



ENTWICKELN VON ANWENDUNGEN FÜR HAND HELD

App für Erfassung von Garantiescheinen

Seminar Arbeit

Studenten: Andreas Grünenfelder
 Micha Schönenberger

Dozent: Christian Vils

© 2013

Dieses Werk einschliesslich seiner Teile ist **urheberrechtlich geschützt**. Jede Verwertung ausserhalb der engen Grenzen des Urheberrechtgesetzes ist ohne Zustimmung der Autoren 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	15
3.2.5. Manifest.xml	16
3.3. App-Vermarktung	18
3.3.1. Android Market - Google Play Store	18
3.3.2. public link	19
3.3.3. Mail, Stick...	19

4. Warranty App	20
4.1. Grundidee	20
4.2. Features	20
4.2.1. Mögliche Erweiterungen	20
4.3. Aufbau der App	21
4.3.1. Klassendiagramm	21
4.3.2. Activities	22
4.3.3. Layouts	23
4.3.4. Menüs	24
4.3.5. Manifest.xml	25
4.3.5.1. Permissions	25
4.3.5.2. Application	27
4.3.6. Datenbank	28
4.3.7. Datenbanklayout	29
4.3.7.1. Definition der Tabelle	30
4.3.7.2. Insert und Update Funktion	31
4.3.7.3. Delete Funktion	32
4.3.8. Ansteuerung der Kamera	33
4.3.8.1. Externen Storage einbinden	33
4.3.8.2. Bilddatei erstellen	34
4.3.8.3. Foto aufnehmen	35
5. Fazit	37
A. Anhang	i
A.1. Verwendete Werkzeuge	i
A.1.1. Software	i
A.1.2. Hardware	ii
A.2. Bilder	iii
A.2.1. Screenshot AppInventor	iii
A.2.2. Screenshot AppInventor Block Aufbau	iii
A.2.3. Klassendiagramm logisches Verknüpfungen Datenbank	iv
Literaturverzeichnis	v

Abkürzungsverzeichnis

ADT	A ndroid D evelopment T ool
API	A pplication P rogramming I nterface
AVD Manager	A ndroid V irtual D evice Manager
DBMS	D atabase M anagement S ystem
IDE	I ntegrated D evelopment E nvironment
MicroSD	M icro S ecure D igital Memory Card
OSGi	O pen S ervices G ateway initiative
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. Lifecycle Activity	13
4.1. Klassendiagramm Activity	21
4.2. Flowchart Warranty	22
4.3. Layout CardListActivity	23
4.4. Layout CardActivity	24
4.5. TBLWarranty Schema	29
4.6. TBLWarranty Index	29
A.1. Screenshot AppInventor	iii
A.2. Screenshot AppInventor Block Aufbau	iii
A.3. Screenshot AppInventor Block Aufbau	iv

Tabellenverzeichnis

1.1. geplante Termine	2
2.1. Arbeitsaufwände	6
A.1. wichtigste Daten Galaxy Nexus	ii
A.2. wichtigste Daten HTC Desire HD	ii

Verzeichnis der Listings

3.1. R.java	13
3.2. MainActivity.java	14
3.3. strings.xml	15
3.4. AndroidManifest.xml	17
4.1. Definition eines Menüs, activity_card_list.xml	24
4.2. Erstellen von Menüs, CardListActivity.java	24
4.3. Activity ohne Menü, CardListActivity.java	25
4.4. Menü Listener, CardListActivity.java	25
4.5. Android permission für Zugriff auf externen Storage, Manifest.xml	26
4.6. Android permission für Kamera, Manifest.xml	26
4.7. Deklaration der Activities, Manifest.xml	27
4.8. Deklaration der Main- Methode, Manifest.xml	27
4.9. Deklaration des Tabellen- sowie der Attributnamen, TBLWarrantyHelper.java	30
4.10. Vorbereiten Erstellungsbefehls, TBLWarrantyHelper.java	30
4.11. Tabelle erstellen, TBLWarrantyHelper.java	31
4.12. Insert und Update Funktion, TBLWarrantyConnector.java	31
4.13. Delete Funktion, TBLWarrantyConnector.java	32
4.14. Zugriff auf SDCard, PhotoActivity.java	33
4.15. leeres File erstellen PhotoActivity.java	34

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 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

Tabelle 1.1.: 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>

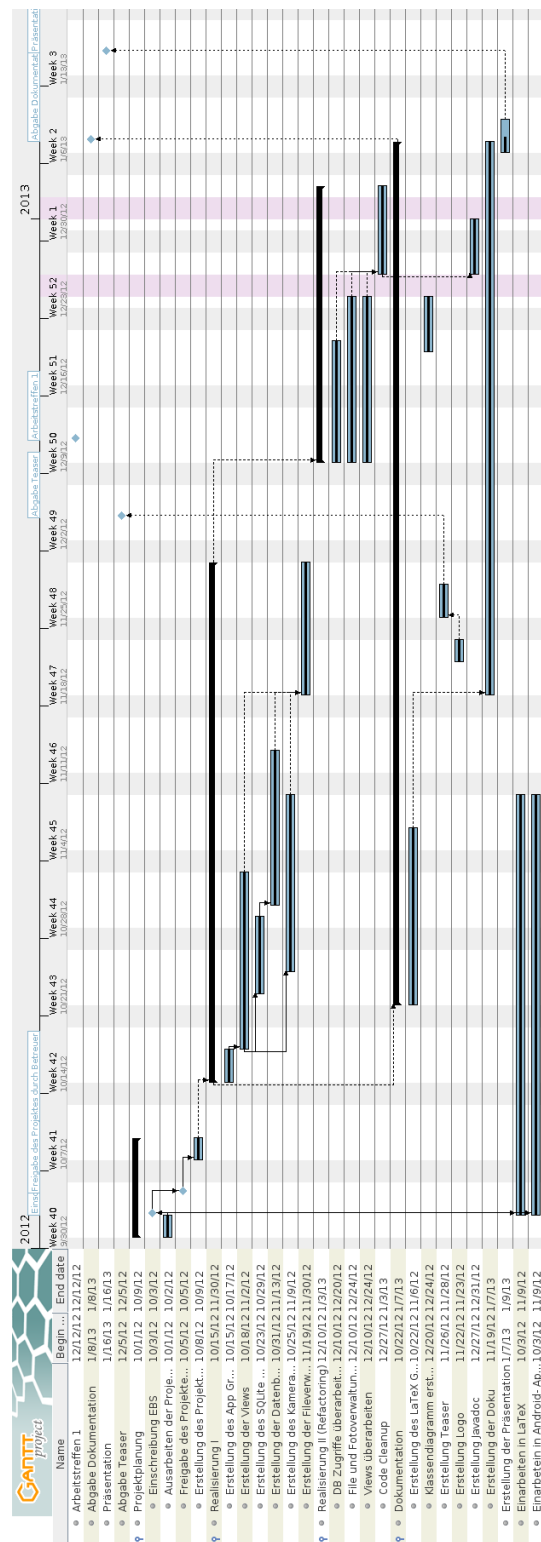


Abbildung 2.1.: Gantt Chart Projekt Warranty

2.2. Arbeitsaufwände

Tabelle 2.1.: Arbeitsaufwände

Bezeichnung	Aufwand geschätzt [h]	Aufwand effektiv [h]
Projektplanung		
Ausarbeitung Projektdetails	4	6
Erstellung Projektplanes	2	3
Realisierung I		
Grundgerüst Applikation	11	15
Views	9	12
SQLite DB Schemas	3	2
Datenbank Methoden	6	3
Kamerasupports	7	5
Fileverwaltung	3	4
Realisierung II		
DB Zugriffe überarbeiten	3	2
File und Fotoverwaltung überarbeiten	3	2
Views überarbeiten	3	6
Code Cleanup	5	9
Dokumentation und Präsentation		
Erstellung Teaser	3	2
Erstellung Logo	3	2
Erstellung LaTeX Grundgerüst	10	23
1. Einleitung	3	3
2. Projektplanung	3	2
3. Grundlagen App Programmierung	10	16
4. Warranty App	6	11
5. Fazit	2	2
Anhang (inkl. Lit.verzeichnis)	4	5
Klassendiagramm	3	5
Erstellung Javadoc	5	5
Erstellung Präsentation	12	14
Einarbeitung in LATEX	15	18
Einarbeitung Android Programming	12	20
Total Stunden	150	197

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 Variante 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 ([Screenshot AppInventor](#)) sowie der Textbausteine ([Screenshot AppInventor Block Aufbau](#)) sind im Anhang zu finden.

Da wir uns in den ersten beiden Studienjahren 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 Werkzeug 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.



Abbildung 3.1.: Logo Eclipse IDE

Quelle: [3]

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 das Eclipse IDE, dass 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.

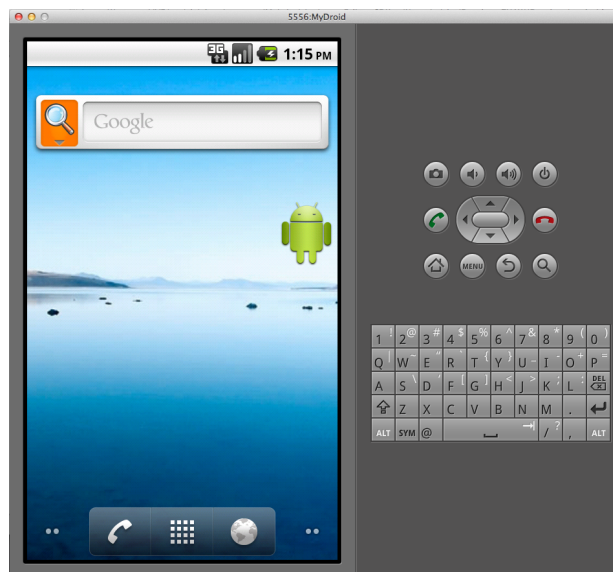


Abbildung 3.2.: Beispiel AVD Manager

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

3.1.3. Testing

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

- Galaxy Nexus, Version 4.0.1
- HTC Desire HD, Version 2.3.5
- virtuelle Maschine (im AVD Manager), 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 Programmierkenntnisse 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
- *Vorschlag* Fast alle Java- Klassen inklusive Unterstützung für SSL, HTTP, JSON und XML stehen zur Verfügung
- 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 die main()-Methode - definiert.
- Die wichtigste Komponente ist die Activity: Sie entspricht einem sichtbaren Fenster auf dem Display

Quelle: [7]

3.2.1. Activity, View, Event, Intent

Diese vier Begriffe sind die Grundsteine der Android Applikations-Entwicklung. 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 der App, der auf dem Display 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 angestiegen ist, ist der Speicherplatz im Vergleich zu Workstations noch immer sehr begrenzt. Aus diesem Grund muss das Android-Betriebssystem nicht aktivierte Activities - also diejenigen, die nicht sichtbar sind - beenden können. Jeder Activity stehen gemäss folgender Grafik die verschiedenen Zustände zur Verfügung. Auf die Details der einzelnen Zustände wird hier verzichtet.

Quelle: (Kapitel [3.2.1](#) und [3.2.2](#)): [\[7\]](#)

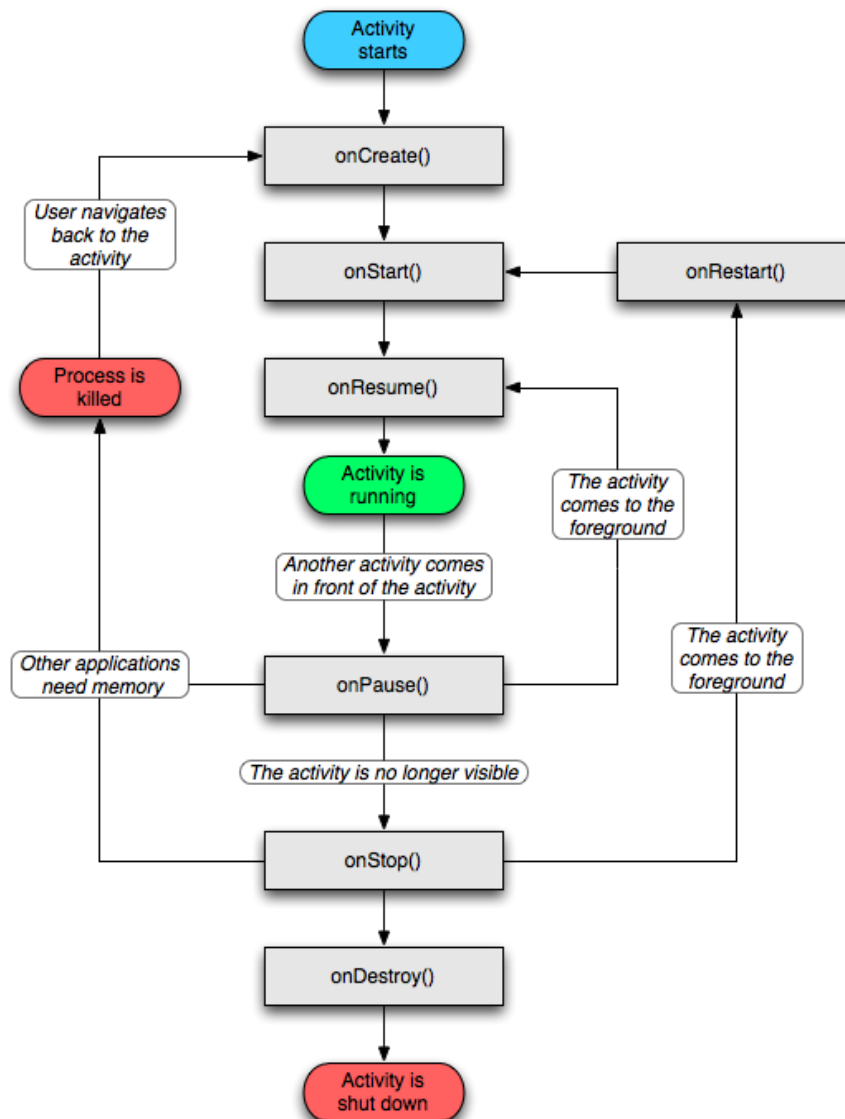


Abbildung 3.3.: Lifescyle Activity

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 allen Ressourcen in R.java eine konstante Integer-Variable 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 beachtet werden, 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/ 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 ausführbar ist
- Manifestdatei enthält Metadaten einer App (Paketname)
- Angabe genutzter Komponenten
Activities, Services, Broadcast Receivers, Content Providers
- Fähigkeiten der App
- Voraussetzungen für den 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.

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


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

Listing 3.4: AndroidManifest.xml

3.3. App-Vermarktung

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

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 im Google 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. Eigene Applikationen können nun kostenlos oder zu einem selbst ernannten Preis vermarktet werden. Momentan sind etwa 65% der Applikationen kostenlos verfügbar.

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

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

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** SlideME bietet eine Plattform für Entwickler und Benutzer von Android. Entwickler können ihre Applikation kostenlos oder auch kostenpflichtig bei SlideME 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, Telefunken, 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 Rooten 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 drei Möglichkeiten, die Garantie auf einem Gerät zu verlieren:

- 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 bleicht mit der Zeit aus und ist nach ein paar Monaten bis Jahre t vom Garantieschein nicht mehr zu lesen.

4.2. Features

Auf die Features der App wurde bereits im Kapitel **1.1.4 Erwartetes Resultat** eingegangen.

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 zum exportieren der Datensätze inklusive dazugehöriger Bilder
- Abgleich der lokalen Datensätzen, bzw. Datenbank mit einem Clouddienst wie Dropbox, GoogleDrive, etc.
- 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

Auf der folgenden Grafik sind die benutzten Activities sichtbar, die in dieser Applikation benutzt wurden.

Auf den logischen Zusammenhang der Activities wird im Kapitel [4.3.2 Activities](#) eingegangen.

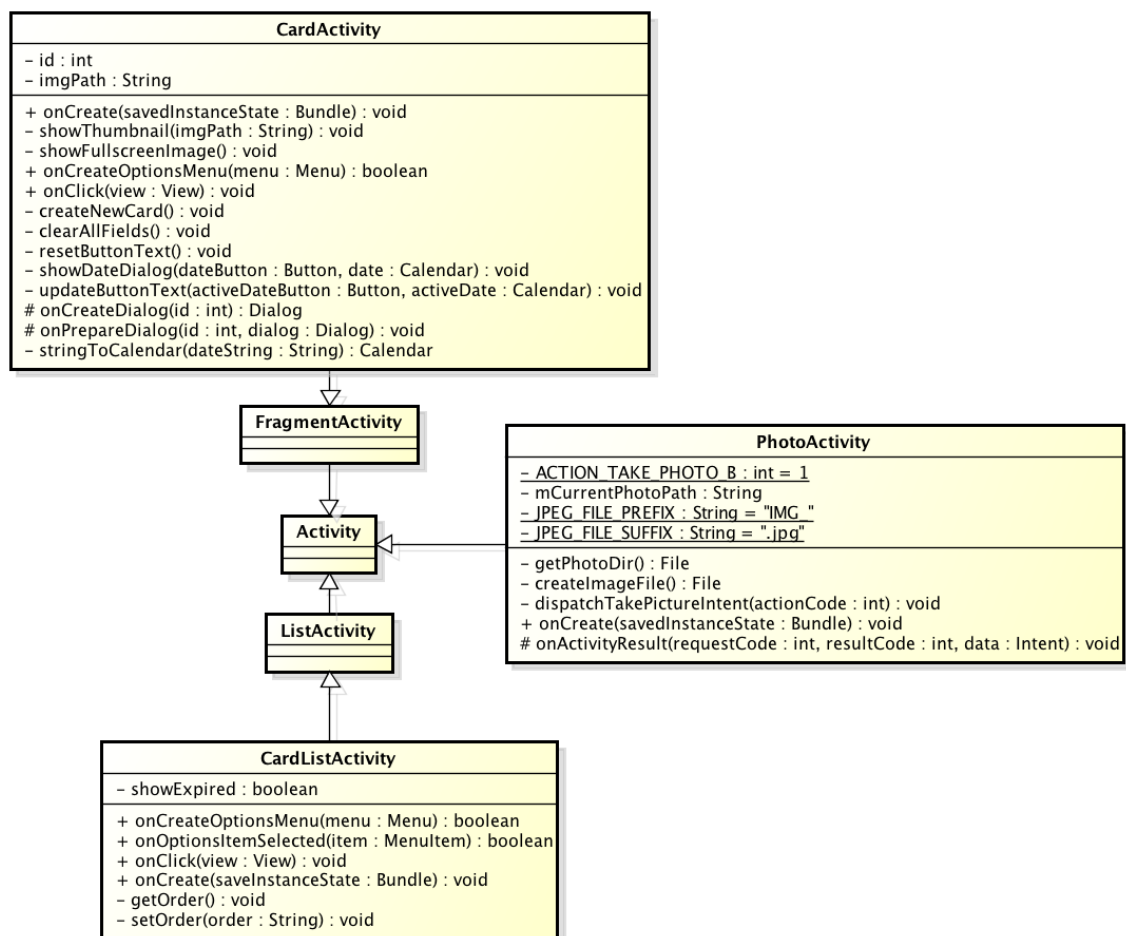


Abbildung 4.1.: Klassendiagramm Activity

Weiter Klassendiagramme befinden sich im Anhang ([Klassendiagramm logisches Verknüpfungen Datenbank](#) und ??)

4.3.2. Activities

Die Warranty App besteht aus drei verschiedenen Activities mit unterschiedlichen Aufgaben.

Die *CardListActivity* ist die eigentliche Haupt- Activity und dient als Einstieg in die App. Diese Activity stellt je eine Funktion zur Bearbeitung bereits existierender Quittungen sowie eine Funktion zur Erstellung eines neuen Eintrags bereit.

Beim Auswählen der Bearbeitungs- Funktion wird der Benutzer in die *CardActivity* geleitet, die es ihm ermöglicht sämtliche Details eines Eintrages zu bearbeiten. Mit der Speichern- Funktion wird der Eintrag in der Datenbank aktualisiert und der Benutzer zurück zur *CardListActivity* geleitet.

Die aufgenommen Bilder werden von der *CardActivity* aus mit Hilfe der von Android bereitgestellten Gallery App dargestellt.

Durch die Auswahl der Erstellungs- Funktion gelangt der Benutzer direkt die *PhotoActivity*, die nebst weiteren Funktionen die von Android bereitgestellte Kameraapplikation aufruft. Nach der Aufnahme eines Fotos wird ohne User- Interaktion in die oben erwähnte *CardActivity* gewechselt.

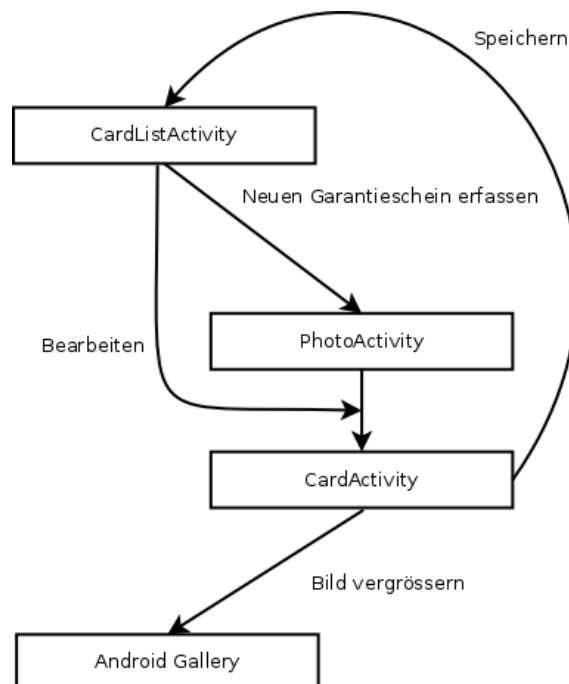


Abbildung 4.2.: Flowchart Warranty

4.3.3. Layouts

Von den in Kapitel 4.3.2 Activities beschriebenen Activities besitzen *CardActivity* und *CardListActivity* ein dazugehöriges Layout.

Das Layout von *CardListActivity* besteht neben den von Android vorgegebenen Teilen aus drei Parts. Zum ersten ist dies eine Listview, die es dem Programmier erlaubt, eine scrollbare Liste mit diversen Items zu erstellen. Die Aufgabe dieser Listview ist es, sämtliche gespeicherten Garantiescheine darzustellen. Darunter folgt eine Zeile mit einer Checkbox, die für die Filterung der angezeigten Einträge verantwortlich ist. Der unterste Teil wird von zwei Buttons belegt. Über diese wird die App gesteuert.

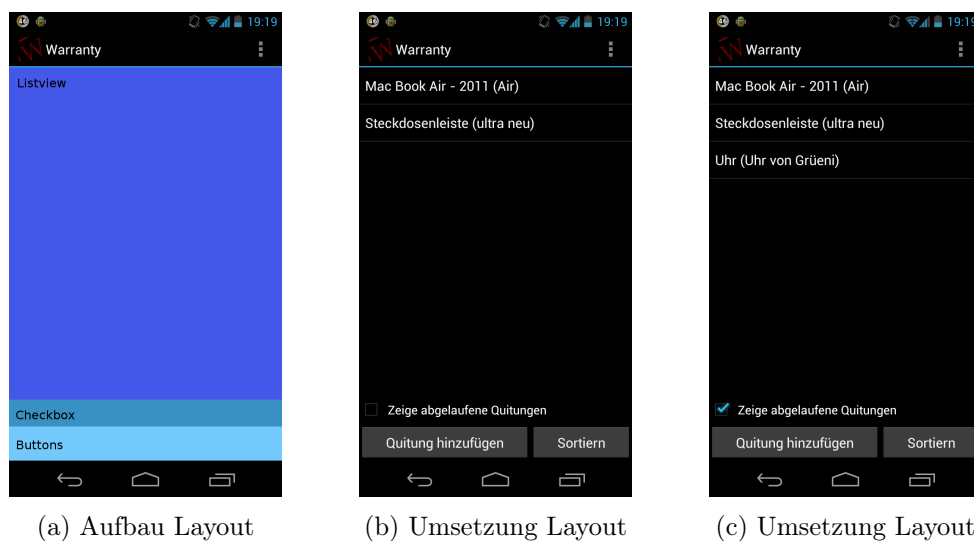


Abbildung 4.3.: Layout *CardListActivity*

Das zweite Layout, das CardActivity Layout besteht aus mehreren Komponenten. Die wichtigste Rolle spielen hier die verschiedenen EditText- Felder, die zur Eingabe von Daten benötigt werden. Daneben existieren diverse Buttons, beispielsweise um das Erstellungs- bzw. das Auslaufdatum eines Garantiescheines zu erfassen. Im untersten Drittel des Bildschirms ist eine ImageView zum darstellen von Bildern untergebracht.

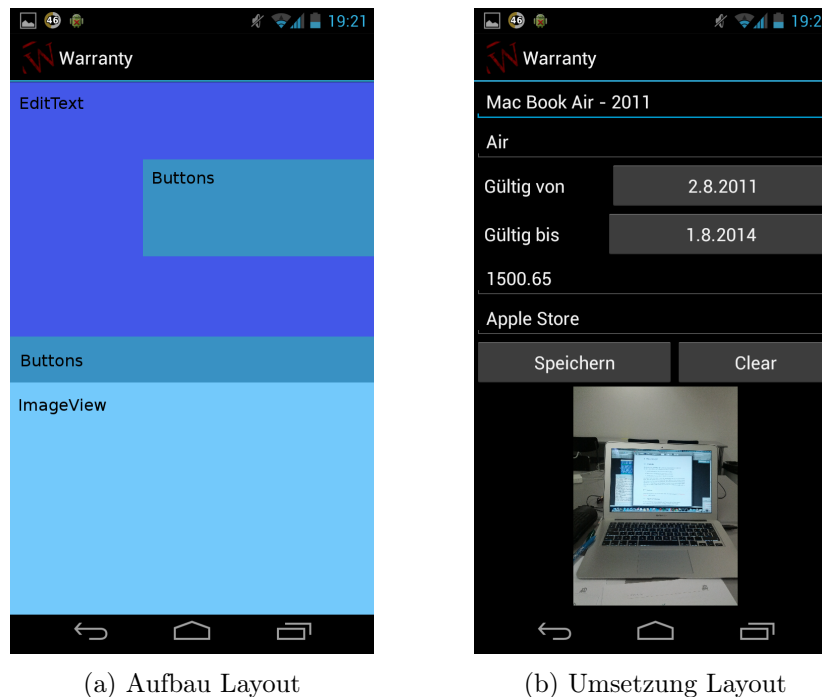


Abbildung 4.4.: Layout CardActivity

4.3.4. Menüs

Bei Warranty ist nur die CardListActivity, die Hauptactivity mit einem Menü ausgestattet. Die beiden anderen Activities benötigen dieses Feature von Android nicht.

Analog zu den Activities werden auch Menüs als XML- Files definiert.

```

1 <item android:id="@+id/cardlist.menu.DeleteAll" android:title="@string/
  deleteAllCards"></item>
2 <item android:id="@+id/cardlist.menu.exit" android:title="@string/exit"></item>

```

Listing 4.1: Definition eines Menüs, activity_card_list.xml

In der dazugehörigen Activity wird das Menü eingebunden und generiert.

```

1 @Override

```



```
2 public boolean onCreateOptionsMenu(Menu menu) {  
3     getMenuInflater().inflate (R.menu.activity_card_list, menu);  
4     return true;  
5 }
```

Listing 4.2: Erstellen von Menüs, CardListActivity.java

Besitzt eine Activity kein Menü, so wird dies in der *onCreateOptionsMenu()* einfach nicht generiert.

```
1 @Override  
2 public boolean onCreateOptionsMenu(Menu menu) {  
3     return false ;  
4 }
```

Listing 4.3: Activity ohne Menü, CardListActivity.java

Anschliessend wird in der Activity ausgewertet, welches Menü- Item ausgewählt wurde und die entsprechende Aktion ausgeführt.

```
1 @Override  
2 public boolean onOptionsItemSelected(MenuItem item) {  
3     switch (item.getItemId()){  
4         case R.id.cardlist_menu_exit:  
5             moveTaskToBack(true);  
6             break;  
7         case R.id.cardlist_menu_DeleteAll:  
8             tblwarranty.deleteCard(0);  
9             setOrder("title");  
10    }  
11    return true;  
12 }
```

Listing 4.4: Menü Listener, CardListActivity.java

4.3.5. Manifest.xml

4.3.5.1. 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 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 4.5: Android permission für Zugriff auf externen Storage, Manifest.xml

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 4.6: Android permission für Kamera, Manifest.xml

4.3.5.2. Application

Zusätzlich werden in der Manifest.xml sämtliche Activities hinterlegt, die von der App ausgeführt werden müssen. Dass diese Liste vollständig und korrekt ist, ist für die App überlebensnotwendig. Wird im Code eine App Activity aufgerufen, die in dieser Sektion nicht aufgeführt ist, stürzt die gesamte App ab.

```
1  <application
2      android:icon="@drawable/icon"
3      android:label="@string/app_name" >
4      <activity
5          android:name="ch.zhaw.warranty.CardListActivity"
6          android:label="@string/app_name" >
7          ...
8      </activity>
9      <activity android:name="ch.zhaw.warranty.CardActivity" />
10     <activity android:name="ch.zhaw.warranty.photo.PhotoActivity" />
11     <activity android:name="ch.zhaw.warranty.photo.PhotoDisplayActivity"/>
12 </application>
```

Listing 4.7: Deklaration der Activities, Manifest.xml

Nebst den zu erlaubenden Activities wird zusätzlich die zu startende Activity, in Programmierjargon "Main" definiert.

```
1  <activity
2      android:name="ch.zhaw.warranty.CardListActivity"
3      android:label="@string/app_name" >
4      <intent-filter>
5          <action android:name="android.intent.action.MAIN" />
6          <category android:name="android.intent.category.LAUNCHER" />
7      </intent-filter>
```

Listing 4.8: Deklaration der Main- Methode, Manifest.xml

4.3.6. 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 MySQL⁹ oder PostgreSQL¹⁰ zu installieren absurd. Zum einen werden Features wie ein Client/Server Model, Partitioning oder ein ausgefeiltes Zugriffsberechtigungssystem nicht benötigt, 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 SQLite¹¹ 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.

Um die Datenbankfunktionalität bereitzustellen, wurde ein eigenes Java- Package *ch.zhaw.warranty.database* erstellt. Die darin enthaltenen Klassen sind für das Erstellen der Datenbank und deren Tabellen (*TBLWarrantyHelper*) beziehungsweise zur Bereitstellung der Datenbankfunktionen für den User (*TBLWarrantyConnector*) zuständig.

⁹offizielle Website: <http://www.mysql.com>

¹⁰offizielle Website: <http://www.postgresql.org>

¹¹offizielle Website: <http://www.sqlite.org>

4.3.7. Datenbanklayout

Da die Anzahl der Datentypen in SQLite auf NULL, INTEGER, REAL, TEXT und BLOB beschränkt ist¹², ist das Datenbankschema von Warranty schnell definiert.

Um die Einzigartigkeit der Einträge zu gewährleisten, ist die ID als auto-increment gekennzeichnet. Somit können wir sicherstellen, dass auch IDs von gelöschten Einträgen nicht wiederverwendet werden.

Da SQLite keine Datentypen für Datum und Zeit zur Verfügung stellt, wir aber trotzdem Datumsstempel benötigen, greifen wir auf den Datentyp TEXT und Datumsfunktionen wie “date”¹³ von SQLite zurück.

Table warranty

Column	Type	Null	Auto	Comment
_id	INTEGER	no	yes	ID
title	TEXT			Titel der Warranty Card
description	TEXT			Beschreibung
img_path	TEXT			Pfad zum Bild
created_at	TEXT			Erstellungsdatum
valid_until	TEXT			Ablaufdatum
price	REAL			Preise
reseller	TEXT			Verkaufsstelle

Abbildung 4.5.: TBLWarranty Schema

Der einzige Index liegt auf der ID, da in der App sämtliche Aktionen mittels dieser ID als Referenz ausgeführt werden. Da wir keine Volltextsuche implementiert haben, benötigen wir kein weiteren Indizes.

Indexes

Column(s)	Type	Sort
_id	Primary Key	ascending

Abbildung 4.6.: TBLWarranty Index

¹²gemäss <http://www.sqlite.org/datatype3.html>

¹³gemäss http://www.sqlite.org/lang_datefunc.html

4.3.7.1. Definition der Tabelle

Um aus sämtlichen Java- Klassen auf den Tabellen- sowie die Attributsnamen zugreifen zu können und um ein statisches Referenzieren in den Java- Klassen zu vermeiden, macht es Sinn, diese als statische Strings zu definieren.

```
1 public static final String TBL_NAME="warranty";
2 public static final String CLMN_ID = "_id";
3 public static final String CLMN_TITLE = "title";
4 public static final String CLMN_DESC = "description";
5 public static final String CLMN_IMGPATH = "img_path";
6 public static final String CLMN_CREATEDAT = "created_at";
7 public static final String CLMN_VLDTIL = "valid_until";
8 public static final String CLMN_PRICE = "price";
9 public static final String CLMN_RESELLER = "reseller";
```

Listing 4.9: Deklaration des Tabellen- sowie der Attributsnamen, TBLWarrantyHelper.java

Darauffolgend wird das SQL-Statement zur Erstellung der Tabelle definiert. Die im Listing 4.3.8.3 aufgeführten Strings können hier bereits als Referenz benutzt werden.

```
1 private static final String DB_CREATE="create table " + TBL_NAME + " (" +
2   CLMN_ID + " integer primary key autoincrement," +
3   CLMN_TITLE + " text," +
4   CLMN_DESC + " text," +
5   CLMN_IMGPATH + " text," +
6   CLMN_CREATEDAT + " text," +
7   CLMN_VLDTIL + " text," +
8   CLMN_PRICE + " real," +
9   CLMN_RESELLER + " text);";
```

Listing 4.10: Vorbereiten Erstellungsbefehls, TBLWarrantyHelper.java

Abschliessend wird die *onCreate*-Methode der Klasse *Activity* überschrieben, so dass bei jedem erstellen der Activity das Tabellen- Erstellungsstatement aufgerufen wird. Existiert die Tabelle bereits, so hat das Statement keinen Einfluss, es wird von SQLite ignoriert.

```
1 @Override
2 public void onCreate(SQLiteDatabase db) {
3     db.execSQL(DB_CREATE);
4 }
```

Listing 4.11: Tabelle erstellen, TBLWarrantyHelper.java

4.3.7.2. Insert und Update Funktion

Die aus Anwendersicht vermutlich wichtigsten Methoden sind die Insert- und die Update-Methoden. Da diese von der Funktionalität sehr ähnlich sind, ist aus Sicht des Programmierers sinnvoll, diese zusammen zulegen. Der grundlegende Unterschied ist , dass beim Insert ein neuer Eintrag erstellt, beim Update ein bereits vorhandener Eintrag angepasst wird.

Im Javacode ist die Differenzierung denkbar einfach.

```
1 public void insertWarrantyCard(WarrantyCard card){
2     ...
3     openDB();
4     if (card.get_id() == 0) {
5         db.insert(TBLWarrantyHelper.TBL_NAME, null, values);
6     } else {
7         db.update(TBLWarrantyHelper.TBL_NAME, values, TBLWarrantyHelper.
8             CLMN_ID + "=" + card.get_id(), null);
9     }
10    closeDB();
11 }
```

Listing 4.12: Insert und Update Funktion, TBLWarrantyConnector.java

Der Grund für diese einfache Unterscheidung liegt in der Auflistung der Quittungen im Homescreen der App. Diese sogenannte Listview **referenz zum bild** kennt von jeder aufgelisteten Quittung ihre dazugehörige ID. Möchte der User eine Quittung bearbeiten, wird in der App intern diese Referenz auf die Quittung weitergegeben. Möchte der User eine neue Quittung hinterlegen, wird das ID Feld nicht gefüllt. Dies führt dazu, dass der Standard Integer- Wert, in Java eine 0¹⁴

4.3.7.3. Delete Funktion

Da die Quittungen nach deren Ablauf nicht automatisch gelöscht werden, kann es durchaus sein, dass ein User die Quittungen manuell löschen möchte.

Wie bereits im Kapitel **4.3.7.2 Insert und Update Funktion** erwähnt, ist der Listview auf dem Homescreen die ID jeder aufgelisteten Quittung bekannt. Somit kann eine Quittung direkt aus der Listview mit dem Methodenaufruf *deleteCard* und der dazugehörigen ID, die anschliessend das entsprechende SQL- Statement an die Datenbank kommuniziert, gelöscht werden.

Wie bereits beim Insert und Update haben wir uns auch hier die Tatsache, dass das **auto increment** von SQLite bei 1 beginnt¹⁵ zu nutzen gemacht. Wird 0 als ID übergeben, hat dies zur Folge, dass ausnahmslos alle gespeicherten Quittungen gelöscht werden.

```
1 public void deleteCard(int cardID) {  
2     openDB();  
3     if (cardID == 0) {  
4         db.delete(TBLWarrantyHelper.TBL_NAME, null, null);  
5     } else {  
6         db.delete(TBLWarrantyHelper.TBL_NAME, TBLWarrantyHelper.CLMN_ID + "=  
           " + cardID, null);  
7     }  
8     closeDB();  
9 }
```

Listing 4.13: Delete Funktion, TBLWarrantyConnector.java

Eine Funktion zum löschen aller Quittungen wurde im Menü des Homescreens untergebracht.

¹⁴gemäss <http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html>

¹⁵gemäss <http://www.sqlite.org/autoinc.html>

4.3.8. Ansteuerung der Kamera

Die Kamera stellt in Warranty eines der zentralen Teile dar, da die Hauptfunktion der App auf dem Speichern von Fotos basiert. Dementsprechend ist es wichtig, dass der Code, der diese Funktionalität bereitstellt übersichtlich und nicht aufgebläht ist und stabil läuft.

4.3.8.1. Externen Storage einbinden

Damit die aufgenommen Bilder in zukünftigen Releases einfach exportiert werden können, werden diese in einem eigenen Directory auf dem externen Storage gespeichert. Dies hat zusätzlich den Vorteil, dass der User beim anschliessen des Smartphones an den Computer dieses als externes Storage- Device mounten und somit einfach auf die aufgenommen Bilder zugreifen kann.

Zu Beginn wird der Zielspeicherort für die Bilder definiert und anschliessend geprüft ob

1. die SDCard gemountet ist
2. das Speicherverzeichnis existiert
3. das Speicherverzeichnis, falls inexistent, erstellt werden kann

Mit dieser Prozedur kann sichergestellt werden, dass die Kamera- Applikation Bilder schreiben kann.

```
1 private File getPhotoDir() {  
2     File storageDir = new File(Environment.getExternalStorageDirectory() + "/Warranty  
3         ");  
4     if (Environment.MEDIA_MOUNTED.equals(Environment.getExternalStorageState()  
5         )) {  
6         if (storageDir != null) {  
7             if (!storageDir.mkdirs()) {  
8                 if (!storageDir.exists()) {  
9                     Log.d("getPhotoDir()", "failed to create directory");  
10                    return null;  
11                }  
12            }  
13            ...  
14            ...  
15            return storageDir;  
16        }  
17    }  
18 }
```

Listing 4.14: Zugriff auf SDCard, PhotoActivity.java

4.3.8.2. Bilddatei erstellen

Da der von Android bereitgestellten Kamera- Applikation anschliessend ein leeres File übergeben werden muss, ist der nächste Schritt, ein leeres File zu erstellen.

Um die Eindeutigkeit des Files sicherzustellen, bedient sich Warranty den beiden Java-eigenen Klassen *Date* und *SimpleDateFormat*. Mit Hilfe der Klasse *SimpleDateFormat* wird der Dateinamen wie folgt aufgebaut: **yyyyMMdd_HHmmss**. Die von Warranty verwendeten Buchstaben haben gemäss Javadoc der Klasse *SimpleDateFormat*¹⁶ folgende Bedeutung:

y	Year
M	Month
d	Day
H	Hour
m	Minute
s	Second

Beispiel:

Der 12. Oktober 2012 zur Zeit 15:27:42 Uhr wird als 20121012_152742 dargestellt.

In Java kann das Erstellen eines leeren Files inklusive Zeitstempel im Filenamen mit ein paar wenigen Zeilen gemacht werden.

```
1 private static final String JPEG_FILE_PREFIX = "IMG_";
2 private static final String JPEG_FILE_SUFFIX = ".jpg";
3 ...
4 private File createImageFile() throws IOException {
5     String timeStamp = new SimpleDateFormat("yyyyMMdd_HHmmss").format(new
6         Date());
7     String imageFileName = JPEG_FILE_PREFIX + timeStamp;
8     File albumF = getPhotoDir();
9     File imageF = File.createTempFile(imageFileName, JPEG_FILE_SUFFIX, albumF);
10 }
```

Listing 4.15: leeres File erstellen PhotoActivity.java

¹⁶<http://docs.oracle.com/javase/6/docs/api/java/text/SimpleDateFormat.html>

4.3.8.3. Foto aufnehmen

Mit dem Überprüfen der Verfügbarkeit der SDCard sowie dem erstellen einer leeren Bilddatei sind die Bedingungen für das Aufnehmen eines Fotos erfüllt. Im nächsten Schritt kann die von Android zur Verfügung gestellte Kamera- Applikation mit dem leeren File als Parameter gestartet werden.

```
1 Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
2 switch (actionCode) {
3 case ACTION_TAKE_PHOTO_B:
4     File f;
5     try {
6         f = createImageFile();
7         mCurrentPhotoPath = f.getAbsolutePath();
8         takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT, Uri.fromFile(f));
9         ...
10    }
11 }
12 startActivityForResult(takePictureIntent, actionCode);
```

Die Kontrolle gelangt erst nach erfolgreichem speichern des Bildes durch die Kamera-Applikation wieder zurück zu Warranty. Warranty öffnet sogleich die Activity *CardActivity* mit dem Pfad der gespeicherten Datei und dem Status “new” als Parameter.

```
1 protected void onActivityResult(int requestCode, int resultCode, Intent data) {
2     switch (requestCode) {
3     case ACTION_TAKE_PHOTO_B: {
4         if (resultCode == RESULT_OK) {
5             Bundle imagePath = new Bundle();
6             imagePath.putString("path", mCurrentPhotoPath);
7             Intent intent = new Intent(PhotoActivity.this, ch.zhaw.warranty.CardActivity.
8                                     class);
9             intent.putExtra("path",mCurrentPhotoPath);
10            intent.putExtra("status","new");
11            startActivity(intent);
12        }
13    }
```

Von *CardActivity* werden anschliessend die *EditText* Felder geparkt und die in Kapitel **4.3.7.2 Insert und Update Funktion** vorgestellte *insertWarrantyCard*- Methode übergeben.

5. Fazit

Rückblickend können wir sagen, dass die App- Programmierung für Android basierte Devices ein sehr spannendes Thema ist. Nach einem gewissen Initialaufwand gelingt es auch Einsteigern schnell, eine kleine App mit ein paar wenigen Funktionen zu programmieren. Schnell konnten wir feststellen, dass mit ein bisschen Übung die Möglichkeiten der App- Programmierung auf Android fast unbegrenzt sind. Nach dem Einarbeiten steigt die Lernkurve rasant an bevor sie dann mit der Zeit abzuflachen beginnt.

Dank diversen Webseiten, Blogs und PDFs die im Internet frei verfügbar sind, kann der eigene Code mit anderem verglichen und allenfalls optimiert werden. Die verfügbaren Beispiele rufen nicht selten gute Ideen für die eigene App hervor. So kam es beispielsweise, dass wir die ursprünglichen, langweiligen Datumsfelder durch optisch ansprechendere DatePicker ersetzten konnten.

Da bei dieser Seminararbeit vor allem auch die Dokumentation im Vordergrund stand, war dies eine optimale Gelegenheit sich auf die kommende Semester- bzw. Bachelorarbeit vorzubereiten. Diese nutzten wir, indem wir uns zum Ziel setzten, die gesamte Arbeit in Latex und gemäss Best- Practice für technische Dokumentation zu schreiben.

A. Anhang

A.1. Verwendete Werkzeuge

Im folgenden Kapitel werden die Hardware und Software vorgestellt, welche zum Erstellen dieser Arbeit und vor allem zur Entwicklung der App verwendet wurden. 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 [TeXstudio](#).

Website TexShop: <http://pages.uoregon.edu/koch/texshop/>


Website TeXstudio: <http://texstudio.sourceforge.net>

A.1.2. Hardware

- **Galaxy Nexus**


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

Tabelle A.1.: wichtigste Daten Galaxy Nexus

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**

Tabelle A.2.: wichtigste Daten HTC Desire HD

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

A.2. Bilder

A.2.1. Screenshot AppInventor

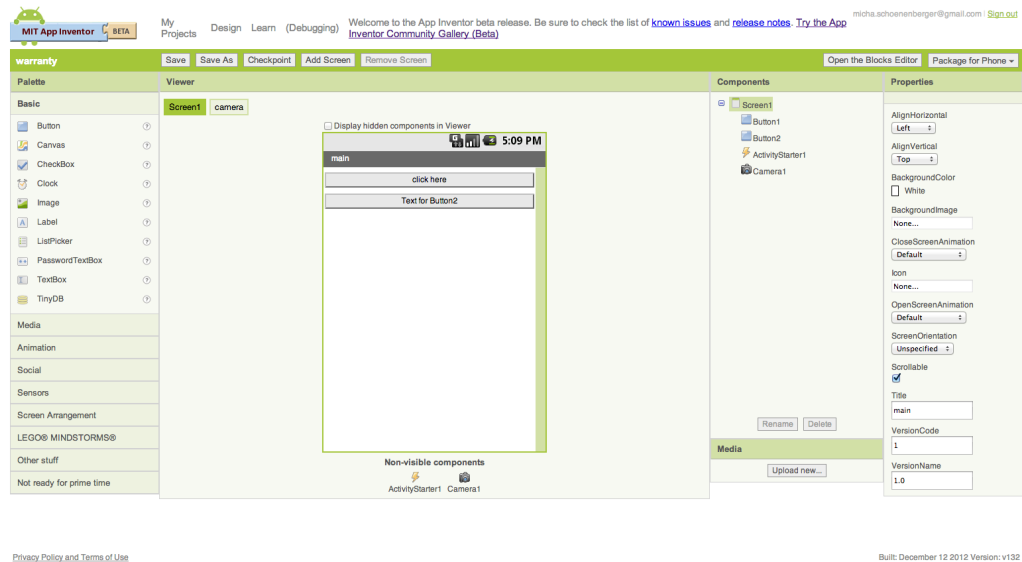


Abbildung A.1.: Screenshot AppInventor

Quelle: <http://beta.appinventor.mit.edu/#3676309>

A.2.2. Screenshot AppInventor Block Aufbau

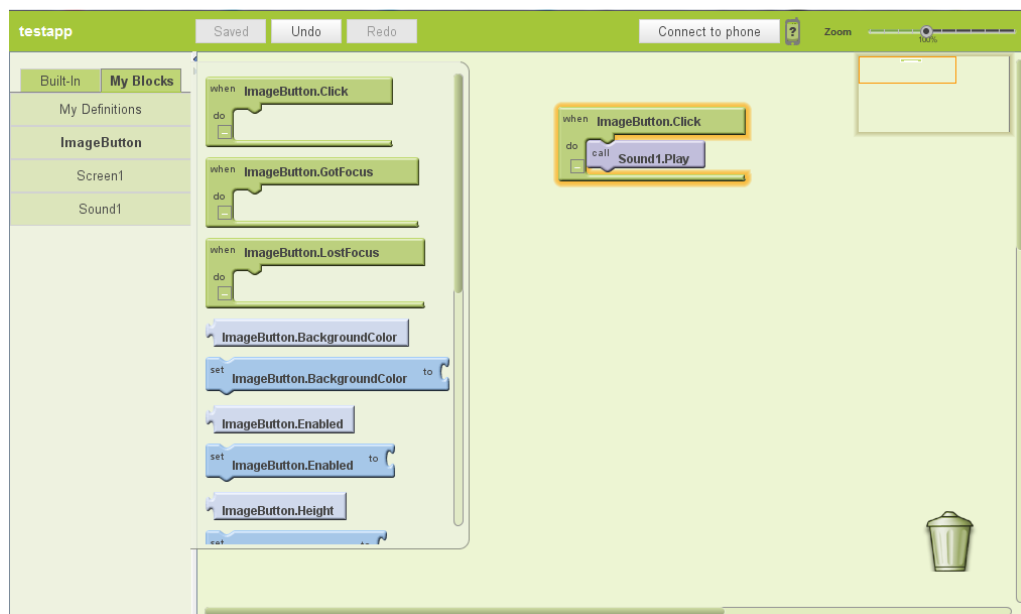


Abbildung A.2.: Screenshot AppInventor Block Aufbau

Quelle: [1]

A.2.3. Klassendiagramm logisches Verknüpfungen Datenbank

Auf dem folgenden Klassendiagramm sind die logischen Verknüpfungen rund um die Datenbank ersichtlich.

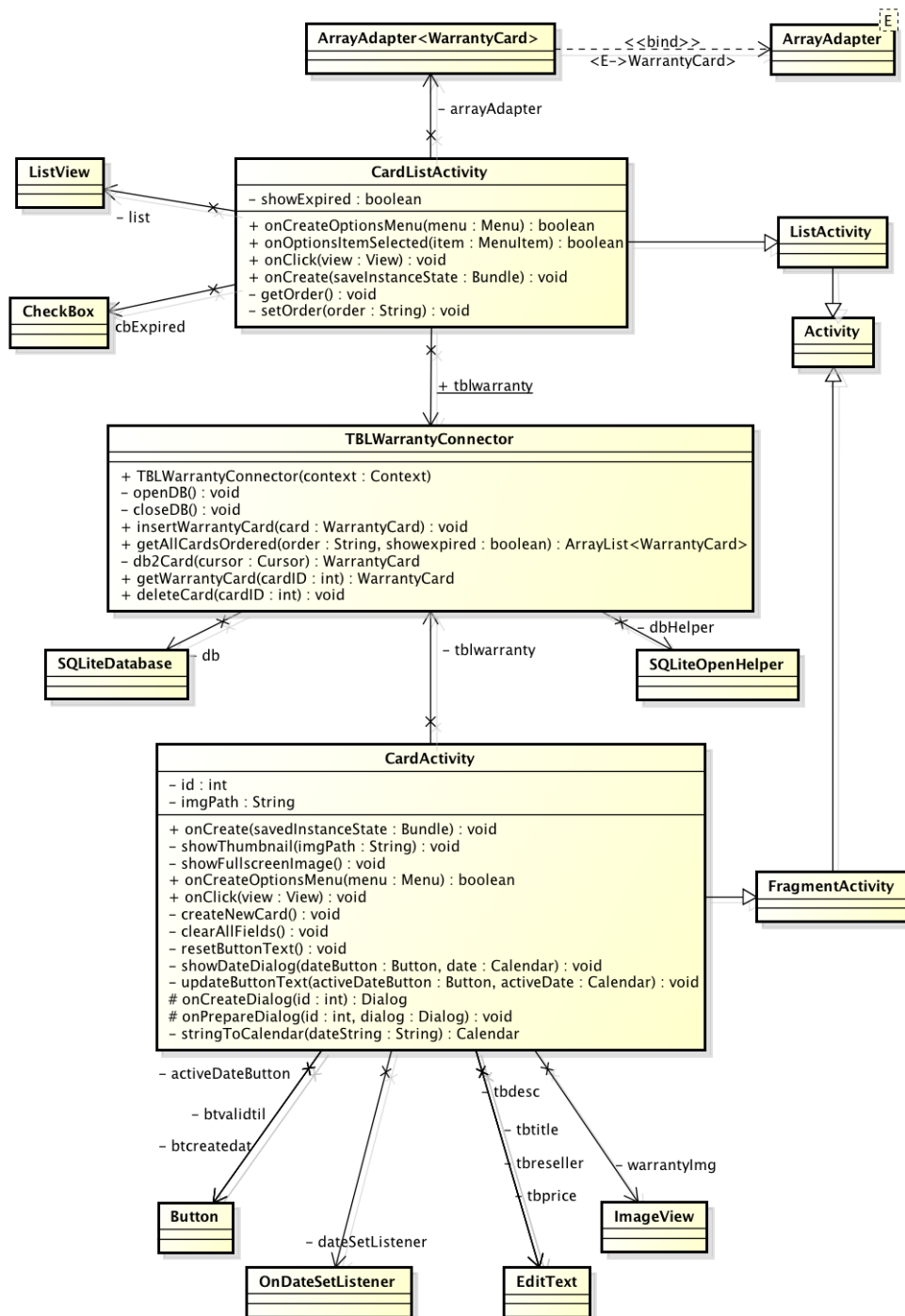


Abbildung A.3.: Screenshot AppInventor Block Aufbau
Klassendiagramm erstellt mit [astah professional](http://www.astah.net)
offizielle Website: <http://www.astah.net>

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, 3.2.2**
- [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**