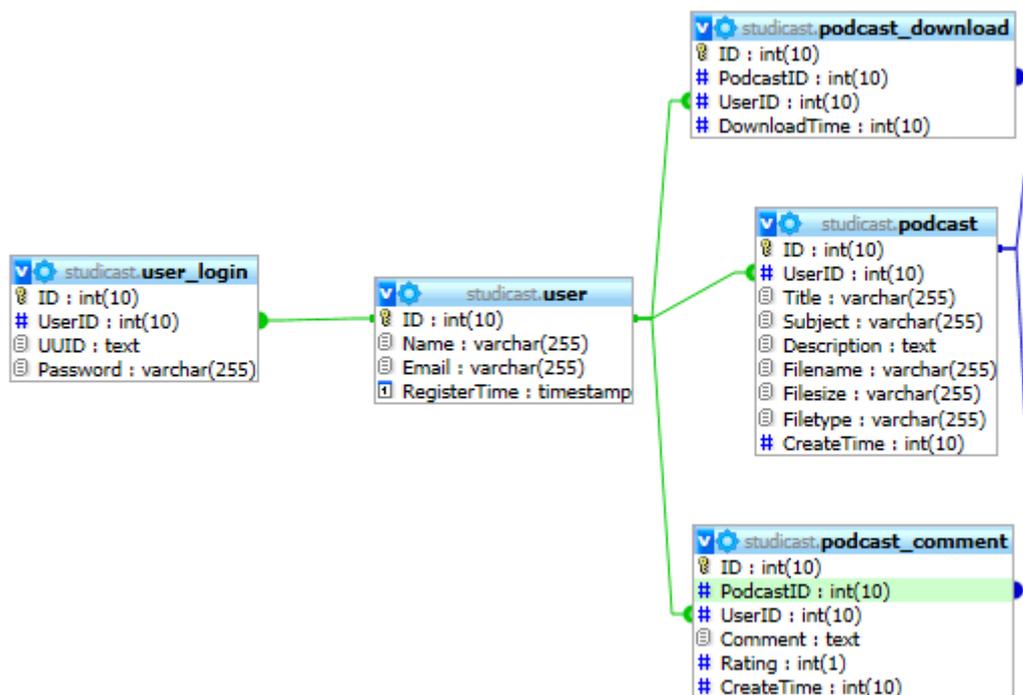


Abbildung 12: Erweitertes Datenbankmodell des Servers

Ergänzt wird das Modell hierbei um die beiden Relationen `podcast_download` und `podcast_comment`. Die Tabelle `podcast_download` erhält einen Eintrag mit jedem

Download einer Aufnahme und speichert dabei die ID von Podcast und Benutzer sowie den Zeitpunkt des Downloads. Damit lassen sich in einer späteren Version nicht nur verschiedene Auswertungen und Aktivitätsstatistiken generieren, sondern auch das im Konzept erwähnte Benutzer-Punkte-System (vgl. Kapitel 3.1) realisieren.

Die Relation `podcast_comment` enthält Kommentare und Bewertungen zu den einzelnen Aufnahmen. Hier sind ein Textfeld für Kommentare sowie ein einstelliger Zahlenwert für die Bewertung vorgesehen.

### 4.1.3. Zugriffslogik

Die Zugriffslogik ist eine Schicht, welche die Anwendungslogik von der Datenhaltung abstrahiert. Sie baut eine Verbindung zur MySQL-Datenbank auf und stellt die drei Controller-Klassen `Util`, `User` und `Podcasts` mit entsprechenden Methoden bzw. Schnittstellen zur Verfügung, die von der Anwendungsschicht genutzt werden können.

Die Klasse `Util` enthält nützliche Hilfsfunktionen, z.B. zum Umwandeln von Kodierungen. `Podcasts` kümmert sich um das Anlegen und Auslesen von Podcasts aus der Datenbank und das Zuordnen von Dateien aus dem `data`-Verzeichnis. Die Klasse `User` enthält alle relevanten Methoden zur Benutzerverwaltung.

### 4.1.4. Anwendungslogik / API

Die Anwendungsschicht stellt gleichzeitig die API dar, das heißt die Schnittstellen, die über das Internet für die Apps und das Web-Interface erreichbar sind. Abbildung 13 zeigt die API. Der Pfeil auf der linken Seite verbindet die Anwendungslogik mit der Zugriffsschicht bzw. dem Dateisystem und der Datenbank.

Die Basis der Anwendungsschicht bildet eine Datei `save.php`. Sie ist für die Authentifizierung von Benutzern zuständig und wird von den vier Schnittstellen `Register`, `Podcasts`, `Users` und `Upload` eingebunden. Die einzelnen Schnittstellen werden nachfolgend kurz vorgestellt.

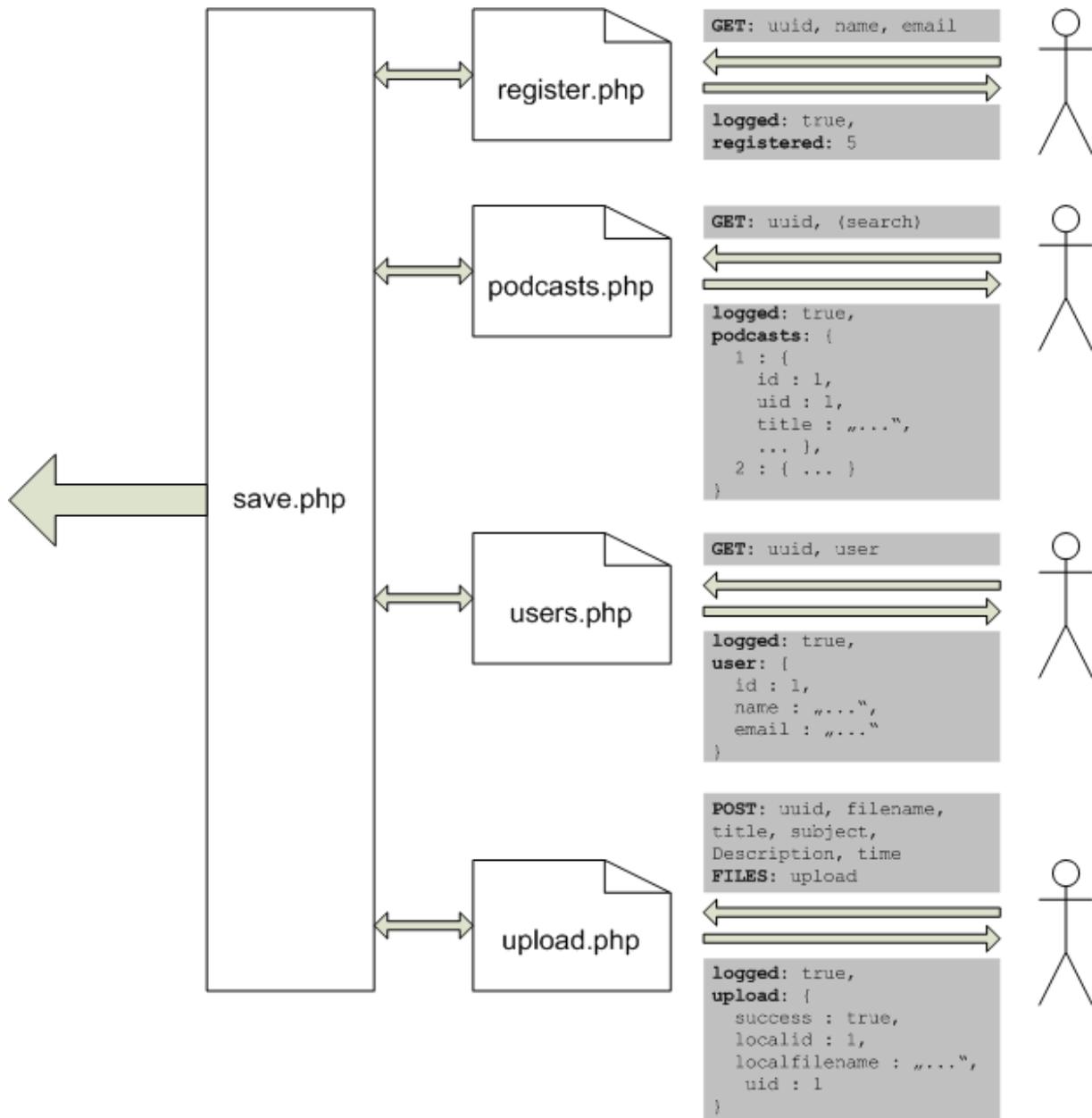


Abbildung 13: StudiCast-API

Register ist für die Registrierung neuer Benutzer zuständig. Sie erwartet die GET-Parameter uuid, name und email und gibt nach erfolgreicher Registrierung ein JSON-Objekt mit den Attributen logged und registered zurück. Das Attribut logged wird von jeder Schnittstelle mit zurückgegeben. Enthält es den booleschen Wert true, konnte der Benutzer authentifiziert und die gewünschte Anfrage ausgeführt werden. Ist logged auf false gesetzt, weiß die App oder das Web-Interface, dass die HTTP-Antwort empfängt, dass die Anmeldung fehlgeschlagen ist. Das Attribut registered enthält die neu generierte einmalige UserID des registrierten Benutzers.

Die Schnittstelle Podcasts erwartet als Parameter ebenfalls die uuid zur Authentifizierung des Benutzers sowie optional einen String search. Die JSON-Antwort enthält das Attribut podcasts, in welchem die ausgewählten Podcasts als Objekte mit allen Metadaten aufgelistet werden. Die Anzahl der zurückgegebenen Podcasts ist zunächst auf 20 begrenzt. Wird der Parameter search mit übergeben, werden nur Podcasts zurückgegeben, die den Suchbegriff search im Titel oder ihrer Beschreibung enthalten.

Diese Schnittstelle genügt in ihrem Umfang für den Prototyp. Für die nächste Ausbaustufe von StudiCast müssen hier jedoch noch etliche Erweiterungen wie eine feinere Suche oder die Einschränkung nach Studiengängen vorgenommen werden.

Die Users-Schnittstelle wird mit einer uuid und der ID eines anderen Benutzers aufgerufen und gibt ein JSON-Objekt user zurück, das die ID, den Namen und die Email-Adresse des entsprechenden Benutzers zurückgibt.

Upload nimmt Dateien entgegen. Neben dem FILES-Parameter werden die POST-Parameter uuid, filename, title, subject, description und time erwartet. Die daraus gewonnenen Metadaten werden gemeinsam mit dem Pfad zu der hochgeladenen Datei in der Datenbank abgelegt. Der Dateityp ist im Prototyp auf mp3 beschränkt. Die Upload-Schnittstelle gibt das JSON-Objekt upload mit dem Pflichtattribut success zurück. Bei erfolgreichem Upload ist success true. Zum Abgleich mit der lokalen Datei enthält das Objekt außerdem die Attribute localid, localfilename sowie uid. Die uid ist die einmalige ID der Datei auf dem Server. Sie dient dem Mapping der lokal gespeicherten Dateien mit den online verfügbaren Podcasts.

#### **4.1.5. Web-Interface**

Die Komponente Web-Interface ist in der im Rahmen der Studienarbeit umgesetzten Version von StudiCast noch nicht enthalten. Gemeint ist damit ein Web-Portal, auf das per Browser zugegriffen werden kann. Es soll ähnliche Funktionen enthalten wie die StudiCast-App und greift hierfür auf dieselben Schnittstellen zu.

## 4.2. App

### 4.2.1. Dateisystem

Auf dem Smartphone oder Tablet werden die Audiodateien ebenso wie auf dem Server in einem zentralen Verzeichnis studicast auf der SD-Karte des Gerätes gespeichert. Aufgrund der vergleichsweise geringen zu erwartenden Anzahl von Dateien ist es hier nicht notwendig, weitere Unterverzeichnisse anzulegen.

### 4.2.2. SQLite

SQLite ist eine leichtgewichtige SQL-Datenbank, die in den meisten mobilen Betriebssystemen und in einigen Webbrowsern implementiert ist [6]. Abbildung 14 zeigt das Datenbankmodell, das für StudiCast auf dem Smartphone nötig ist.



Abbildung 14: SQLite-Datenbankmodell auf dem Smartphone

Die Tabelle podcasts enthält einen Eintrag für jede Aufnahme mit den entsprechenden Metatags. Für eigene Aufnahmen wird in das Feld user der Wert 0 eingetragen. Das Feld status enthält Werte wie „downloaded“, „downloading“ oder „uploaded“. Die uid entspricht der einmaligen UserID des Autors.

In der Relation users werden Benutzer eingetragen. Die ID 0 ist für den eigenen Account vorgesehen. Die systemweit einmalige UserID wird in das Feld uid geschrieben. Es werden nur diejenigen Benutzer lokal gespeichert, von denen auch Podcasts auf dem Gerät vorhanden sind.

### 4.2.3. HTTPClient

Der HTTPClient ist diejenige Komponente der App, die sich um die Kommunikation mit dem Server kümmert. Sie empfängt Anweisungen aus der Anwendungsschicht, sendet HTTP-Requests an den Server, wertet die Antworten aus und leitet sie an die Anwendungsschicht weiter.

### 4.2.4. Anwendungslogik

Die Anwendungslogik ist der Kern der App. Sie steuert den Zugriff auf die einzelnen Daten im Dateisystem oder der SQLite-Datenbank, kommuniziert mit dem HTTPClient sowie der Oberfläche.

Eine Basisaufgabe der Anwendungsschicht ist der Audibereich: Aufnahme von Sprache, Erstellen, Verschieben und Umbenennen von Dateien, Wiedergabe der Aufnahmen inklusive Pause und Beenden.

Der zweite große Bereich ist die Podcast-Verwaltung: Auslesen der gespeicherten Aufnahmen, Sortierung und Suche von Dateien, Anlegen heruntergeladener Dateien, Bearbeiten und Entfernen von Dateien und Metainformationen.

Die dritte Aufgabe ist die Benutzerverwaltung: Registrierung und Login des Benutzers und Herunterladen der erforderlichen Informationen anderer Benutzer.

### 4.2.5. Smartphone- und Tablet-GUI

Die Bildschirmgröße und Auflösung von Smartphones und Tablets unterscheiden sich grundlegend. Eine gute App muss deshalb für beide Formate optimiert sein. Im Rahmen der Studienarbeit wird zwar nur die Grafische Benutzeroberfläche für das Smartphone umgesetzt, allerdings ist der Code durch die Teilung in Komponenten so modular aufgebaut, dass eine eigene Tablet-GUI jederzeit mit wenig Aufwand nachgerüstet werden kann.

Der Aufbau der Smartphone-Oberfläche ist in Abbildung 15 dargestellt. Die Hauptnavigation befindet sich in Form von vier Reitern am unteren Bildschirmrand. Je nach ausgewählter Aktion werden weitere Buttons oder Reiter am oberen Bildschirmrand eingeblendet.

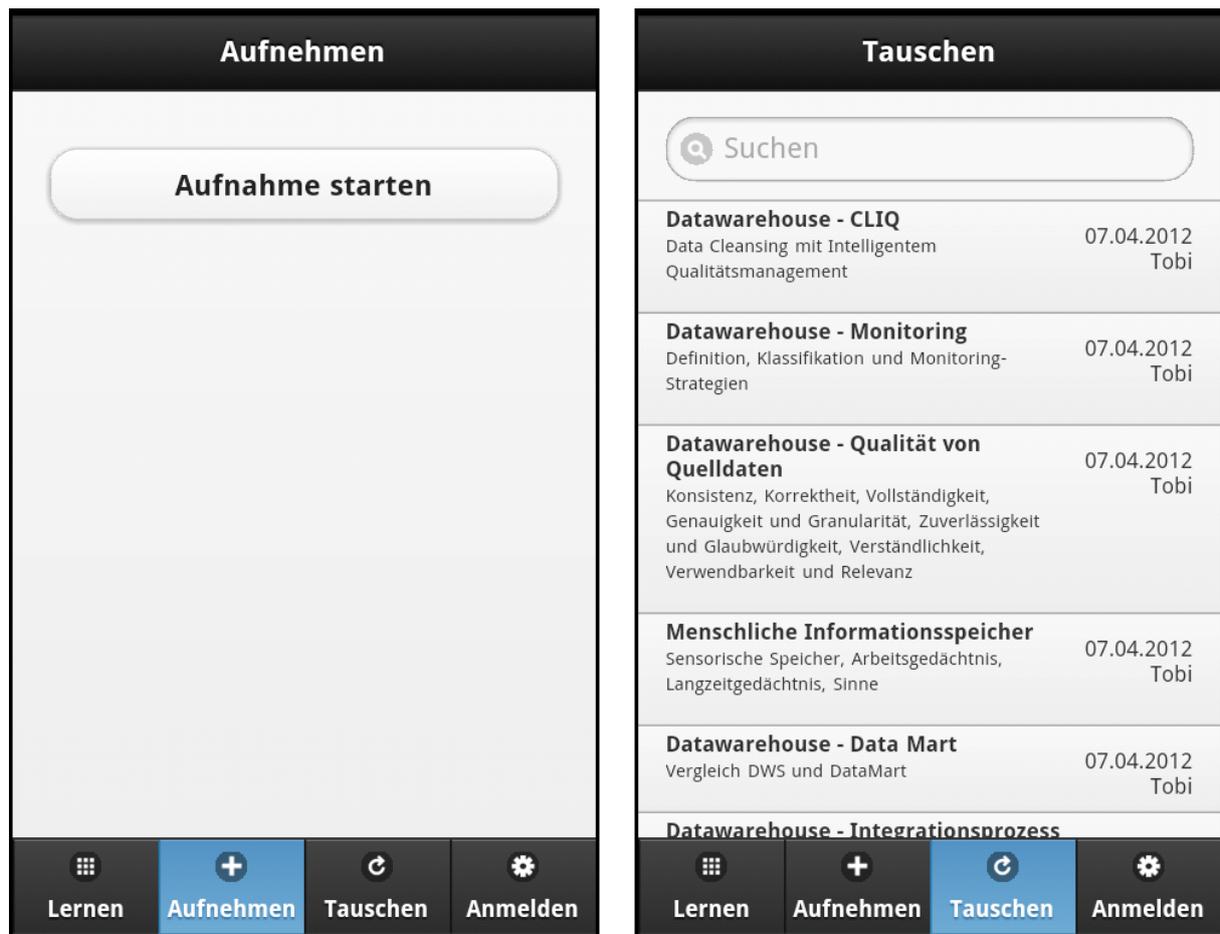


Abbildung 15: Smartphone-Oberfläche von Studicast

Die Oberfläche besteht aus vier Hauptbereichen: Lernen, Aufnehmen, Tauschen, Anmelden. Die ersten beiden Menüs können auch im Offline-Modus genutzt werden. Hinter Aufnehmen verbirgt sich die Möglichkeit, eigene Podcasts aufzunehmen und mit Metatags zu versehen. Die Aufnahmen werden unter dem Punkt Lernen aufgelistet und können von dort direkt gestartet werden.

Für die Tauschen-Funktionalität muss zunächst ein Account angelegt werden. Das ist im Menü Anmelden möglich. Hier können später Optionen zum Einrichten eines weiteren Zugangs untergebracht werden. Die Seite hinter Tauschen ist ähnlich aufgebaut wie hinter Lernen, allerdings werden die angezeigten Podcasts bzw. deren Metadaten direkt vom Server abgerufen. Um eine Aufnahme anhören zu können, muss diese zuerst heruntergeladen werden und steht dann im Lernen-Menü zur Verfügung.

## 5. Fazit

Im Rahmen dieser Studienarbeit wurden zunächst die verschiedenen Plattformen mobiler Endgeräte vorgestellt. Ausführlich wurde auf die Möglichkeiten der Softwareentwicklung für diese Betriebssysteme eingegangen. In der Gegenüberstellung wurden Vor- und Nachteile von Native Apps und Web Apps gegeneinander aufgewogen. Das Konzept der Hybrid Apps setzt genau dort an und versucht, alle Vorteile miteinander zu verbinden – vor allem die Plattformunabhängigkeit mit den nativen Features der Plattformen zu kombinieren. Außerdem wurde der jüngere Ansatz der Generated Apps kurz erläutert.

Anschließend wurde das Framework PhoneGap aufgesetzt und praktisch an einigen Anwendungsbeispielen getestet. Es konnte alle Anforderungen erfüllen, überzeugte allerdings nicht komplett, was an der Performance der Oberflächen liegt. Zum Vergleich wurden ähnliche Programmierversuche mit dem Titanium SDK vorgestellt. Es konnte durch sehr gute Performance und native Oberflächen punkten. Ausschlaggebend für die abschließende Wahl der Technologie für das Praxisprojekt war jedoch die Tatsache, dass Titanium die Audio-Aufnahme unter Android noch nicht unterstützt, was eine Grundvoraussetzung darstellt.

Im zweiten Teil der Arbeit wurde das Projekt StudiCast, ein Konzept zum mobilen Lernen, vorgestellt und praktisch umgesetzt. Dabei ist eine sehr gute Grundlage mit den wichtigsten Funktionen wie Audio-Aufnahme und Wiedergabe und Tauschen von Podcasts über einen zentralen Server entstanden. Das Projekt kann auf dieser Basis nun in die verschiedensten Richtungen weiter ausgebaut werden.

Als nächster Schritt könnten die Kommentar- und Bewertungsfunktionen implementiert werden. Ebenso sollte eine Web-Oberfläche für den Zugriff per Browser hinzugefügt werden. Langfristig muss die App dann auf weitere Plattformen wie iOS und Formate wie Tablets portiert werden, was mit dem entstandenen Quellcode relativ leicht möglich sein sollte. Der Aufbau einer Community, die das Projekt weiterführt und StudiCast mit Leben füllt, kann beginnen.

Ein besonderer Anreiz für das Projekt war – neben der Tatsache, etwas zu entwickeln, das später wirklich genutzt werden kann – der Wettbewerb BW goes Mobile. Hier schaffte es StudiCast unter die Gewinner-Konzepte.

## 6. Verzeichnisse

### 6.1. Abbildungsverzeichnis

Abbildung 1: App.java nach der Bearbeitung.....	17
Abbildung 2: Android SDK Manager.....	20
Abbildung 3: Android Virtual Device Manager .....	21
Abbildung 4: Eclipse Run Configurations .....	23
Abbildung 5: Android Device Chooser.....	24
Abbildung 6: Anlegen eines neuen Projekts in Titanium Studio (1) .....	30
Abbildung 7: Anlegen eines neuen Projekts in Titanium Studio (2) .....	31
Abbildung 8: Projektstruktur in Titanium Studio .....	32
Abbildung 9: Struktur der einzelnen Komponenten von StudiCast .....	39
Abbildung 10: Das data-Verzeichnis des Servers.....	40
Abbildung 11: Einfaches Datenbankmodell des Servers .....	41
Abbildung 12: Erweitertes Datenbankmodell des Servers .....	42
Abbildung 13: StudiCast-API .....	44
Abbildung 14: SQLite-Datenbankmodell auf dem Smartphone .....	46
Abbildung 15: Smartphone-Oberfläche von StudiCast .....	48

### 6.2. Tabellenverzeichnis

Tabelle 1: Probemessungen Aufnahmedauer und Dateigröße.....	38
---	----

### 6.3. Literaturverzeichnis

- [1] „Bundesamt für Sicherheit in der Informationstechnik,“ [Online]. Available: [www.bsi.bund.de](http://www.bsi.bund.de).
- [2] H. Behrens, „MobileTechCon 2010 - Heiko Behrens about Cross-Platform Mobile,“ [Online]. Available: <http://vimeo.com/20983647>. [Zugriff am 02 02 2012].
- [3] Appcelerator, „Appcelerator,“ [Online]. Available: <http://www.appcelerator.com/>.
- [4] C. Sandtner, „MACNOTES.de - Techtalk: Crosscompile Android Apps to iPhone,“ 7 12 2009. [Online]. Available: <http://www.macnotes.de/2009/12/07/techtalk-crosscompile-android-apps-to-iphone/>. [Zugriff am 05 02 2012].
- [5] W. Würmseer, „web&mobile DEVELOPER - Netbiscuits Tactile: HTML5-Cloud-Framework für Web-Apps,“ 15 03 2012. [Online]. Available: <http://www.webundmobile.de/Aktuelles/News/Netbiscuits-Tactile-HTML5-Cloud-Framework-fuer-Web-Apps>. [Zugriff am 25 03 2012].
- [6] „SQLite,“ [Online]. Available: <http://www.sqlite.org/>. [Zugriff am 30 05 2012].
- [7] „PhoneGap,“ Adobe, [Online]. Available: <http://phonegap.com/>.
- [8] Enough Software GmbH, „Mobile Developer's Guide to the Galaxy,“ Bremen, 2011.
- [9] A. Lunny, PhoneGap Beginner's Guide, Birmingham: Packt Publishing, 2011.