



ENTWICKELN VON ANWENDUNGEN FÜR HAND HELD

App für Erfassung von Garantiescheinen

Seminar Arbeit

Studenten: Andreas Grünenfelder
 Micha Schönenberger

Dozent: Christian Vils

© 2012

Dieses Werk einschließlich seiner Teile ist **urheberrechtlich geschützt**. Jede Verwertung ausserhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Autors unzulässig und strafbar. Das gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen sowie die Einspeicherung und Verarbeitung in elektronischen Systemen.

Zusammenfassung

!!!! HIER KOMMT NOCH TEXT !!!!

Abstract

!!!! HIER KOMMT NOCH TEXT !!!!

Inhaltsverzeichnis

Abkürzungsverzeichnis	III
Abbildungsverzeichnis	IV
Tabellenverzeichnis	V
Verzeichnis der Listings	VI
1. Einleitung	1
1.1. Das Projekt	1
1.1.1. Ausgangslage	1
1.1.2. Ziel der Arbeit	1
1.1.3. Aufgabenstellung	1
1.1.4. Erwartetes Resultat	2
1.1.5. Geplante Termine	2
1.1.6. Teaser	3
2. Projektplanung	4
2.1. Gantt Chart	4
2.2. Arbeitsaufwände	6
3. Grundlagen App Programmierung	7
3.1. Was wird für die App Programmierung benötigt	7
3.1.1. Eclipse (IDE)	8
3.1.2. Android SDK Plugin for Eclipse	9
3.1.3. Testing	10
3.2. Aufbau der Architektur Android App	11
3.2.1. Activity, View, Event, Intent	11
3.2.2. Lifecycle Activity	12
3.2.3. R.java	13
3.2.4. strings.xml	14
3.2.5. Manifest.xml	15
3.3. App-Vermarktung	17
3.3.1. Android Market - Google Play Store	17
3.3.2. public link	18
3.3.3. Mail, Stick...	18

4. Warranty App	19
4.1. Grundidee	19
4.2. Features	19
4.2.1. mögliche Erweiterungen	19
4.3. Aufbau der App	20
4.3.1. Klassendiagramm	20
4.3.2. Permissions	20
4.3.3. Datenbank	22
4.4. Core Komponenten	23
4.4.1. Ansteuerung der Kamera	23
4.4.2. Datenbank	23
4.4.3. Layouts	23
5. Fazit	24
5.1. Punkt 1	24
5.1.1. Punkt 1.1	24
A. Anhang	i
A.1. Verwendete Werkzeuge	i
A.1.1. Software	i
A.1.2. Hardware	i
A.2. Bilder	iii
A.2.1. Screenshot AppInventor	iii
A.2.2. Screenshot AppInventor Block Aufbau	iii
Index	iv
Literaturverzeichnis	vi

Abkürzungsverzeichnis

ADT	A ndroid D evelopment T ool
AVD Manager	A ndroid V irtual D evice Manager
DBMS	D atabase M anagement S ystem
IDE	I ntegrated D evelopment E nvironment
SDK	S oftware D evelopment K it

Abbildungsverzeichnis

1.1. abgegebener Teaser (05. Dezember 2012)	3
2.1. Gantt Chart Projekt Warranty	5
3.1. Logo Eclipse IDE	8
3.2. Beispiel AVD Manager	9
3.3. Lifescyle Activity	12
4.1. Klassendiagramm Activity	20
A.1. Screenshot AppInventor http://beta.appinventor.mit.edu/#3676309 . .	iii
A.2. Screenshot AppInventor Block Aufbau	iii

Tabellenverzeichnis

Verzeichnis der Listings

3.1. R.java	13
3.2. MainActivity.java	13
3.3. strings.xml	14
3.4. AndroidManifest.xml	15
4.1. Android permission for external Storage	21
4.2. Android permission for Camera	21

1. Einleitung

1.1. Das Projekt

1.1.1. Ausgangslage

Aus den ersten beiden Studienjahren haben wir uns die Grundkenntnisse der Java-Programmierung angeeignet. Wir möchten dieses Wissen nutzen, um ein neues Gebiet zu betreten (Native-App Android) und uns einem Thema zu widmen, das uns interessiert, wir aber bis anhin keine Zeit gefunden haben. Keiner von uns hat berufliche Programmiererfahrung. Deshalb liegt all unsere Erfahrung auf den schulischen Kenntnissen.

1.1.2. Ziel der Arbeit

Unser primäres Ziel ist es, einen Einblick in die Programmierung von Android Apps zu haben. Zusätzlich möchten wir unser bereits angeeignetes Java-Wissen auffrischen und vertiefen.

1.1.3. Aufgabenstellung

Wir möchten eine Android App erstellen, die es dem User ermöglicht, Garantiescheine in Form von einem Foto einfach lokal auf dem Smartphone zu verwalten. Die App soll die Möglichkeit bieten, zusätzliche Details in Form von Text zu speichern. Da im Fokus vor allem der Einblick in die App-Programmierung steht, verzichten wir bewusst gänzlich auf Netzwerk-Unterstützung. Des Weiteren ist das Backup der Fotos sowie der dazugehörigen Details nicht Teil dieser Arbeit, da dies unsere Zeitlimiten übersteigen würde.

1.1.4. Erwartetes Resultat

Das erwartete Resultat ist ein Native-App, welches auf Android funktioniert. Folgende Anforderungen müssen erfüllt sein.

- Kein Absturz der Applikation
- Lauffähig auf Geräten mit OS > 2.2 (Froyo) bis hin zum aktuellen 4.1.x (Jelly Bean)
- Fotos können aufgenommen und lokal gespeichert werden
- Es können Details in Form von Freitext zu den Fotos hinzugefügt werden
- Garantiescheine sollen nach folgenden Kriterien sortiert werden können
 - Speicherdatum des Garantiescheins
 - Alphabetisch nach Titel
 - Anzahl Tage bis Garantie ausläuft

1.1.5. Geplante Termine

Datum	Beschreibung
3 Oktober 2012	Einschreiben des Projektes im EBS
5. Dezember 2012	Abgabe Teaser im EBS
12. Dezember 2012	Arbeitstreffen
9. Januar 2013	Abgabe der Dokumentation
16. Januar 2013	Präsentation

1.1.6. Teaser

Am 5. Dezember 2012 musste per Mail ein Teaser abgegeben werden.

Definition Teaser gemäss Wikipedia:

Ein Teaser (von engl. tease = reizen, necken) ist in der Werbesprache ein kurzes Text- oder Bildelement, das zum Weiterlesen, -hören, -sehen, -klicken verleiten soll.

Da das Ziel eines Teaser ist, den Leser zur Applikation zu verleiten, sollte der Teaser auch die Applikation widerspiegeln. Aus diesem Grund wurde der Teaser bewusst ganz schlicht gehalten, genau so wie die daraus resultierende Applikation.



Abbildung 1.1.: abgegebener Teaser (05. Dezember 2012)

2. Projektplanung

2.1. Gantt Chart

Für die Projektplanung wurde ein Java-Tool benutzt, welches auf dem Gantt-Diagramm basiert. Es ist unter dem Namen [GanttProject](#) ¹ bekannt.

Ein Gantt-Diagramm oder Balkenplan ist ein nach dem Unternehmensberater Henry L. Gantt (1861-1919) benanntes Instrument des Projektmanagements, das die zeitliche Abfolge von Aktivitäten grafisch in Form von Balken auf einer Zeitachse darstellt.

Der Vorteil des Gantt Chart besteht darin, dass die Aktivitätsdauer durch die Balkenlänge wiedergegeben wird. Eine Ende-Start-Beziehung kann auch im Verlauf einer Aktivität ansetzen.

Der Nachteil liegt darin, dass Abhängigkeiten zwischen einzelnen Aktivitäten nur zeitbezogen dargestellt werden können.

Es gibt viele kostenlose sowie kostenpflichtige Software, die mit Gantt Chart arbeiten. Das wohl bekannteste lizenzpflichtige Programm ist das [Microsoft Visio](#) ² oder [Microsoft Project](#) ³. Quelle: [8] und [9]

¹offizielle Website <http://www.ganttproject.biz>

²<http://office.microsoft.com/de-ch/visio>

³<http://www.microsoft.com/project/en-us/Preview>

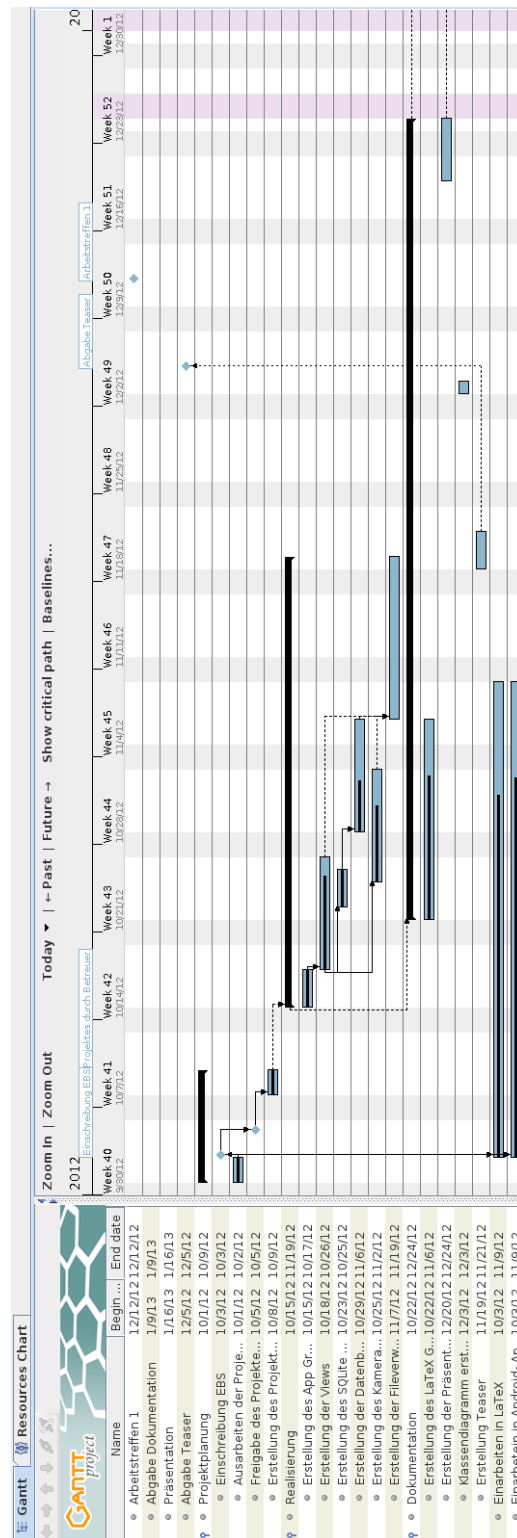


Abbildung 2.1.: Gantt Chart Projekt Warranty

2.2. Arbeitsaufwände

Bezeichnung	Aufwand geschätzt [h]	Aufwand effektiv [h]
Projektplanung		
Ausarbeitung Projektdetails	4	6
Erstellung Projektplanes	2	3
Realisierung I		
Grundgerüst Applikation	11	11
Views	9	9
SQLite DB Schemas	3	3
Datenbank Methoden	6	3
Kamerasupports	7	7
Fileverwaltung	4	4
Realisierung II		
DB Zugriffe überarbeiten	4	4
File und Fotoverwaltung überarbeiten	3	3
Views überarbeiten	3	3
Code Cleanup	5	5
Dokumentation und Präsentation		
Erstellung Teaser	3	2
Erstellung Logo	3	2
Erstellung LaTeX Grundgerüst	10	23
1. Einleitung	3	3
2. Projektplanung	3	3
3. Grundlagen App Programmierung	10	10
4. Warranty App	4	8
5. Fazit	2	2
Anhang	4	3
Klassendiagramm	3	5
Erstellung Javadoc	5	5
Erstellung Präsentation	12	8
Einarbeitung in LATEX	15	26
Einarbeitung Android Programming	12	20
Total Stunden	150	185

3. Grundlagen App Programmierung

3.1. Was wird für die App Programmierung benötigt

Es gibt eine ganze Reihe von Möglichkeiten, ein Android App zu programmieren. Neben Software, die es erlaubt, offline zu programmieren, gibt es von Google eine Online-Plattform, welche es erlaubt, ohne Programmierkenntnisse ein App zu erstellen. Diese Möglichkeit ist jedoch beim Umfang der Möglichkeiten beschränkt und läuft auf einer Beta-Phase. Für eine Nutzung von Googles [App Inventor](#)⁴ ist ein Google Account notwendig.

Ein Screenshot der Weboberfläche ([A.2.1 Screenshot AppInventor](#)) sowie der Textbausteine ([A.2.2 Screenshot AppInventor Block Aufbau](#)) sind im Anhang zu finden.

Da wir in den ersten beiden Studienjahren uns einige Java-Kenntnisse aneignen konnten, wird auf die Nutzung und das Austesten von Googles [App Inventor](#) verzichtet.

Die einfachste Weise ist die Nutzung der uns zum Teil bereits bekannten Frameworks. Nachfolgend werden alle benutzten Frameworks kurz erläutert.

⁴offizielle Website: <http://beta.appinventor.mit.edu/>

3.1.1. Eclipse (IDE)

Eclipse⁵ (vom englischen eclipse = Sonnenfinsternis hergeleitet) ist ein open-source Programmierwerkzeug. Zu Beginn wurde Eclipse als eine Entwicklungsumgebung für Java entwickelt. Im Laufe der Zeit hat sich Eclipse weiterentwickelt und durch die Möglichkeit der Skalierbarkeit wurde vom Java-Programmiertool ein Werkzeug, welches für viele Entwicklungsaufgaben eingesetzt werden kann. Die grosse Community und der modulare Aufbau, welche die Weiterentwicklung vom Modulen und Plugins immer vorantreiben, haben aus diesem Tool ein mächtiges Tool gemacht, welches sich für den Entwickler individuell zuschneiden lässt. Es gibt für Eclipse mittlerweile open-souce sowie auch kommerzielle Erweiterungen. Eclipse selbst basiert auf Java-Technik, seit Version 3.0 auf einem sogenannten OSGi-Framework namens Equinox.

Speziell für die Entwicklung von Android Applikationen existiert das ADT (Android Development Tools) Plug-in. Dieses Plug-in erweitert den Funktionsumfang von Eclipse und ermöglicht somit ein einfaches Entwickeln von Android Projekten. Eclipse wurde als Grundwerkzeug für die App-Programmierung benutzt. In den ersten beiden Studienjahren haben wir mit Eclipse Java-Applikationen entwickelt.

Quelle: [3]



Abbildung 3.1.: Logo Eclipse IDE

⁵ offizielle Website: <http://www.eclipse.org>

3.1.2. Android SDK Plugin for Eclipse

Das Android Software Development Kit (SDK) ⁶ [2] ist ein Plugin für Eclipse IDE welches als mächtige, integrierte Entwicklungsumgebung konzipiert wurde, um Android Applikationen zu entwickeln.

Will man beginnen, Android Apps zu programmieren, kommt man um das Android SDK nicht herum.

Die Android SDK gibt es für Windows, Mac OS X sowie Linux Plattformen.

Um Android SDK nutzen zu können, ist die Java SDK (Software Development Kit) unabdingbar. Diese ist je nach Betriebssystem bereits vorinstalliert oder kann nachträglich heruntergeladen und installiert werden.

Im Android SDK integriert ist der AVD Manager (Android Virtual Device Manager). Dieser ermöglicht das Testen der App in einer virtuellen Umgebung. Das Betriebssystem, die Speichermöglichkeiten sowie das Telefon können beliebig geändert werden.

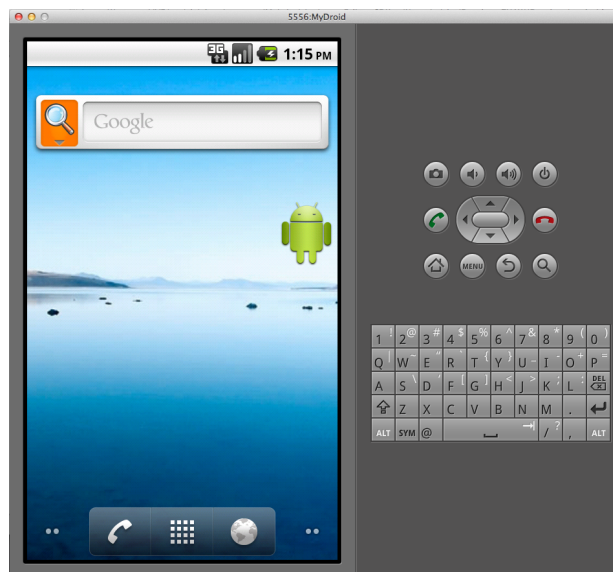


Abbildung 3.2.: Beispiel AVD Manager

⁶<http://developer.android.com/sdk/index.html>

3.1.3. Testing

Da die Zeit für dieses Projekt nicht ausreicht für ein ausgiebiges Testen mit JUnit- und JMock-Klassen, haben wir uns auf ein Testing beschränkt auf das Live-Testing. Zur Auswahl standen:

- Galaxy Nexus, Version 4.0.1
- HTC Desire HD, Version 2.3.5
- virtuelle Maschine, welche in Eclipse integriert ist und sich wahlweise die Android Version, aber auch Telefontyp ändern lässt

3.2. Aufbau der Architektur Android App

Wie bereits erwähnt werden Android Applikationen in Java geschrieben. Sind grundlegende Java Programmier Kenntnisse vorhanden, sollte der Einstieg kein Problem sein. Eine Grundlegende Änderung gilt es jedoch zu beachten:

- Fast alle Java-Klassen stehen zur Verfügung plus Verschlüsselung, HTTP, JSON, XML Bibliotheken
- Es existiert keine main()-Funktion wie bei klassischen Java-Applikationen
Es existieren stattdessen lose gekoppelte Komponenten. Eine oder mehrere davon werden als Einstiegspunkt - ähnlich wie main()-Methode - definiert.
- Die wichtigste Komponente ist die Activity: Sie entspricht einem sichtbaren Fenster auf dem Screen

Quelle (ganzes Unterkapitel): [\[7\]](#)

3.2.1. Activity, View, Event, Intent

Anbei kurz die wichtigsten Eigenschaften:

Activity

- definiert eine View, zur Anzeige auf dem Screen
- behandelt Events , z. B. Klick auf einen Button - onClick()
- benutzt Intents, um andere Activities zu starten

View

- die View ist der sichtbare Teil, welche auf dem Screen angezeigt wird
- ist definiert in einer XML-Layout-Datei (oder im Code)

Event

- Wird ausgelöst, wenn etwas geschieht (z. B ein Button geklickt wird)
- ruft eine Listener-Methode auf, sofern ein Listener definiert ist

Intent

- startet eine andere Activity - öffnet ein neues Fenster
- kann Daten an die zu startende Activity übergeben
- kann Activities aus anderen Apps starten!

3.2.2. Lifecycle Activity

Obwohl die Rechenleistungen und der Speicherplatz in den letzten Jahren bei Smartphones rasant anstieg, ist der Speicherplatz im Vergleich zu Workstation immer noch sehr begrenzt. Aus diesem Grund muss das Android-Betriebssystem nicht aktivierte Activities - also diejenigen, die nicht sichtbar sind - zu beenden. Jeder Activity stehen gemäss folgender Grafik die verschiedenen Zustände zur Verfügung. Auf die Details der einzelnen Zustände wird hier verzichtet.

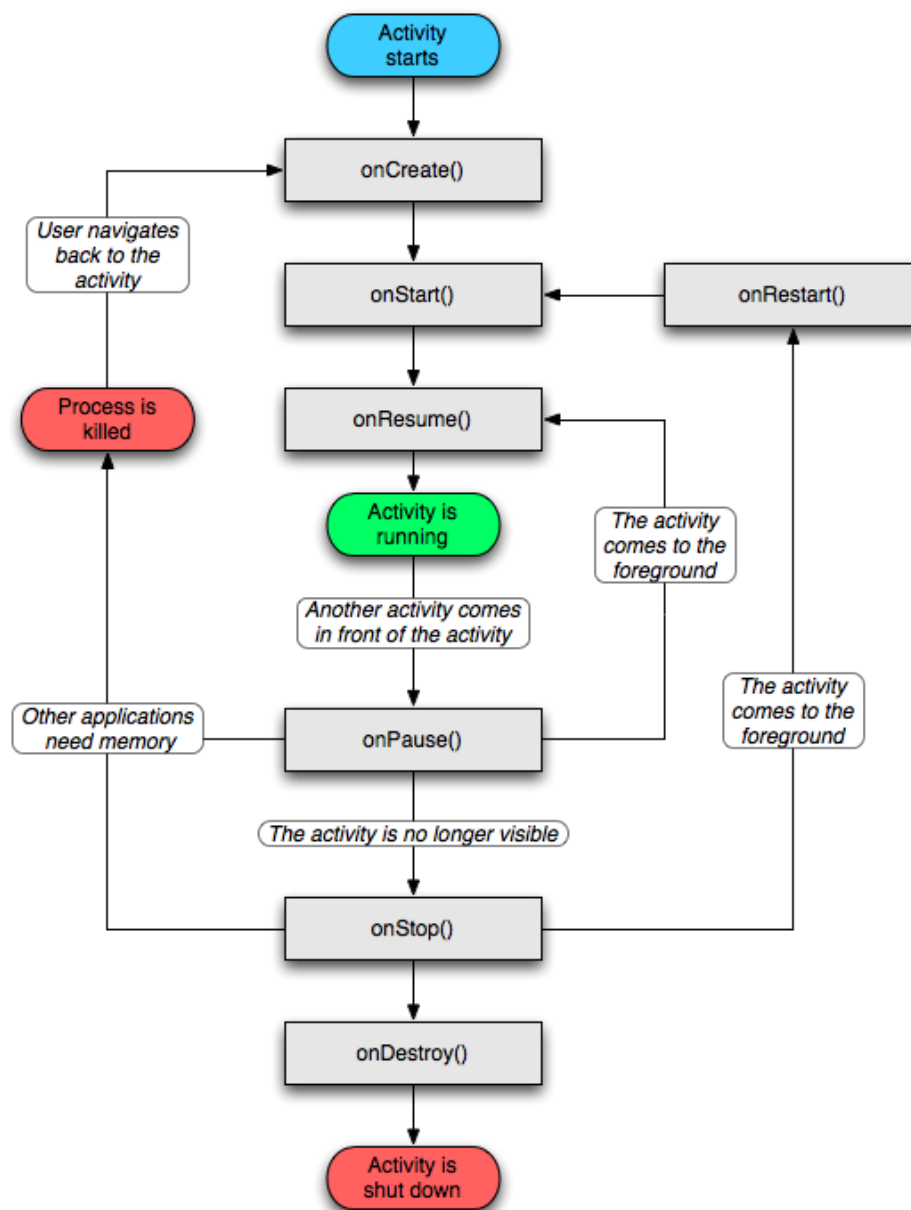


Abbildung 3.3.: Lifecycle Activity

3.2.3. R.java

R.java ist eine selbstgenerierte Java Klasse. Sie speichert für jede Ressource eine Integer-Konstante. Für die Applikationsentwicklung ist es nicht notwendig, diese Datei einzusehen. Anbei ein Ausschnitt der R.java Datei unser Applikation.

```
1  /* AUTO-GENERATED FILE. DO NOT MODIFY.
2  * This class was automatically generated by the
3  * aapt tool from the resource data it found. It
4  * should not be modified by hand.
5  */
6
7  package ch.zhaw.warranty;
8  public final class R {
9      public static final class id {
10         public static final int BTQuit=0x7f07000f;
11     }
12 }
```

Listing 3.1: R.java

Für die anderen Klassen hat R.java jedoch eine sehr grosse Bedeutung: Da alle Ressourcen in R.java eine konstante Integer zugewiesen ist, hat jede .java Klasse einen Verweis auf R.java, damit das richtige Layout geladen werden kann. Anbei ein Auszug aus MainActivity.java. Diese Activity ist die erste Activity, die geladen wird. Sie ist verknüpft mit dem Layout **activity_main**.

```
1  package ch.zhaw.warranty;
2  import ...;
3
4  public class MainActivity extends Activity {
5      public static TBLWarrantyConnector tblwarranty;
6
7      @Override
8      public void onCreate(Bundle savedInstanceState) {
9          super.onCreate(savedInstanceState);
10         setContentView(R.layout.activity_main);
11         ...
12 }
```

Listing 3.2: MainActivity.java

3.2.4. strings.xml

Die Datei **strings.xml** wird verwendet, um alle sichtbaren Texte, welche zur Laufzeit der App auf dem Bildschirm erscheinen, zu verwalten.

Als Programmierer sollte man beachten, dass eine Beschriftung eines Buttons nicht hard-coded wird, da die Flexibilität und die lose Kopplung verloren gehen. Wird stattdessen auf die Variable in der strings.xml Datei verwiesen, können alle App-Texte zentral verwaltet werden.

Ist dies einmal gegeben, ist ein Hinzufügen einer weiteren Sprache kein Problem mehr. Defaultmässig liegt die strings.xml Datei im Verzeichnis ../res/values/strings.xml und definiert die englische Sprache.

Möchte man nun eine weitere Sprache hinzufügen, erstellt man für die entsprechende Sprache einen neuen Ordner (Beispiel: ../res/**values-de**/). Nun kann die Datei strings.xml aus ../res/values/ kopiert werden und die Englischen Texte auf Deutsch angepasst werden.

Es gibt zwei Varianten, die Sprache der App zu ändern:

- Wird nichts eingestellt, wird die App in der Sprache gestartet, welche die Landeseinstellungen des Android Smartphone vorgeben. Ist die entsprechende Sprache in der App nicht vorhanden, wird per default Englisch benutzt.
- In der App kann ein Menüpunkt eingebaut werden, über welchen man auf jede beliebige Sprache, welche die App beherrscht, umstellen kann.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3   <string name="quit_app">Quit Warranty</string>
4   <string-array name="browse_list_spinner">
5     <item>Mercury</item>
6     <item>Earth</item>
7   </string-array>
8 </resources>
```

Listing 3.3: strings.xml

3.2.5. Manifest.xml

Die Datei **Manifest.xml** ist gewissermassen das Herzstück der App. Sie enthält die wichtigsten Informationen über die App [5]:

- enthält wichtige Informationen, damit die App auf einem System laufen kann
- Manifestdatei enthält Metadaten einer App (Paketname)
- Angabe genutzter Komponenten (Activities, Services, Broadcast Receivers, Content Providers)
- Fähigkeiten der App
- Voraussetzungen zum Betrieb (z.B. nötige Bibliotheken)
- Intent-Filter und IntentReceiver der Activities
- Zugriffsrechte auf andere Dienste und Rechte, die andere Apps für den Zugriff haben müssen
- Angabe von Angeboten für andere Apps
- Geforderte Android API

Welche Elemente die Manifest.xml Datei annehmen kann, sind auf der Android Developer Website ⁷ ersichtlich.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="ch.zhaw.warranty"
4     android:versionCode="1"
5     android:versionName="1.0">
6     <uses-sdk android:minSdkVersion="4" android:targetSdkVersion="12" />
7     <uses-feature android:name="android.hardware.camera" />
8     <uses-permission android:name="android.permission.
9         WRITE_EXTERNAL_STORAGE" />
10    <application android:icon="@drawable/icon" android:label="@string/app_name">
11        <activity android:name="ch.zhaw.warranty.MainActivity"
12            android:label="@string/app_name">
13            <intent-filter>
14                <action android:name="android.intent.action.MAIN" />
15                <category android:name="android.intent.category.LAUNCHER" />
16            </intent-filter>
17        </activity>
```

⁷<http://developer.android.com/guide/topics/manifest/manifest-intro.html>

```
16     <activity android:name="ch.zhaw.photobyintent.BrowseList"></activity>
17     ...
18 </application>
19 </manifest>
```

Listing 3.4: AndroidManifest.xml

3.3. App-Vermarktung

Online gibt es hunderte von Abhandlungen über die Vermarktungs-Strategie von Android Apps.

Es gibt Anleitungen, wie man seine App unter die 100 Besten im Play Store bringen kann, wie man den Preis festlegen soll, so dass der maximale Profit erwirtschaftet werden kann, wie man sich einen Namen macht als Entwickler... Da sind keine Grenzen gesetzt.

zwei Beispiele:

<http://www.androidpit.de/Strategische-Herangehensweise-bei-der-App-Entwicklung-Die-Idee-Teil-1>

<http://theappencypress.com/2010/08/04/everything-you-need-to-know-about-being-an-android-app-seller/>

3.3.1. Android Market - Google Play Store

Die wohl bekannteste Variante ist der Android Market. Seit dem 06. März 2012 heisst dieser offiziell Google Play [4] Store.⁸ Gemäss Angaben von Wikipedia [6] waren Ende Januar 2012 über 360'000 Anwendungen verfügbar, welche insgesamt über 10 Milliarden mal heruntergeladen wurden. Zirka 15% der Anwendungen sind Spiele. Der Umsatz beträgt mehr als 5 Millionen US-Dollar pro Monat.

Um eine Anwendung auf dem Play Store anbieten zu können, muss man sich als Entwickler auf <https://play.google.com/apps/publish/signup> registrieren und einmalig 25 US-Dollar bezahlen. Nach der Registrierung stehen einem Tür und Tor offen. Man kann seine Applikationen nun kostenlos oder zu einem selbst ernannten Preis vermarkten. Momentan sind etwa 65% der Applikationen kostenlos verfügbar.

Google verlangt, genauso wie Apple und Microsoft, eine Transaktionsgebühr von 30% des Verkaufswert.

Es gibt jedoch auch Alternativen [6] zu Googles Play Store:

- **F-Droid** Ein Appstore, der als non-profit Projekt von einer Community freiwilliger Unterstützer betrieben wird und über den ausschließlich kostenlose, freie Software-Apps (Open Source) bereitgestellt werden.

<http://f-droid.org/>

- **SlideME** SlidME bietet eine Plattform für Entwickler und Benutzer von Android. Entwickler können ihre Applikation kostenlos oder auch kostenpflichtig bei SlideME

⁸offizielle Website: <https://play.google.com>

veröffentlichen. Der SlideME Market umfasst ca. 2000 Applikationen.

<http://www.slideme.org>

- **AndroidPIT** AndroidPIT betreibt einen eigenen Store und bietet auch anderen Unternehmen diesen Store für ihre Android-Geräte an. Hierzu gehören Unternehmen wie 1&1, Telefunkn, Pearl, Point of View und Interpad.

<http://www.androidpit.de/>

- ... und noch viele mehr ...

3.3.2. public link

Will man seine Applikation nicht in einem online-Store wie Google Play Store anbieten, kann man sie auch direkt verkaufen oder kostenlos anbieten.

Hier liegt ein Vorteil von Google gegenüber anderen Anbietern, wie zum Beispiel Apple. Auf jedem Gerät, auf welchem die Android Plattform als Betriebssystem läuft, kann man die Funktion ein- und ausschalten, welche es erlaubt, auch Applikationen von Fremdanbietern (also nicht Play Store von Google) zu installieren. Dafür ist kein Jailbreak oder dergleichen notwendig.

Jeder, der diese Funktion nun eingeschaltet hat, kann die Applikation von jeglichem Webserver auf der Welt herunterladen.

Sicherheitshinweis: Hier sollte man unbedingt beachten, dass man auch schädliche Software installieren kann, wenn man erlaubt, aus nicht Google-Seiten Applikationen zu laden!

3.3.3. Mail, Stick...

Als dritte Variante bietet sich die direkte Vervielfältigung an. Diese kann per Mail, USB-Stick oder dergleichen erfolgen. Jede Android Applikation endet mit **.apk**. Ist das Smartphone am Computer angeschlossen (Windows, Mac OS X, Linux), kann die Applikation ohne Probleme installiert werden.

4. Warranty App

4.1. Grundidee

Die Grundidee der **Warranty-App** besteht darin, Garantiescheine zu verwalten. Es gibt 3 Möglichkeiten, keine Garantie auf ein Gerät zu erhalten:

- Garantie ist abgelaufen: hier gibt's nicht mehr zu helfen
- Gerät wurde zu unsorgfältig gehandelt. Selbstverschulden.
- Garantieschein ist verloren oder im Haushalt unauffindbar

Am dritten Punkt knüpft dieses App an. Ein Garantieschein kann nur noch dann verloren gehen, wenn die digitalen Daten weg sind. Mit einem Backup auf dem Rechner zu Hause oder in der Cloud ist dies nicht mehr möglich. In vielen Fällen schwirren die Garantiescheine irgendwo im Haushalt herum oder die schwarze Farbe gleich sich mit der Zeit dem weissen Papier an und nach ein paar Monaten bis Jahre ist vom Garantieschein nicht mehr zu lesen..

4.2. Features

Die bereits eingebauten Features sind bereits zum Teil bereits im Kapitel [1.1.4 Erwartetes Resultat](#) erwähnt worden.

4.2.1. mögliche Erweiterungen

Bei den Erweiterungen sind grundsätzlich den Phantasien keine Grenzen gesetzt. Wir beschränken uns auf die sinnvollen, im Alltag nutzbaren Erweiterungen.

- Möglichkeit, die Datensätze zu exportieren inkl. Bilder, welche sehr wichtig sind
- Abgleich der lokalen Datensätzen, bzw. Datenbank mit einem Clouddienst wie Dropbox, GoogleDrive...
- Versenden von einzelnen Datensätzen per E-Mail

Für viele dieser Erweiterungen wäre es sicherlich möglich, auf bereits geschriebenen Code der Android-App-Entwickler zuzugreifen. Somit müsste das Rad nicht neu erfunden werden.

4.3. Aufbau der App

4.3.1. Klassendiagramm

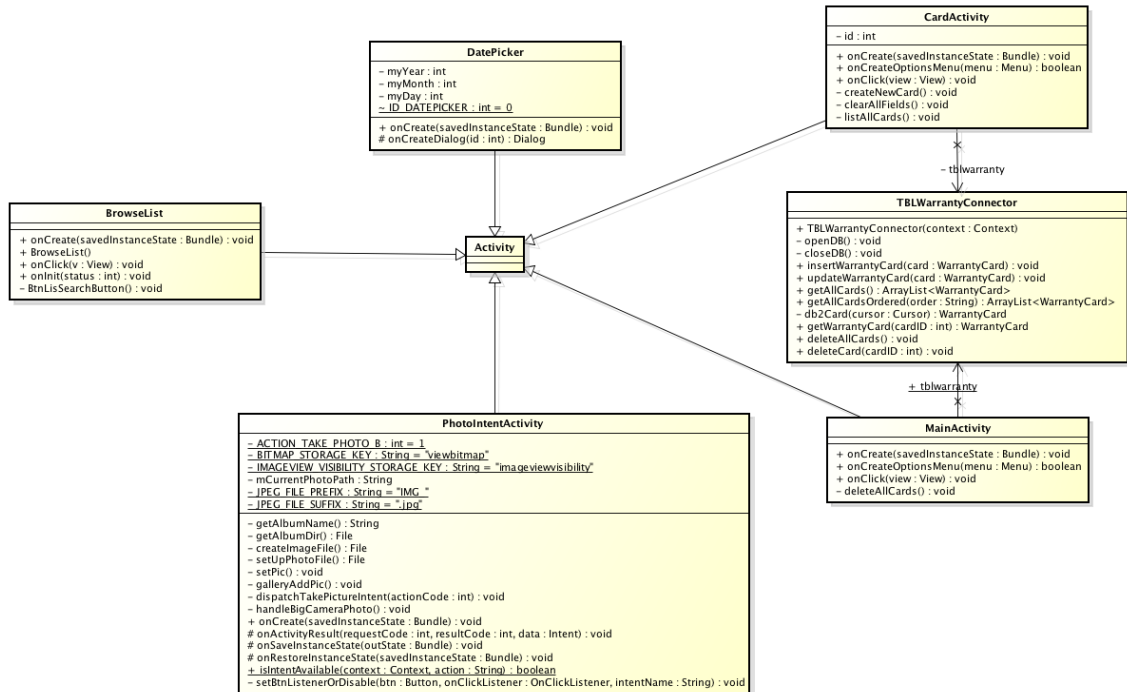


Abbildung 4.1.: Klassendiagramm Activity

!!!! HIER KOMMEN NOCH MEHR DIAGRAMME !!!!

!!!! KOMMT HIER NOCH TEXT? !!!!

4.3.2. Permissions

Die Warranty App benötigt wenige zusätzliche Rechte. Um Bilder speichern zu können ist jedoch ein Speicherplatz erforderlich. Da die meisten Android basierten Smartphones einen sehr kleinen internen Speicher haben, bietet sich der externe Speicher bestens an. Je nach Modell ist dies entweder eine zusätzlichen MicroSD [Glossar] Karte oder einfach eine zusätzliche logische Partition auf dem internen Speicher. Der Vorteil dieses externen Speichers ist, dass man ihn ohne weiteres auf dem Computer einhängen und auf die Dateien zugreifen kann.

Um auf diesen externen Speicher zuzugreifen, wird folgende Zeile im Manifest.xml benötigt.

```
1 <uses-permission android:name="android.permission.  
    WRITE_EXTERNAL_STORAGE" />
```

Listing 4.1: Android permission for external Storage

Nebst dem Zugriff auf den Speicher, bedient sich Warranty den Kamerafunktionalitäten. Da diese ebenfalls explizit erlaubt werden müssen, wird das entsprechende Recht im Manifest.xml hinterlegt.

```
1 <uses-permission android:name="android.permission.CAMERA" />
```

Listing 4.2: Android permission for Camera

4.3.3. Datenbank

Auch wenn wir auf unserem Smartphone eine Datenbank benötigen, so scheint die Idee, ein vollumfängliches DBMS (**D**atabase **M**anagement **S**ystem) wie beispielsweise MySQL oder PostgreSQL zu installieren absurd. Zum einen benötigen wir Features wie ein Client/Server Model, Partitioning oder ein ausgefeiltes Zugriffsberechtigungssystem nicht, zum anderen steht die dazu benötigte Performance auf einem Smartphone schlicht und einfach nicht zur Verfügung.

Um dennoch eine Datenbank auf einem Smartphone verwenden zu können, bietet sich SQLite an. SQLite ist eine Programmbibliothek, die sich direkt in der Applikation einbinden lässt und somit keinen Server- Prozess benötigt, also ressourcensparend ist. Die gesamte Datenbank inklusive aller Tabellen, Indizes und Werten werden in einer einzigen Datei abgelegt, was ein paralleles Schreiben auf die Datenbank unmöglich macht.

Dank der nativen SQLite Unterstützung von Android, fällt ein aufwändiges einbinden einer 3rd Party Library weg.

4.4. Core Komponenten

4.4.1. Ansteuerung der Kamera

!!!! HIER KOMMT NOCH TEXT !!!!

4.4.2. Datenbank

!!!! HIER KOMMT NOCH TEXT !!!!

4.4.3. Layouts

Die Warranty App basiert auf zwei Layouts, dem Homescreen mit der Übersicht über alle gespeicherten Quittungen sowie einer Detailansicht pro Quittung.

Der Homescreen der Applikation besteht aus einer Listview, die den oberen Teil des Bildschirms ausfüllt. Darin werden sämtliche gespeicherten Quittungen aufgelistet.

Mit der sich darunter befindenden Checkbox können zusätzlich auch bereits abgelaufene Garantiescheine angezeigt werden.

5. Fazit

!!!! HIER KOMMT NOCH TEXT !!!!

5.1. Punkt 1

!!!! HIER KOMMT NOCH TEXT !!!!

5.1.1. Punkt 1.1

!!!! HIER KOMMT NOCH TEXT !!!!

A. Anhang

A.1. Verwendete Werkzeuge

Im Folgenden werden die Hardware und Software vorgestellt, welche die Autoren zum Erstellen dieser Arbeit und vor allem zur Entwicklung der App verwendet haben. Es wurden ausschliesslich Open-Source-Programme eingesetzt.

Hier benutzte Beschreibungen können von Website (offizielle Site der Software, Wikipedia...) übernommen sein. Dieser Abschnitt dient zur Information für die verwendeten Werkzeuge.

A.1.1. Software

- **L^AT_EX**


Diese Arbeit wurde mit L^AT_EX geschrieben. Als Distribution und Editor wurde auf dem Mac OS Mountain Lion TexShop verwendet, auf Basis von Linux ???????????.

Websites: <http://pages.uoregon.edu/koch/texshop/>


A.1.2. Hardware

- **Galaxy Nexus**

Auf diesem Smartphone läuft das brandaktuelle Andoid OS 4.1.1 (Jelly Bean).

Bezeichnung	Version	
model number	Galaxy Nexus	
Android-Version	4.1.1 (Jelly Bean)	
Baseband-Version	I9250XXLF1	
Kernel-Version	3.0.31-g6fb96c9	
Build number	JR003C.I9250XWLH2	
Screen Resolution	1280 x 720 pixel	
diagonal	4.65 inch	

- HTC Desire HD A9191

Bezeichnung	Version	
model number	HTC Desire HD A9191	
Android-Version	2.3.5 (Gingerbread)	
Baseband-Version	12.65.60.29U_26.14.04.28_M	
Kernel-Version	2.6.35.10-g931a37e	
Build number	3.13.163.3 CL208029	
Screen Resolution	800 × 480 pixel	
diagonal	4.3 inch	

A.2. Bilder

A.2.1. Screenshot AppInventor

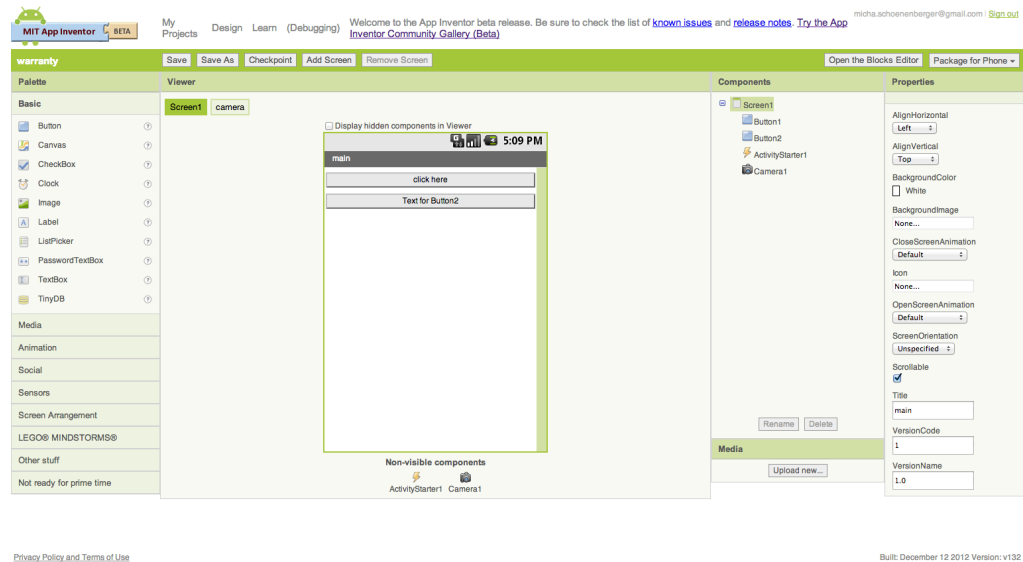


Abbildung A.1.: Screenshot AppInventor <http://beta.appinventor.mit.edu/#3676309>

A.2.2. Screenshot AppInventor Block Aufbau

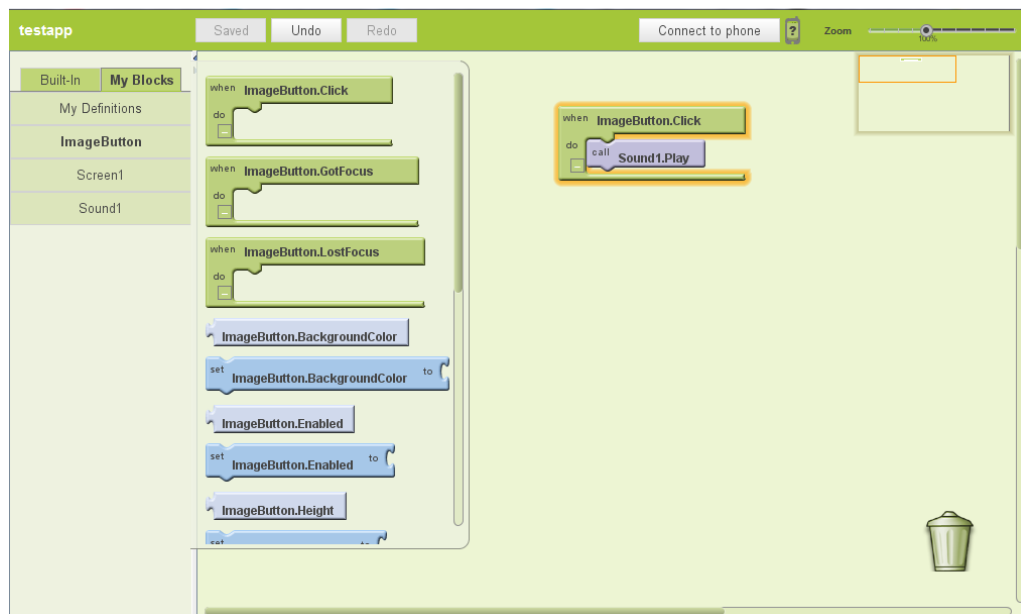


Abbildung A.2.: Screenshot AppInventor Block Aufbau
Quelle: [1]

Index

A

Activity	11
Android Market	<i>siehe</i> Play Store
Android SDK	9
AndroidPIT	18
App Inventor	iii, 7
App-Vermarktung	17
Architektur	11
Aufgabenstellung	1
Ausgangslage	1
AVD Manager	9

D

Datenbank	22
DBMS	22

E

Eclipse	8
Event	11

F

F-Droid	17
Features	19
Froyo	ii

G

Galaxy Nexus	i
Gantt Chart	4
Google Play Store	<i>siehe</i> Play Store

H

HTC Desire	ii
------------------	----

I

Intent	11
--------------	----

J

Jelly Bean	i
------------------	---

K

Klassendiagramm	20
-----------------------	----

L

Lifecycle Activity	<i>siehe</i> Activity
--------------------------	-----------------------

M

mögliche Erweiterungen	19
Mail	18
Main Activity	13
Manifest.xml	15

P

Permissions	20
Play Store	17
Projektplanung	4
public link	18

R

R.java	13
--------------	----

S

SlideME	17
Stick	18
strings.xml	14

T

Teaser 3

V

View 11

Z

Ziel der Arbeit 1

Literaturverzeichnis

- [1] *Google App Inventor*. http://4.bp.blogspot.com/_J5CxIaQd0mg/TLLqOKy0JUI/AAAAAAAAAS0/WjWVhrKNwFQ/s1600/App-Inventor-Blocks-Editor.PNG, october 2010. **A.2**
- [2] *Android Development Tools (ADT)*. <http://developer.android.com/tools/sdk/eclipse-adt.html>, november 2012. **3.1.2**
- [3] *Eclipse (IDE)*, *Wikipedia*. [http://de.wikipedia.org/wiki/Eclipse_\(IDE\)](http://de.wikipedia.org/wiki/Eclipse_(IDE)), november 2012. **3.1.1**
- [4] *Google Play Store*. <https://play.google.com>, december 2012. **3.3.1**
- [5] *Manifest Datei - Uni Dortmund*. <http://ls13-www.cs.uni-dortmund.de/dokuwiki-fachprojekt-ss11/lib/exe/fetch.php?media=lammers-vortrag.pdf>, november 2012. **3.2.5**
- [6] *Wikipedia - Google Play Store*. http://de.wikipedia.org/wiki/Google_Play, december 2012. **3.3.1**
- [7] *Eclipse (IDE)*, *Wikipedia*. <http://www.androidpit.de/de/android/wiki/view/Android-Anfänger-Workshop>, january 2013. **3.2**
- [8] *Gantt-Diagramm*. <http://de.wikipedia.org/wiki/Gantt-Diagramm>, january 2013. **2.1**
- [9] *Gantt Project*. <http://http://www.ganttproject.biz/download>, january 2013. **2.1**