Hochschule Merseburg (FH)

University of Applied Sciences

Konzeption einer Benutzerschnittstelle sowie Realisierung von Usability – und UI-Tests im Rahmen der prototypischen Entwicklung einer Android-App für eine Crowdtesting-Plattform

---

Bachelorarbeit

zur Erlangung

des akademischen Grades des Bachelor of Science

- B. Sc. -

Eingereicht von: Matthias Scharrig

Geboren am: 25.06.1992

Erstgutachter: Prof. Dr. rer. pol. Uwe Schröter (Hochschule Merseburg)

Zweitgutachter: Dipl. Inf. Robert Neumann (Softwareforen Leipzig GmbH)

Kreischau, 06. November 2013

Inhaltsverzeichnis

Abkürzungsverzeichnis

ADF	Android Developer Framework
MCT	Mobile Crowd Testing
UAV	uiautomatorviewer
UIA	uiautomator
App	Applikation
UI	User Interface

Abbildungsverzeichnis

Listingverzeichnis

1. Einleitung

Diese Bachelorarbeit befasst sich mit der Konzeption einer grafischen Benutzerschnittstelle sowie der Realisierung von Usability- und User Interface-Tests im Rahmen der prototypischen Entwicklung einer Android-App. Für den Test mobiler Applikationen (App) findet in den letzten Jahren der Ansatz des Crowdtesting immer weitere Verbreitung. Dabei testen Personen, ohne genaues Vorwissen über die Programmierung und Umsetzung, die mobile Applikation und ihre Funktionen auf auftretende Fehler. Typischerweise wird für die Abwicklung solcher Crowdtesting-Projekte eine Web-Plattform eingesetzt. Die Softwareforen Leipzig GmbH arbeitet an einem Prototyp für eine solche Plattform. Im Rahmen der Bachelorarbeit wurde eine Android-App zur mobilen Nutzung einer Web-Plattform und zur Unterstützung der Tester konzipiert und prototypisch entwickelt.
Das Ziel dieser Arbeit ist es, einen Überblick über Bedienkonzepte in Android-Apps zu vermitteln sowie ein einheitliches User Interface (UI) auf verschiedenen Android-Geräten zu erzeugen und die eigenständige Durchführung von Usability- und UI-Tests.
Diese Arbeit setzt sich dabei aus vier Abschnitten zusammen. Der erste Abschnitt beschäftigt sich mit den Begriffen Crowdsourcing und Crowdtesting. Im zweiten Abschnitt werden der konzeptionelle Aufbau der Crowdtesting-Plattform, die Ziele der App sowie die Anforderungen an einen Prototyp näher beschrieben. Der dritte Abschnitt beschreibt die theoretischen Grundlagen mit Hauptaugenmerk auf den Gestaltungsmöglichkeiten eines UI in Android und den Möglichkeiten zur Umsetzung von automatisierten UI-Tests auf mobilen Geräten. Außerdem wird die geplante Umsetzung des UI näher erläutert. Im vierten Abschnitt wird auf den genauen Aufbau der Android-App und die Durchführung der Usability- und UI-Tests eingegangen. Abschließend wird ein kurzes Fazit gezogen.

2. Überblick über Crowdsourcing und Crowdtesting

2.1. Merkmale Crowdsourcing

Bis heute gibt es keine allgemeine, wissenschaftliche Definition für das Crowdsourcing. Der Begriff steht für das gemeinsame Zusammenarbeiten von anonymen Nutzern einer Internetgemeinschaft. Jeff Howe verwendete den Begriff Crowdsourcing das erste Mal im Jahr 2006 [Wire06]. Mit „Crowd“ wird eine Gruppe von Menschen bezeichnet, die ein bestimmtes Arbeitsziel verfolgen. Crowdsourcing selbst ist eng an das aus der Ökonomie stammende Outsourcing angelehnt. Der Grund dafür ist, dass dabei auch interne Unternehmensaufgaben als ganze Prozesse extern ausgelagert werden. Der Unterschied ist, das die Auslagerung nicht an Zulieferer oder Subunternehmen erfolgt, sondern an Gruppen von arbeitswilligen Leuten, die über das Internet organisiert sind. Crowdsourcing gab es jedoch schon vor dessen Erfassung durch Jeff Howe im Jahr 2006. Dies waren in der Regel selbstorganisierte Gruppen, die in nichtkommerziellen Bereichen tätig waren. Open-Source-Projekte sind dafür ein perfektes Beispiel. Dort finden sich immer wieder Gruppen unterschiedlicher Größe zusammen, um arbeitsteilig ein Softwareprodukt zu erstellen, weiter zu entwickeln und es einem gewissen Nutzerkreis zur Verfügung zu stellen. [Clic13]
Crowdsourcing lässt sich allgemein in verschiedene Formen unterteilen. Dazu zählen

- Microworking,
- Collective Knowledge,
- Creative Content,
- Crowdfunding,
- Open Innovation.

Unter **Microworking** ist die Abwicklung von Kleinstaufgaben gegen eine geringe Bezahlung zu verstehen. Zu diesen Aufgaben zählen zum Beispiel das Verfassen von Texten oder das Tagging. Dies kann noch nicht komplett maschinell durchgeführt werden. Die entstanden Teilaufgaben werden häufig am Ende des Prozesses wieder zu einem komplexen Ganzen zusammengefügt. Die bekannteste Microworking-Plattform ist „Amazon Mechanical Turk“. Im Bereich des **Collective Knowledge** werden alle Formen des Crowdsourcing zusammengefasst, die Wissen sammeln, filtern und organisieren. Paradebeispiel hierfür ist Wikipedia.

Creative Content umfasst die Auslagerung von Kreativprozessen, beispielsweise im Design-Bereich. Dabei schreiben Firmen oder Privatleute bezahlte Aufträge für die Erstellung von Logos, Websites oder Bannern an die Crowd aus. Als **Crowdfunding** wird eine webgestützt Finanzierungsform bezeichnet, bei der die Crowd Projekte finanziert. Die Crowd erhält für ihren finanziellen Einsatz eine Gegenleistung, beispielsweise eine Provision bei der erfolgreichen Finanzierung eines Projektes. **Open Innovation** ist ein Brainstorming-Verfahren, bei dem gemeinsam mit der Crowd Problemlösungen und Produktideen erarbeitet werden. Beispielsweise werden auf der Website „InnoCentive.com“ wissenschaftliche Probleme an die Crowd weitergegeben, welche sich gegen Bezahlung mit dem Problem beschäftigt. [Grim12a]

Unternehmen erkannten schon früh die Möglichkeiten des Crowdsourcing. Bei der Verbesserung oder der Entwicklung von neuen Produkten kommt das Verfahren daher des Öfteren zum Einsatz, beispielsweise in den angesprochenen Designwettbewerben. Dies hat den einfachen Grund, dass die kreative Leistung einer großen Menschenmenge Gedanken und Ideen hervorbringen kann, an die die entsprechende Firma nie gedacht hätte. Mittlerweile gilt es als unstrittig, dass das Urteil einer repräsentativ großen Crowd mit einem Expertenrat gleichzusetzen ist. [Clic13]

2.2. Crowdtesting und Mobile Crowdtesting

Crowdtesting ist eine besondere Form des Crowdsourcing. Es bezeichnet die Anwendung des Crowdsourcing zum Testen von Software durch die Crowd. Dabei wird die Intelligenz und Arbeitskraft von Privatpersonen genutzt, die in ihrer Freizeit auf ihren privaten Geräten testen. Während der wachsenden Verbreitung von Smartphones und Tablets hat sich aus dem Crowdtesting das Mobile Crowdtesting (MCT) entwickelt. Es bezeichnet speziell das Testen von mobilen Applikationen durch die Crowd. [Ptex12]

In der Abbildung 2.1 ist das Grundkonzept des MCT dargestellt.

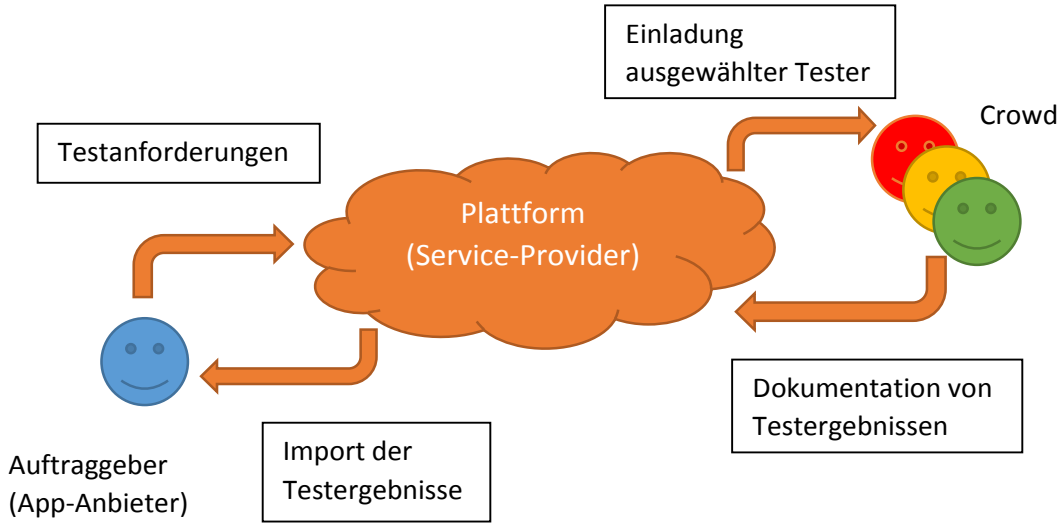

Abbildung 2.1: MCT Grundkonzept

Das folgende Szenario soll die Vorgänge in Abbildung 2.1 näher erläutern: Ein Auftraggeber(App-Anbieter) möchte eine neue Applikation für die bevorstehende Veröffentlichung auf z.B. Funktionalität, Bugs und Usability testen. Dafür erteilt er dem Service-Provider den Auftrag, diesen Test durchzuführen. Die Betreiber der Plattform erhalten nun alle notwendigen Daten über die App selbst sowie die Testanforderungen ihres Kunden. Im nächsten Schritt werden Tester aus der Community ausgewählt und dazu eingeladen, sich am Test zu beteiligen. Die Tester werden entsprechend den Anforderungen des Auftraggebers nach bestimmten Kriterien ausgewählt. Unter diese Kriterien fallen zum Beispiel bestimmte Hard- und Softwarekonfigurationen oder die Erfahrung der Tester. Beispielsweise kann eine große Menge von Testern eingeladen werden, um eine möglichst große Abdeckung an Geräte- und Softwareeigenschaften zu erreichen bzw. es kann eine kleine Menge von Testern eingeladen werden, welche dann über ein spezielles Gerät verfügen oder eine Kombination aus beiden. Der große Vorteil gegenüber dem internen Testen ist hierbei die Parallelisierung von Testaufgaben. Die ausgewählten Tester bearbeiten parallel dieselbe App, aber auf unterschiedlichen Geräten mit ihren unterschiedlichen Fähigkeiten, wodurch wenig individueller Aufwand für die einzelnen Tester entsteht, aber ein maximales Ergebnis für den Auftraggeber.

Hat ein Tester seinen Test abgeschlossen, dokumentiert dieser seine Testergebnisse in einem Testbericht auf der Webplattform und erhält ggf. eine Aufwandsentschädigung für den Test selbst oder für gefundene Bugs. Die Plattform wertet im

Nachhinein die Ergebnisse des Tests aus und stellt sie dem jeweiligen Auftraggeber zur Verfügung.

Crowdtesting-Projekte können zur Durchführung verschiedener Testarten genutzt werden. Dazu zählen unter anderem **Funktions-Tests**, **Sicherheits-Tests**, **Performance-Tests, Usability-Tests** und **Lokalisierungs-Tests**. Im Folgenden werden diese genauer erläutert. Das Hauptziel von **Funktions-Tests** ist es, die Funktionalitäten sowie die Zugänglichkeit des Softwaresystems zu überprüfen. Dabei konzentriert man sich gezielt auf einzelne Funktionen der Software. Außerdem wird die grundlegende Navigation getestet. Das heißt es wird getestet, ob ein Nutzer ohne Fehler durch die Software navigieren kann. [Guru13]
Sicherheits-Tests zeigen, ob die Anwendung über Sicherheitslücken verfügt. Im Verlauf des Tests werden auch gezielt Attacken auf zum Beispiel SQL-Datenbanken durchgeführt, um dortige Sicherheitsmängel aufzudecken. [Utes13a]
Performance-Tests beschäftigen sich mit der Performance und dem Verhalten der Anwendung unter sogenannten „real-world“-Bedingungen. Dazu zählen unter anderem Eigenschaften wie Netzqualität und Geschwindigkeit sowie Rechenleistung des Endgerätes. [Borl13]
Usability-Tests beschreiben die Benutzerfreundlichkeit der Anwendung. Der Zugang zu Feedback von potentiellen Kunden hilft vielen Firmen ihre Anwendungen anzupassen und weiterzuentwickeln. Usability-Tests sind regelmäßig in den Entwicklungsprozess fest eingeplant. [Utes13b]
Die korrekte Übersetzung aller Texte der Anwendung wird im Bereich der **Lokalisierungs-Tests** überprüft [Utes13c].

2.3. Vorteile und Nachteile des Crowdtesting

Auftraggeber profitieren zum einen von der Verbesserung der Qualität ihrer Anwendung sowie der Risikominimierung durch früh entdeckte Fehler. Die verbesserte Qualität der Anwendung kann dazu führen, dass aufgrund der Nutzerzufriedenheit und der positiven Bewertungen die Nutzerzahlen steigen, was somit zu einem großen Imagegewinn führen kann. Des Weiteren hat das Unternehmen einen geringeren Supportaufwand und kann deshalb seine Kapazitäten an anderen Stellen investieren. Eine Risikominimierung durch frühzeitiges Aufdecken von Fehlern resultiert überwiegend aus der hohen Abdeckung der real existierenden Endgerätekonfigurationen und Nutzungsszenarien, die durch die Crowd vorhanden

sind. Ein weiterer wichtiger Punkt für das Unternehmen sind die möglichen Kosteneinsparungen durch die Crowd. Der Auftraggeber muss keine teuren Geräte anschaffen und Fachpersonal damit beauftragen, die Anwendung zu testen. Er bezahlt nur, wenn auch tatsächlich getestet wird. Die Wartung dieser Geräte entfällt natürlich auch, da jeder Tester für sein eigenes Gerät verantwortlich ist. Das Unternehmen erhält durch die Crowd ein schnelles und wertvolles Feedback. Dieses Feedback kann verschiedener Art sein. Einerseits können es direkt Fehler in Form von Fehlerberichten sein, andererseits können es Ideen für Verbesserungen oder neue Features sowie Hinweise zu Usability-Problemen sein. Bei der Auswahl der Tester kann zu dem auf die Wahl einer bestimmten Zielgruppe geachtet werden. Beispielsweise wird beim Testen eines Online-Shops für Sportartikel eine Zielgruppe bestehend aus sportinteressierten Personen gewählt. Zum Schluss bleibt noch die enorme Zeitersparnis durch die Parallelisierung der einzelnen Tests. Dadurch hat der Auftraggeber schnell Ergebnisse zur Verfügung, was eine verkürzte Zeit zum Release zur Folge hat. [Ptex12]
Private Tester haben natürlich ebenso Vorteile, wenn sie die Tests der App-Anbieter in ihrer Freizeit durchführen. Ein Hauptvorteil ist die Zuverdienst Möglichkeit. Die Tests werden von den Service-Providern vergütet. Ein Service-Provider zahlt beispielsweise einen festen Grundbetrag pro Test und abhängig von der Anzahl der neu gefundenen Fehler einen entsprechenden Bonus [Test13]. Des Weiteren baut der Nutzer im Laufe der Zeit neues Wissen auf und erlernt neue bzw. verbessert schon vorhandene Fähigkeiten. Durch die flexible „Arbeitszeit“ bleibt es dem Nutzer selbst überlassen, wann, wie lange und ob er überhaupt am jeweiligen Test teilnimmt. Ein weiterer Vorteil ist der mögliche Aufbau und die Mitgestaltung einer neuen Community. Außerdem bietet sich dem Tester die Möglichkeit, vorab Zugriff auf neue Anwendungen zu erhalten.

Neben den genannten Vorteilen gibt es auch eine Reihe von Nachteilen für die App-Anbieter, beispielsweise spiel die Sicherheit und Diskretion hierbei eine wichtige Rolle. Anwendungen werden an unbekannte Tester herausgegeben. Dafür muss die Diskretion gewahrt bleiben und die Sicherheit gewährleistet werden. Der Testablauf sollte davon möglichst nicht beeinträchtigt werden. Das zu erwartende Feedback einzelner Tester ist oftmals qualitativ niedriger angesiedelt, als das von Testexperten, da die Tester meist Laien sind und im Erstellen von Testberichten wenig bis keine Erfahrungen haben. Jedoch können die Tester mit etwas Erfahrung qualitativ hochwertigere Berichte schreiben. Daher muss das Feedback ei-

nerseits von einer repräsentativ großen Anzahl von Testern vorliegen und andererseits müssen die Testberichte qualitativ den Vorgaben des Projektmanagers genügen, damit ernsthafte Konsequenzen aus dem Test gezogen werden können. Das Entlohnungssystem für Tester muss fair und anspornend sein, ansonsten verlieren die Tester schnell die Lust und das Feedback bleibt aus oder verliert an Qualität. [Zieg13]

3. Konzept der Crowdtesting-Plattform und der App

3.1. Konzeptioneller Aufbau der Crowdtesting-Plattform

Die Struktur der angestrebten Crowdtesting-Plattform besteht, wie in Abbildung 3.1 dargestellt, aus einer Datenbank, einer App für die Betriebssysteme Android und iOS sowie einer Webplattform mit einer Schnittstelle zur Datenübertragung an die Apps.

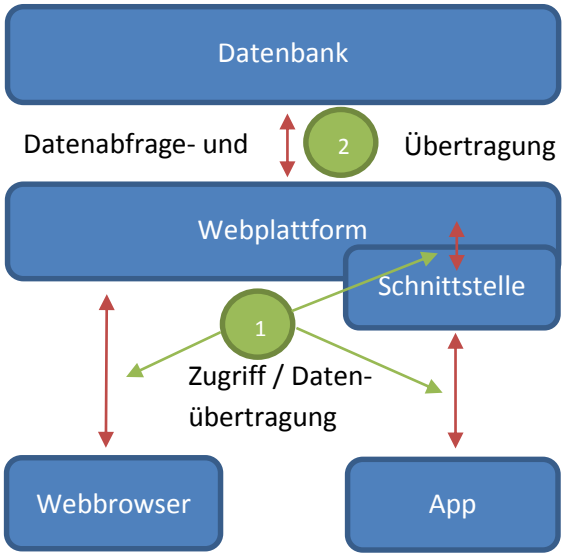

Abbildung 3.1: Struktur der Crowdtesting-Plattform

PC-Nutzer werden über ihren Webbrowser und Smartphone- bzw. Tablet-Nutzer über die App auf die Webplattform zugreifen können (1 in Abbildung 3.1). Die Webplattform bezieht über eine Datenbank ihre notwendigen Daten (2 in Abbildung 3.1). Die Datenbank beinhaltet alle Daten eines Nutzers, dazu zählen beispielsweise dessen Account-Daten, Nachrichten, Gerätedaten oder aktuelle Testberichte. Die Datenbank ist jedoch nur über die Webplattform erreichbar, daher wird als Verbindung zwischen App und Webplattform eine Schnittstelle benötigt. Die mobile App für Android bzw. iOS soll den Nutzern zwar die Grundfunktionen der Webplattform bieten, aber gleichzeitig soll diese die Webplattform nicht vollkommen ersetzen. Deshalb ist das Hauptaugenmerk der App die Unterstützung beim Testen von Applikationen.

3.2. Geplante Funktionalitäten der mobilen Applikation

Die Android-App soll dem Tester die wesentlichen und für mobile Verwendung geeigneten Funktionen wie die Webplattform bieten. Das Hauptaugenmerk liegt dabei beim Verfassen von Testberichten, einer Unterstützung bei der Testdurchführung, einem Überblick über alle aktuellen sowie verfügbaren Tests, einem internen Nachrichtensystem sowie dem Verwalten des Tester-Profils. Die App soll dabei alle benötigten Daten über die Datenbank der Webplattform beziehen (z.B. Profildaten, laufende Tests, etc.). Das in der App verwendete Nachrichtensystem ist das der Webplattform. Die App ruft dazu die Daten über die Webplattform ab und versendet auch Nachrichten über diese. Es sollen dem Tester verschiedene Einstellungen angeboten werden, um die App zu personalisieren und somit eine bessere Nutzungserfahrung zu gewährleisten. Außerdem soll der Tester seine Geräte, die er besitzt und für die er gerne Tests erhalten möchte in einem Gerätemanager verwalten können. Um den Nutzer bei der Dateneingabe eine wenig zu unterstützen wird es eine Option geben, die Hardware und Softwareeigenschaften des eigenen Gerätes automatisch zu erkennen, anstatt diese manuell eingeben zu müssen. Dem Nutzer soll etwas Komfort geboten werden, indem die zu testenden Apps per Direktdownload abrufbar sein sollen. Außerdem soll sich der Nutzer nur einmalig in der App anmelden müssen, später soll der Login beim Starten der App automatisch erfolgen. Gegebenenfalls soll die App dazu Daten intern auf dem Gerät in einer Datenbank speichern.

Das Layout der App soll funktional sein, dabei sollen die aktuellen Standards der Navigation verwendet werden. Dazu zählen unter anderem das Öffnen neuer Seiten, das hin und her „Wischen“ zwischen mehreren Seiten, die Anzeige von Dialogen zur Eingabe von Daten sowie eine Anpassung an die Größenverhältnisse und Auflösung des Displays. Alle verwendeten UI-Elemente - dazu zählen Buttons, Textfelder, etc. – sollen sich dynamisch an die Größe des Displays anpassen. Des Weiteren soll eine Unterscheidung zwischen Tablet und Smartphone stattfinden. Genauer gesagt soll für Tablets ein anderes Layout verwendet werden, als für Smartphones. Dies hat den einfachen Grund das Tablets weitaus mehr Platz zur Darstellung von Daten zur Verfügung stellen. Ab einer Displaygröße von sieben Zoll wird ein Gerät als Tablet betrachtet und das entsprechende Layout verwendet. Wie diese Layouts genau aufgebaut sind wird im Laufe der Arbeit näher erläutert.

3.3. Abgrenzungskriterien des Prototyps

Bei den Funktionalitäten der App wird sich auf die Umsetzung einzelner Grundfunktionalitäten beschränkt. Dazu zählen das Nachrichtensystem, der Gerätemanager, eine prototypische Einstellungsseite sowie eine Profilseite zur Änderung von Daten. Das Hauptaugenmerk der Entwicklung liegt bei der Realisierung der grafischen Benutzerschnittstelle, der Umsetzung der Tablet- und Smartphone-Unterscheidung sowie der Durchführung von Usability- und UI-Tests. Der Login von Nutzern wird umgesetzt, jedoch der automatisierte Login nicht. Die Schnittstelle zum externen Speichern und Laden der Daten wird in diesem Prototyp ebenso nicht umgesetzt werden, da diese zu diesem Zeitpunkt noch nicht vorhanden war. Um die Datenspeicherung jedoch zu Testzwecken zu gewährleisten wird mit einer lokalen Smartphone- bzw. Tablet-SQLite-Datenbank gearbeitet.

4. Theoretische Grundlagen

4.1. Android-Bedienkonzept

4.1.1. Activity

Die einfachste Variante der Darstellung von Layouts ist die Verwendung einer sogenannten Activity. Eine Activity füllt in der Regel das komplette Display aus und stellt das UI dar, in dem der Nutzer seine Eingaben tätigen kann. In der Regel besteht eine App aus einer Vielzahl von Activities. Diese sind aneinander gebunden, wobei eine als „Main Activity“ bezeichnet wird. Die Main Activity wird immer beim Start der App aufgerufen. Es ist möglich, neue Activities, zum Beispiel durch einen Button-Klick oder andere Nutzereingaben zu starten. Alle Activities werden im sogenannten „Back Stack“ abgelegt. Der Back Stack selbst basiert auf dem „Last in, First out“-Prinzip. Abbildung 4.1 verdeutlicht dieses Prinzip. [Andr13a]

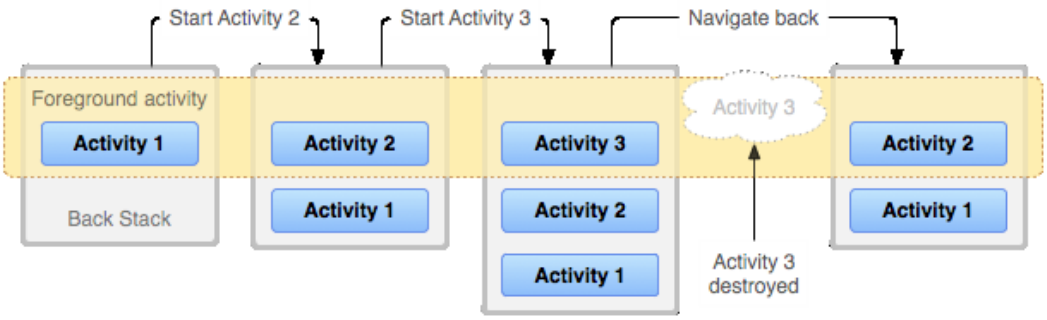

Abbildung 4.1: Back Stack Funktionsweise [Andr13b]

Beim Start einer neuen Activity rückt diese in den Vordergrund. Navigiert der Nutzer zurück, dann wird die aktuelle Activity zerstört und die vorhergehende Activity rückt zurück in den Vordergrund. Diese Activity erhält nun wieder den Fokus.

Im Folgenden wird der Lebenszyklus einer Activity näher erläutert. Die Abbildung 4.2 verdeutlicht diesen Vorgang. Der Lebenszyklus einer Activity findet zwischen dem Aufruf *onCreate()* und *onDestroy()* statt. Jedoch ist eine Activity nur zwischen den Aufrufen *onStart()* und *onStop()* für den Nutzer sichtbar. Während dieser Zeit nimmt die Activity Nutzereingaben entgegen. Des Weiteren gibt es noch den Zustand, in dem sich die Activity im Vordergrund befindet. Activities können jederzeit aus dem Vordergrund in den Hintergrund wechseln und umgekehrt. Ein Beispiel für das ständige Wechseln ist der Ruhemodus oder das Anzeigen von Dialogen. [Andr13a]

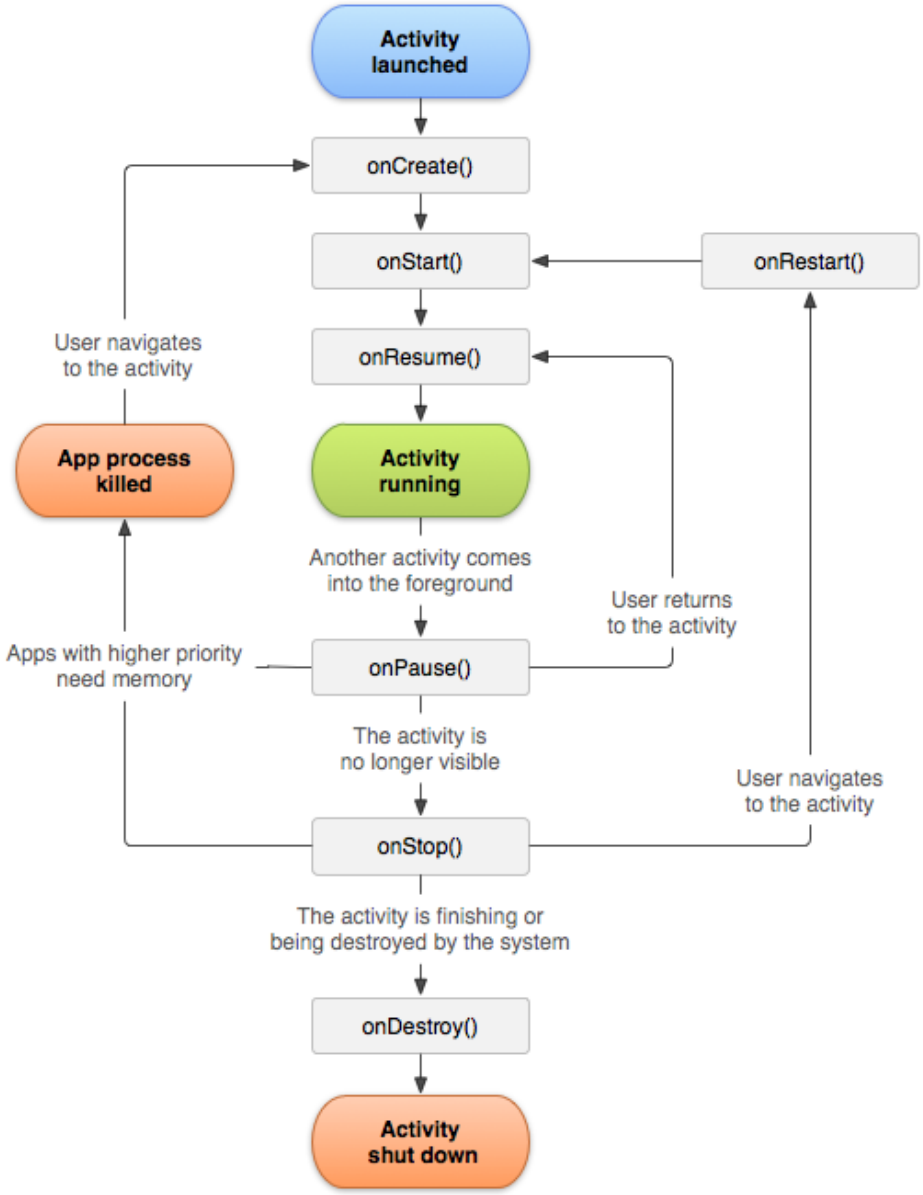

Abbildung 4.2: Activity Lebenszyklus [Andr13a]

Die Funktion *onCreate()* wird aufgerufen, wenn die Activity erstellt wird. Diese Funktion sollte in der Regel alle Operationen beinhalten, die das UI der Activity betreffen, denn hier wird dieses festgelegt. Die *onStart()*-Methode wird ausgeführt, kurz bevor die Activity für den Nutzer sichtbar wird. Auf die Methode *onStart()* kann entweder der Aufruf *onResume()* oder *onStop()* folgen, aber nur wenn der Zyklus komplett durchlaufen wird. *OnResume()* wird aufgerufen, wenn die Activity in den Vordergrund rückt bzw. *onStop()*, wenn sie in den Hintergrund rückt. Die *onPause()*-Methode wird ausgeführt, sobald eine andere Activity in den Vordergrund rückt. Die neue Activity erhält nun den oberen Platz des Back Stack und somit auch den Input Fokus. Die alte Activity rückt in den Hintergrund und rutscht im

Back Stack einen Platz nach unten. *OnPause()* wird hauptsächlich dazu verwendet, um persistente Daten beim Activity-Wechsel zu speichern, beispielsweise, wenn ein Nutzer ein Textfeld editiert und daraufhin eine neue Seite öffnet. Dieser möchte dann nach dem Zurücknavigieren nicht alle Daten neu eingeben müssen. Außerdem werden hier Vorgänge beendet, die eine bemerkbare Menge an CPU-Leistung verbrauchen. Diese Vorgänge könnten sonst das Öffnen der neuen Activity verzögern. Wichtig hierbei ist, das *onPause()* schnell durchlaufen wird, denn die nächste Methode kann erst ausgeführt werden, sobald *onPause()* einen Return-Wert liefert. *OnPause()* hat in der Regel *onStop()* zur Folge, aber in manchen Situationen wird direkt *onResume()* erneut ausgeführt, wenn die Activity zurück in den Vordergrund rückt. Während sich eine Activity in der *onStop()*-Methode befindet ist diese nicht mehr sichtbar, währenddessen wartet die Activity auf einen neuen Befehl. Dieser Befehl ist entweder der Neustart oder das Beenden der Activity. Die Folgemethoden sind also *onDestroy()* oder *onRestart().* Der Neustart der Activity wird mittels *onRestart()* durchgeführt. Diese wird ausgeführt, wenn der Nutzer zu einer alten Activity zurück navigiert und diese sich noch im Wartezustand befindet. Es folgt die Methode *onStart().* Mittels der in der Grafik nicht aufgeführten Methode *finish()* kann eine Activity beendet werden, dies hat unmittelbar *onDestroy()* zur Folge. *OnDestroy()* wird kurz vor dem Beenden der Activity aufgerufen und ist somit die letzte Möglichkeit, auf die Activity zuzugreifen. Das System hat die Möglichkeit, Activities im Zuge von Speichermangel selbst zu beenden. Diese Möglichkeit besteht während der Ausführung der *onPause()*-, *onStop()*- und *onDestroy()*-Methoden. Das heißt *onPause()* ist garantiert die letzte Methode, die ausgeführt wird, bevor die Activity beendet werden kann. Daher sollten persistente Daten in dieser Methode gespeichert werden. Wird eine gestoppte Activity beendet und der Nutzer navigiert dennoch zu ihr, dann wird diese neu gestartet und deren Lebenszyklus beginnt von neuem. [Andr13a]

4.1.2. Fragment

Mit der Android-Version 3.0 wurde das sogenannte Fragment eingeführt. Ein Fragment stellt einen Teil des UI einer einzelnen Activity dar. Dabei können mehrere Fragmente aneinandergefügt werden, um so ein mehrteiliges UI zu erzeugen. Die Fragmente innerhalb einer Activity sind dadurch immer an ihre Activity gebunden. Wird beispielsweise eine Activity pausiert, so werden auch alle darin befindlichen Fragmente pausiert. Dasselbe gilt, wenn die Activity geschlossen bzw. zerstört

wird. Der große Vorteil von Fragmenten ist es Interfaces und Layouts zu ermöglichen, die von Smartphones sowie Tablets verwendet werden, aber unterschiedlich dargestellt werden. Dieses Prinzip ist in der Abbildung 4.3 zu sehen. [Andr13c]

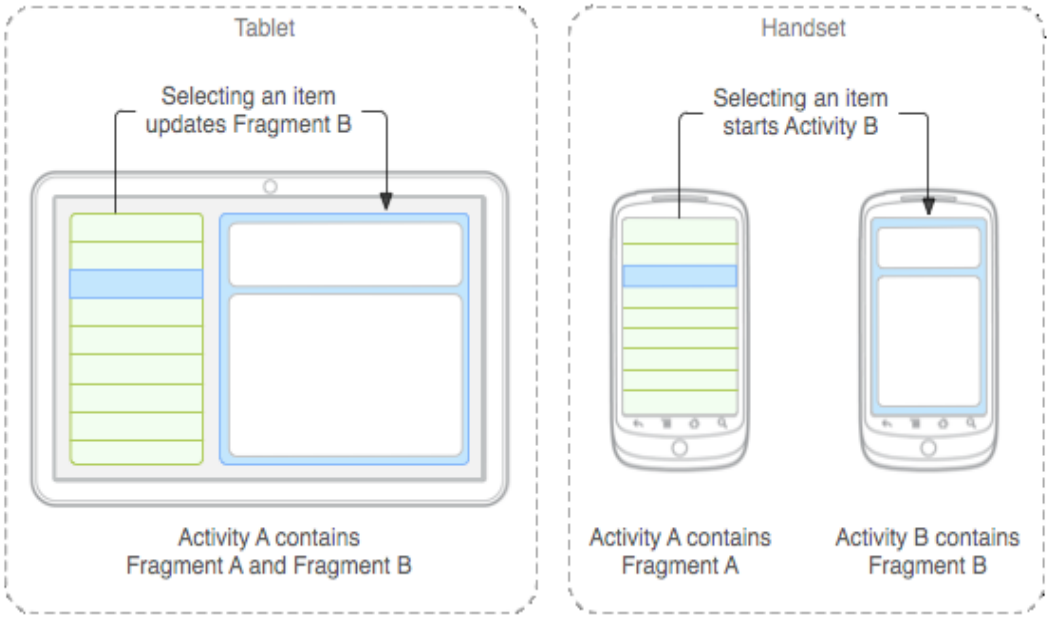

Abbildung 4.3: Fragmente-Layout [Andr13c]

Die Abbildung beinhaltet ein Layout, welches von Smartphones und Tablets unterschiedlich dargestellt wird. Auf einem Tablet beinhaltet eine Activity zwei Fragmente. Bei Auswahl eines Elements der Liste wird Fragment B im rechten Teil der App verändert, beispielsweise durch die Anzeige eines neuen Layouts. Ein Smartphone verfügt jedoch nicht über den Platz um dies umzusetzen. Daher werden hier zwei Activities anstatt einer eingesetzt, um dasselbe Ergebnis zu erzielen. Die Auswahl eines Listenitems startet eine neue Activity, die Fragment B mit ihrem Layout beinhaltet. [Andr13c]

Es besteht die Möglichkeit Fragmente mit einem sogenannten „View Pager“ zu verknüpfen. Ein View Pager ist eine Komponente, die es dem Nutzer ermöglicht Gesten zur Navigation zu verwenden, speziell ist damit das hin und her „Wischen“ (engl. Swipe) zwischen verschiedenen Seiten einer App gemeint. Um dafür einen fließenden Übergang von einer zur nächsten Seite zu ermöglichen, werden immer die aktuelle, die vorhergehende sowie die nachfolgende Seite geladen und aktiv gehalten. Abbildung 4.4 verdeutlicht dies an einem Beispiel. [Andr13d]

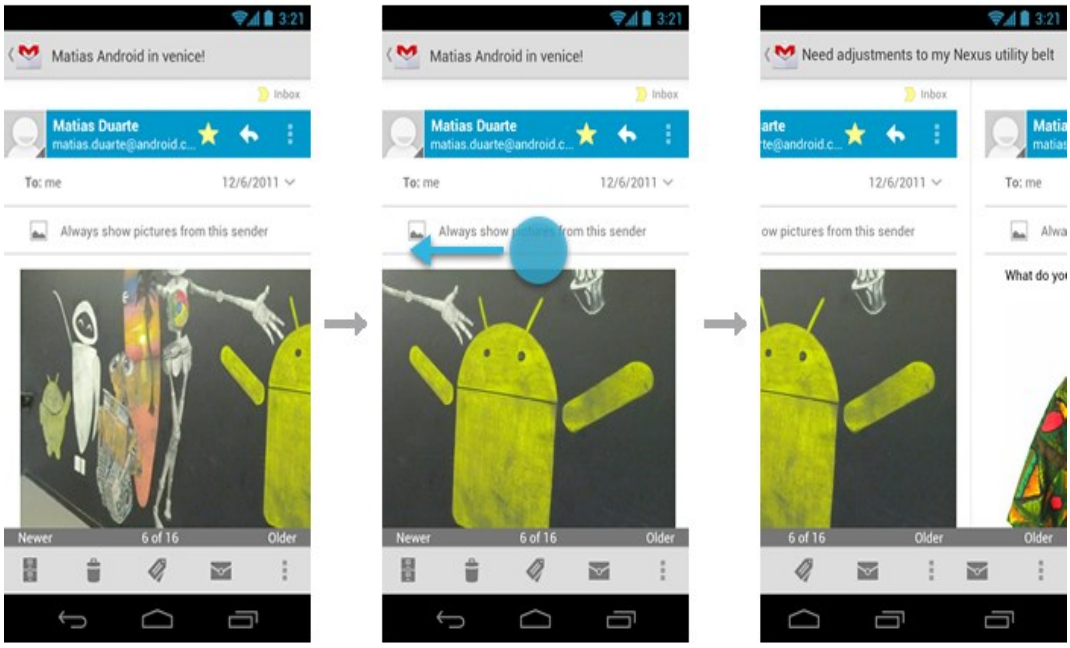

Abbildung 4.4: Navigation mittels Swipe [Andr13d]

Die Abbildung zeigt eine Swipe-Bewegung zur Navigation. Der Nutzer wischt mit seinen Fingern von der rechten Seite des Displays zur linken Seite. Dabei wird das Ausgangfragment langsam von dem zweiten Fragment verdrängt. Die Geschwindigkeit, mit der die Fragmente verschoben werden, ist dabei vom Tempo des Wischens abhängig.

4.1.3. Dialoge

Dialoge werden hauptsächlich genutzt, um Nutzereingaben anzufordern oder um zusätzliche Daten abzufragen. Diese Abfragen können von einfachen Ja-Nein-Dialogen bis hin zu Dialogen mit einem komplexen Layout reichen. Es gibt eine Vielzahl von vordefinierten Dialogen, wie zum Beispiel einen „DatePicker“ zur Abfrage eines Datums oder ein „ProgressDialog“ zur Darstellung eines Ladevorgangs. Der Grundaufbau ist hingegen immer gleich, jedoch kann das Layout für einen Dialog auch nach den eigenen Wünschen selbst erstellt werden. In Abbildung 4.5 ist der Grundaufbau dargestellt. [Andr13e]

Abbildung 4.5: Dialog Grundaufbau [Andr13e]

Die Bestandteile eines Dialoges (in Abbildung 4.5 markiert) sind ein Titel (1), welcher optional ist, ein Hauptbereich zur Darstellung von Inhalten (2) und bis zu drei Buttons zur Aktionsdurchführung (3). Diese Buttons sind ein positiver Button(Ok), ein negativer Button(Cancel) und ein neutraler Button. Der Entwickler hat hier Einfluss auf alle drei Bestandteile nehmen. Den Titel kann man nach Bedarf anzeigen und setzen. Der Hauptteil zur Inhaltsdarstellung kann nach Belieben selbst bestimmt werden. Die Buttons sind jedoch in ihrer logischen Reihenfolge festgelegt, d.h. der Positive-Button befindet sich immer rechts, der Neutral-Button mittig und der Negative-Button immer links. Der Entwickler kann hier lediglich Einfluss auf die Anzeige und die Funktion der Buttons nehmen. [Andr13e]

4.1.4. Action Bar

Eine Action Bar ist ein spezielles Element, welches sich in der Regel im oberen Bildschirmteil der App befindet und Nutzereingaben entgegen nimmt. Action Bars helfen dem Nutzer wichtige und häufig verwendete Funktionen schnell aufzurufen. Um dies zu ermöglichen werden Intuitive Icons verwendet, zum Beispiel eine Lupe für eine Suchanfrage oder ein Plus-Symbol um etwas hinzuzufügen. Des Weiteren wird dem Nutzer das Wechseln zwischen verschiedenen Ansichten einer Seite mittels eines Drop-Down Menüs, einer Tab Bar oder eines sogenannten View Controls erleichtert. Abbildung 4.6 zeigt diesen Aufbau noch einmal an einem Beispiel auf. [Andr13f]

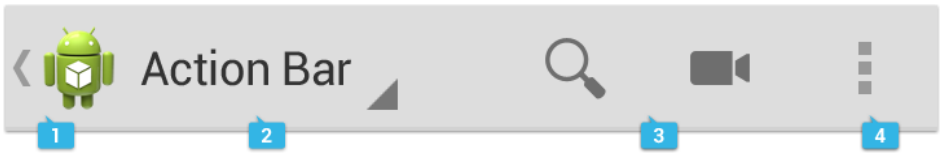

Abbildung 4.6: Aufbau Action Bar [Andr13f]

Der generelle Aufbau umfasst das App-Icon (1) auf der linken Seite, eine Anzahl von Action Buttons (3), abhängig von der Displaygröße sowie ggf. View Controls (2) und einen Action Overflow (4).
Das App-Icon kann, wie in Abbildung 4.7 dargestellt, entweder zur dekorativen Darstellung des Icons der App oder als Home-Button verwendet werden. Ein Home-Button ist ein Button, der auf eine bestimmte Hauptseite der App zurückverweist.

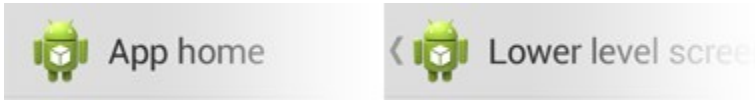

Abbildung 4.7: Action Bar – Home-Button [Andr13f]

Stellt die App Daten in verschiedenen Views dar, dann hilft ein View Control in der Action Bar dabei, zwischen diesen Views zu navigieren. In der Regel wird dies über Drop-Down Menüs (2 in Abbildung 4.6) realisiert. Falls kein Wechsel zwischen Views gewünscht ist, kann optional an dieser Stelle der App-Titel angezeigt werden. Der Platz für Aktionen in einer Action Bar ist begrenzt, daher sollte der Programmierer hier nur die am häufigsten verwendeten unterbringen. Alle weniger oft genutzten Aktionen können in einem Action Overflow (4 in Abbildung 4.6) abgelegt werden. Prinzipiell ist dies einfach ein Menü das alle übrigen Aktionen in einer Liste beinhaltet. Ist der vorhandene Platz in einer Action Bar verbraucht, dann werden alle übrigen, noch verbliebenen Aktionen automatisch im Action Overflow platziert. Das Overflow Icon wird aber nur auf Geräten angezeigt, die über keinen Hardware Menü-Button verfügen. Auf Geräten, die über einen Hardware Menü-Button verfügen, wird das Overflow-Menü erst angezeigt, wenn dieser Button betätigt wird. Es ist möglich den Inhalt einer Action Bar auf mehrere Action Bars aufzuteilen. Deshalb gibt es neben der Darstellung im Header noch zwei weitere Bereiche zur Darstellung einer Action Bar. Abbildung 4.8 zeigt diese Darstellungsmöglichkeiten genauer auf. [Andr13f]

Abbildung 4.8: Split Action Bar [Andr13f]

Die Main Action Bar beinhaltet mindestens das App-Icon und den Titel (1 in Abbildung 4.8). Des Weiteren können hier die wichtigsten Funktionen untergebracht werden. Die Top Bar (2 in Abbildung 4.8) besteht aus einem Control zur Steuerung von Views, in diesem Fall eine Tab Bar. In der Bottom Bar (3 in Abbildung 4.8) können alle weiteren Aktionen sowie der Action Overflow angezeigt werden. Die Darstellung mehrerer Action Bars ist vor allem interessant, wenn sich die Displaygröße bzw. die Ausrichtung des Smartphones ändert (Abbildung 4.9). [Andr13f]

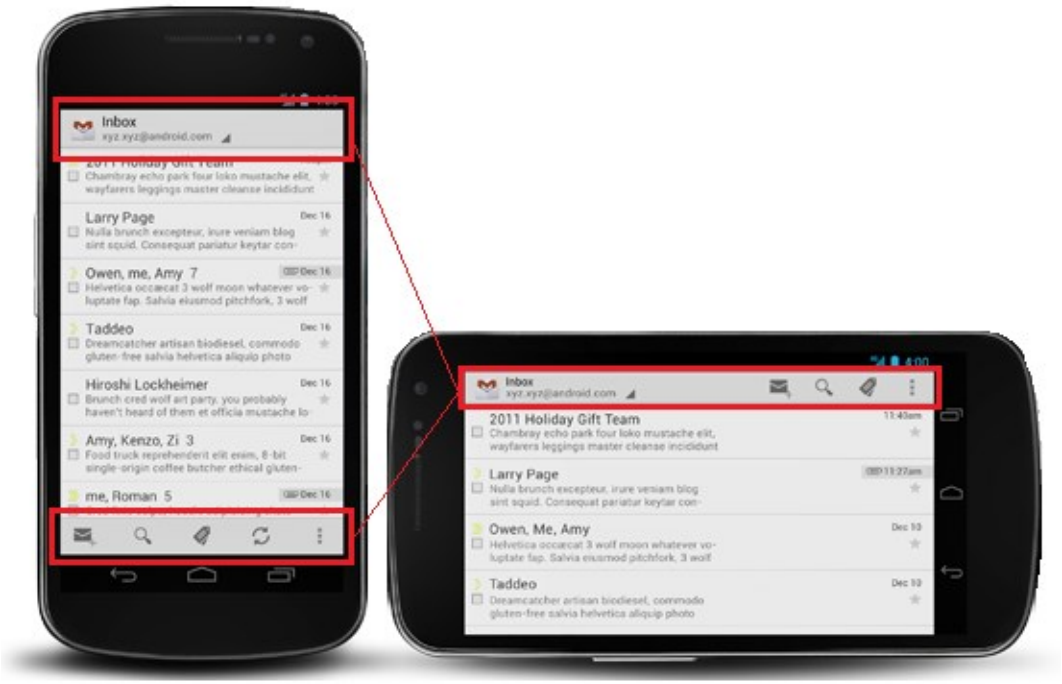

Abbildung 4.9: Action Bar – Orientation [Andr13f]

Wie in Abbildung 4.9 zu erkennen ist, wird bei der Queransicht die untere Action Bar aufgelöst und in der neuen, oberen Action Bar dargestellt. Um dies möglich zu machen wird die schon angesprochene Split Action Bar verwendet. Beim Drehen des Smartphones in die horizontale Ausrichtung wird die Bottom Bar aufgelöst, da die neue Main Action Bar genug Platz für alle Buttons beider Action Bars bietet. [Andr13f]

4.1.5. Smartphone vs. Tablet

Ein Hauptaugenmerk während der Entwicklung der App ist es, die Kompatibilität zwischen Smartphones und Tablets zu gewährleisten. In den folgenden Abbildungen ist diese Kompatibilität dargestellt. In Abbildung 4.10 ist das geplante Tablet-Layout dargestellt. Es verfügt über das bereits in Abschnitt 4.1.2 beschriebene Android-Grundkonzept zur UI-Darstellung für Tablets.

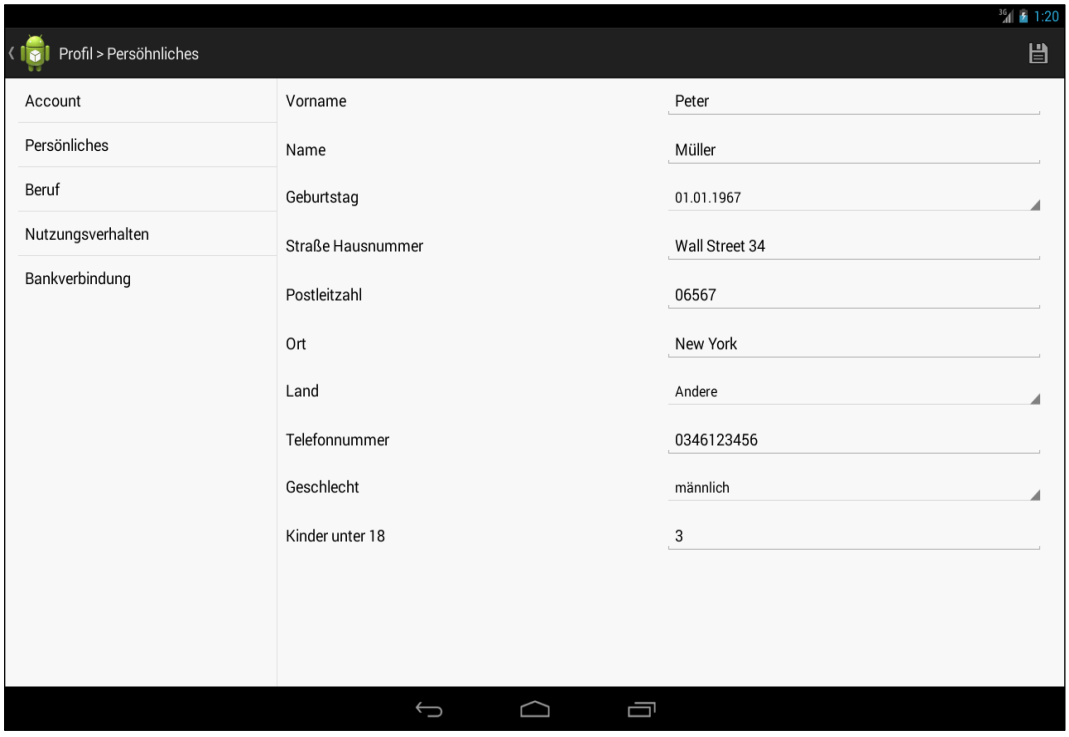

Abbildung 4.10: Tablet-Layout

Das Layout besteht aus zwei Fragmenten, einmal aus der Liste auf der linken Seite und einmal aus der Detailansicht auf der rechten Seite. Durch einen Klick auf ein Listenobjekt soll die zugehörige Detailseite im Fragment daneben angezeigt werden. Dieses Konzept soll, wenn möglich, in der ganzen App angewendet werden. Dies hat einerseits den Vorteil, dass der vorhandene Platz eines Tablet-Displays

optimal ausgenutzt werden kann, aber andererseits entsteht auch eine übersichtliche Darstellung der Daten. In Abbildung 4.11 ist das identische Layout, für einen Smartphone-Nutzer dargestellt.

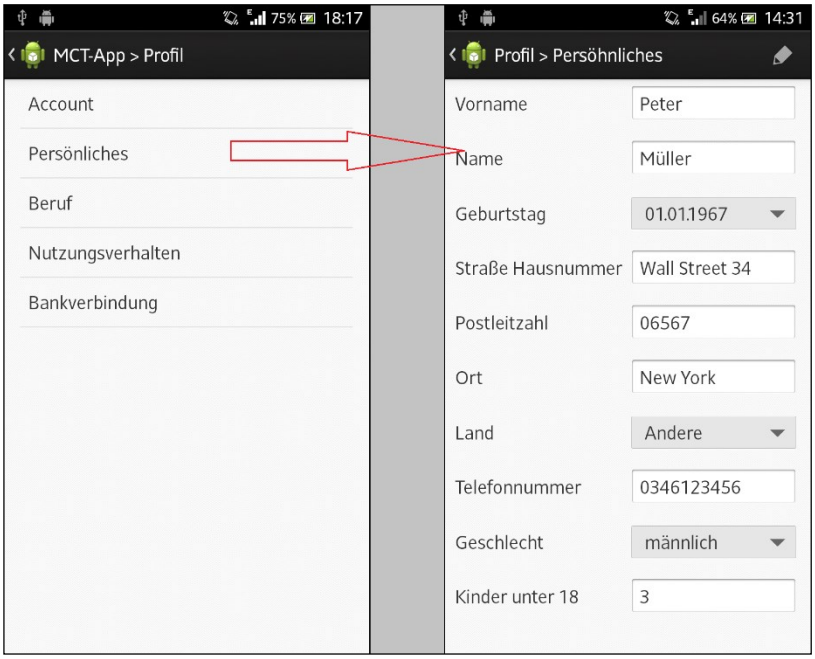

Abbildung 4.11:Smartphone-Layout

Wählt der Nutzer einen Unterpunkt aus der Liste aus, dann öffnet sich nun eine neue Activity mit der Detailansicht. Ein Layout, wie es für Tablets verwendet wird, wäre an dieser Stelle unsinnig, da hierfür der nötige Platz einfach nicht vorhanden ist. Die App soll automatisch erkennen, um was für eine Displaygröße es sich handelt. Das Tablet-Layout wird ab einer Displaygröße von sieben Zoll benutzt, ansonsten soll das Smartphone-Layout verwendet werden.

4.2. Planung der Umsetzung im Prototyp

Aufgrund der vielen Möglichkeiten der Navigation und der Bedienung wird versucht werden jedes Bedienkonzept in einer angemessenen Form in den Prototyp einzubauen. Eine Unterscheidung zwischen Tablet und Smartphone sowie eine dem entsprechende Anpassung der Layouts soll immer stattfinden. Im Smartphone-Layout wird für jede Seite eine Activity verwendet, d.h. es wird für jede neue Seite auch eine neue Activity erstellt. Für das Tablet-Layout wird das schon angesprochene Zwei-Fragmente-Layout verwendet. Dabei wird im kleineren, linken Teil immer eine Liste von Menüpunkten angezeigt und im größeren, rechten Teil die jeweils zugehörige Detailseite.

Dem Nutzer soll immer eine Action Bar angezeigt werden. In dieser soll der Titel der momentan angezeigten Seite dargestellt werden. Befindet sich der Nutzer nicht im Hauptmenü und nicht auf der Loginseite, dann wird das Icon der App im linken Teil als Home-Button verwendet. Der Home-Button verlinkt bei jedem Klick nun auf das Hauptmenü zurück. Sollte sich der Nutzer hingegen auf der Loginseite oder im Hauptmenü befinden, dann wird der Home-Button deaktiviert und es wird nur das Icon der App und der Titel der App angezeigt. Des Weiteren werden in der Action Bar Buttons zu finden sein, um Daten auf der entsprechenden Unterseite zu ändern, zu entfernen und hinzuzufügen. Diese sollen abhängig von der momentan geöffneten Seite unterschiedliche Icons und Funktionen haben. Um dies umzusetzen werden verschiedene Layouts für die Action Bar verwendet. Zusätzlich wird in der Action Bar der Nachrichtenseite ein View Control zu finden sein, welches dazu dient zwischen den empfangenen und den gesendeten Nachrichten hin und her zu wechseln. Um für den Nutzer die Dateneingabe zu vereinfachen werden Dialoge verwendet. Die Layouts dieser Dialoge wurden vorgefertigt und werden entsprechend der gewünschten Aktion aufgerufen. Die Dialoge sollen dem Nutzer so weit entgegenkommen, dass dieser nur noch wenige Aktionen durchführen muss, um das gewünschte Ergebnis zu erhalten. Für das Entfernen von Datensätzen wird daher eine Liste mit Checkboxen verwendet, in der der Nutzer einfach die zu entfernenden Items markiert und mit dem Betätigen des Ok-Buttons aus der Datenbank löscht. Ansonsten werden Dialoge mit Edit-Textfeldern benutzt, in denen der Nutzer die entsprechenden Daten eingibt oder Dropdown Menüs, in denen dem Nutzer eine Auswahl an möglichen Daten bereitgestellt wird. Des Weiteren wird es möglich sein die Geräteeigenschaften, wie zum Beispiel den Namen des Modelles, den Netzprovider oder den Hersteller des Gerätes automatisch auszulesen. Zusätzlich soll im Hauptmenü, wie in Abbildung 4.12 dargestellt, ein Zähler angezeigt werden, welcher die Anzahl der nicht gelesenen Nachrichten darstellt.

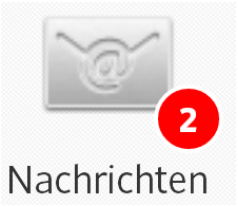

Abbildung 4.12: Nachrichtenzähler

Diese Darstellung wird durch ein zusätzliches Relative-Layout erreicht, welches aus einem ImageView und einem zusätzlichen TextView besteht, das einen speziellen Hintergrund erhält.

4.3. Automatisiertes UI-Testen auf mobilen Geräten

4.3.1. UI-Testing

UI-Testing gewährleistet, dass die App das korrekte UI widergibt. Dazu zählen unter anderem Nutzerinteraktionen, Menüs, Dialoge, sowie alle anderen UI-Komponenten. Funktionelles oder "Black-Box" UI-Testing erfordert kein Hintergrundwissen der Tester über die interne Implementierung der App. Einzig der erwartete Output muss ihnen bekannt sein, zum Beispiel wenn ein Nutzer eine spezielle Aktion durchführt oder einen spezifischen Input tätigt. Eine häufige Herangehensweise an das UI-Testing ist es, die App manuell auszuführen und das erwartete Verhalten zu verifizieren. Allerdings kann diese Methode sehr zeitaufwändig, langwierig und fehleranfällig sein. Effizienter ist es, die Software mit einem UI-Testing Framework zu testen. Automatisiertes Testen beinhaltet die Erstellung von Programmen, die den Test durchführen und Testberichte erstellen. Dabei werden spezifische Szenarien abgedeckt und alle Testfälle werden automatisiert durch das Framework durchgeführt. [Andr13g]
Im Rahmen des automatisierten UI-Testing auf Android Geräten gibt es zwei weitverbreitete Frameworks zur Testfallerstellung und Auswertung. Diese sind das Android Developer Framework (ADF) und das Robotium Framework.

4.3.2. Android Developer Framework

Das ADF setzt sich aus zwei Tools zusammen und wird im Android-SDK von Haus aus mitgeliefert. Diese Tools sind der „uiatomatorviewer“ (UAV) und der „uiautomator“ (UIA). Der UAV ist für das Scannen und Analysieren der GUI-Komponenten einer Applikation zuständig. Dieser stellt detailliert dar, wie das UI der ausgewählten Activity aufgebaut ist und über welche Eigenschaften die einzelnen Komponenten verfügen. Ein großer Nachteil des ADF ist es, das zur Ausführung der automatisierten Testfälle mindestens Android-Version 4.1 vorhanden sein muss. Weiterhin ist mindestens die SDK-Version 21 nötig, um Testfälle erstellen und durchführen zu können. In Abbildung 4.13 ist ein Beispiel des analysierten UI des Prototyps zu finden. [Andr13g]

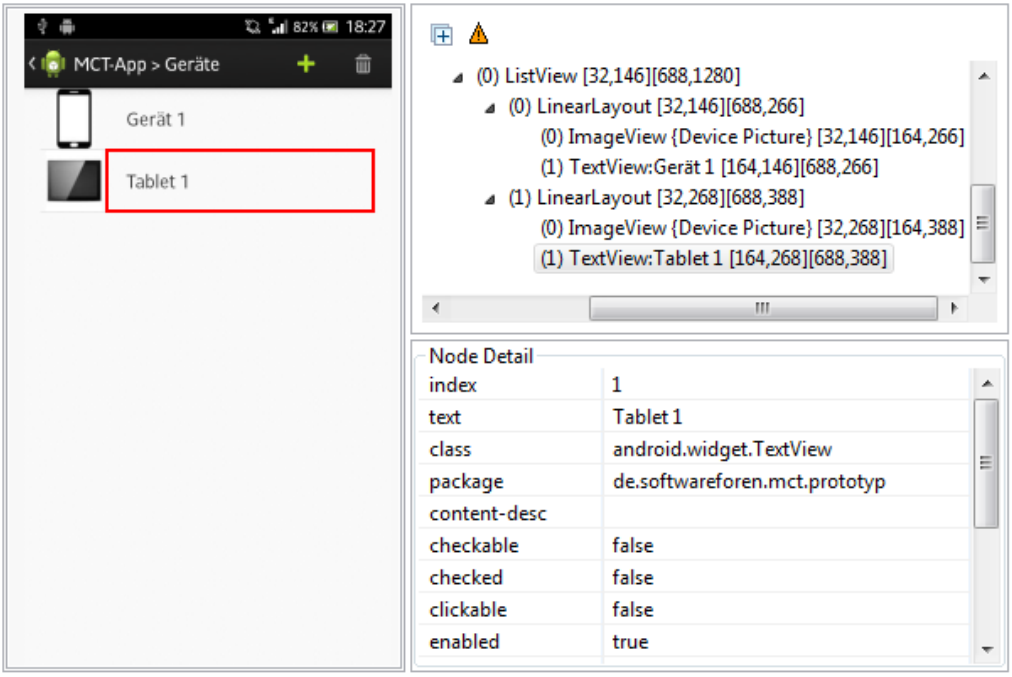

Abbildung 4.13: UAV-Beispiel

Die analysierte Activity (linker Teil in Abbildung 4.13) verfügt über die angegebene Layout-Struktur (oberer rechter Teil in Abbildung 4.13). Die ausgewählte UI-Komponente, in diesem Fall das TextView, hat die aufgeführten Eigenschaften (unterer rechter Teil in Abbildung 4.13).
Das zweite Tool, der UIA, besteht aus einer Ausführungsengine, um Tests auszuführen und zu automatisieren sowie einer Java-Bibliothek, um eigene funktionelle UI-Tests zu erstellen. Im folgenden Listing 1 ist Beispielcode für das ADF zu finden, daran soll der generelle, programmtechnische Aufbau eines automatisierten Testfalls aufgezeigt werden. [Andr13g]

```
1     // Ok & Cancel-Button erkennen
2     UiObject cancelButton = new UiObject (new UiSelector
                                  ().text("Cancel"));
3     UiObject okButton = new UiObject (new UiSelector ().text("OK"));

4     if (okButton.exists() && okButton.isEnabled())
5     {
6             // Ok-Button klicken
7             okButton.click();
8     }
```

Listing 1: UIA Beispielcode [Andr13g]

In diesem Beispielcode wird ein Button mit dem Text „Cancel“ und ein Button mit dem Text „OK“ im UI gesucht (Zeile 2 und 3). Falls der Ok-Button existiert und falls dieser aktiviert ist, dann wird er betätigt (Zeile 4 - 8).

Der Ablauf eines automatisierten Tests mit Hilfe des ADF durchläuft in der Regel die folgenden Schritte. Zur Vorbereitung wird die zu testende App auf dem Zielgerät installiert und mittels des UAV analysiert. Des Weiteren wird sichergestellt, dass das Automatisierungsframework Zugang zur Applikation hat. Im nächsten Schritt wird der automatisierte Test erstellt. Dieser beinhaltet die Nutzerinteraktionen, welche simuliert werden sollen. Daraufhin wird der Testfall in eine .jar-Datei kompiliert und zusammen mit der App auf dem Testgerät installiert. Als nächstes wird der erstellte Testfall automatisiert vom ADF durchgeführt. Sollten Fehler auftreten wird dies dem Tester in Form von Exceptiones durch das Framework mitgeteilt. Zum Schluss korrigiert der Tester wenn nötig seinen Prototyp anhand des Fehlerberichtes des ADF. [Andr13g]

4.3.3. Robotium Framework

Das zweite, große UI-Testing Framework ist ein Open Source Projekt und nennt sich Robotium. Im Gegenzug zum ADF ist Robotium nicht im Standard-SDK enthalten und muss extern auf der Entwicklerseite heruntergeladen werden. Der Download beschränkt sich dabei auf eine .jar-Datei. Diese muss lediglich in das Testprojekt eingebunden werden, um Zugriff auf die Programmierschnittstelle (API) von Robotium zu erhalten. [Reda13]
Über eine Art UAV, wie im ADF, verfügt Robotium jedoch nicht. Robotium wurde hauptsächlich entwickelt, um automatisierte Black-Box Tests von Android-Applikationen zu erstellen, welche wesentlich leichter und schneller zu erstellen und auszuführen sind als die des ADF [Sigg13]. Die Erstellung von Testfällen mit Hilfe von Robotium ist in der Tat schneller und einfacher. Die Funktionen der API sind außerdem intuitiver als im ADF. Im Listing 2 ist kurz etwas Beispielcode zu finden.

```
1     private Solo solo;
2     public void testDisplayBlackBox() {
3             // Button mit Text “Login” klicken
4             solo.clickOnButton("Login");
5             // Add Button suchen und klicken
6             solo.clickOnView(solo
                      .getView(de.softwareforen.mct.prototyp.R.id.add));
7             // „Test“ in das 1. Editfeld schreiben
8             solo.typeText(0, "Test");
9     }
```

Listing 2: Robotium Beispielcode

Die darin auftretende Solo-Klasse (Zeile 1) ist dabei die Hauptklasse, des Testframeworks. Diese beinhaltet alle Funktionen, die zum Testen des Interfaces benötigt werden (Zeile 4, 6 und 8). Ein großer Vorteil von Robotium ist die Unterstützung aller Android-Versionen ab Version 1.6. Der allgemeine Testablauf ist derselbe, wie der des ADF, mit der Ausnahme des nicht vorhandenen UAV.

5. Prototypische Realisierung

5.1. Aufbau der App

5.1.1. Login

Der Login erfolgt mittels eines Usernamen und dem dazugehörigen Passwort. Aus dem Passwort wurde bei seiner Erstellung mittels SHA-512 ein Hash erstellt und so in der Datenbank gespeichert. Da dies eine Einwegverschlüsselung ist, wird beim Login aus dem eingegebenen Passwort ebenso ein Hash erstellt. Daraufhin wird dieser Hash mit dem in der Datenbank abgeglichen. Bei der richtigen Username – Passwort Kombination wird dem Nutzer der Login gewährt. Es wird ein Toast mit der Beschriftung „Login erfolgreich“ angezeigt und seine entsprechenden Daten werden aus der Datenbank geladen. Bei einem falsch eingegeben Passwort oder einem ungültigen Usernamen wird hingegen ein Toast mit der Beschriftung „Login fehlgeschlagen“ angezeigt. Als Toast wird in Android das kleine, schwarze Nachrichtenfenster aus Abbildung 5.1 bezeichnet.

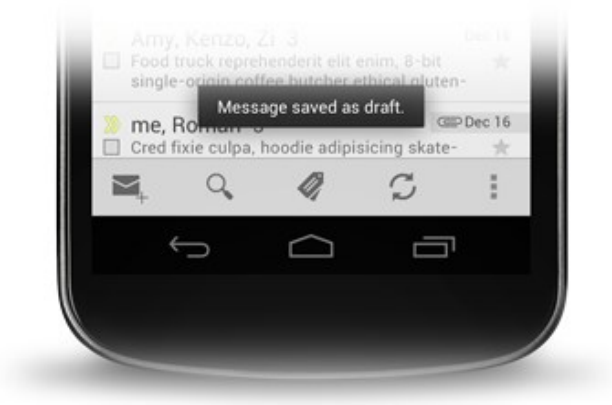

Abbildung 5.1: Toast Beispiel [Andr13h]

Toast werden hauptsächlich dazu benutzt, um dem Nutzer Feedback in Form einer kurzen Nachricht zu geben. Die momentan geöffnete Activity bleibt bei dessen Anzeige sichtbar und interaktiv. Der Nutzer kann also wie gewohnt seine App weiter bedienen. [Andr13h]

5.1.2. Nutzerdaten beziehen

Nach dem erfolgreichen Login werden alle Daten des Nutzers aus der Datenbank geladen und in eine Instanz der User-Klasse geschrieben. Diese Klasse beinhaltet alle Daten des Nutzers, die später dargestellt und verändert werden können. Der Aufbau dieser Klasse ist im folgenden Listing 3 zu finden.

```
1     public class User {
2             private List<Message> _messagereceivedlist;
3             private List<Message> _messagesendlist;
4             private List<Device> _devicelist;
5             private Account _user;
6     }
7     public class Account {
8             private String _username;
9             private String _password;
10            private String _email;
11            private String _firstname;
12            private String _lastname;
13            private String _birthday;
14            private String _street;
15            private int _housenumber;
16            private String _postcode;
17            private String _city;
18            private String _country;
19            private String _phonenumber;
20            private String _gender;
21            private int _childs;
22    }
```

Listing 3: User- und Account-Klasse

Alle Nachrichten und Geräte des Nutzers werden in Listen abgespeichert (Zeile 2, 3 und 4). Nachrichten werden noch einmal zwischen empfangenen oder gesendeten Nachrichten unterschieden. Dies hat den Grund, dass für deren Darstellung in den anschaulichen Listen ein Adapter gesetzt werden muss, welcher aus allen darzustellenden Datensätzen und einem vorgefertigten Layout bestehen muss. Eine Liste ist hierfür optimal geeignet, denn im Gegensatz zu Arrays können Listen flexible Mengen an Objekten enthalten und müssen nicht mit einer speziellen Größe initialisiert werden [Java13a]. Aber um Datensätze in Listen abzulegen kann kein Standard-Adapter verwendet werden, denn diese unterstützen nur Datensätze, die aus einem einzelnen String bestehen. Daher muss für die Nachrichten- und Geräteliste ein eigener List-Adapter entwickelt werden. Auf den genauen Aufbau dieser Adapter und Klassen wird im Abschnitt 5.1.3 eingegangen werden. Alle persönlichen Daten des Nutzers werden in einer separaten Account-Klasse gespeichert. Diese Daten umfassen:

- Username (Zeile 8),

- Passwort (Zeile 9),
- Email (Zeile 10),
- vollständigen Namen (Zeile 11 und 12),
- Geburtsdatum (Zeile 13),
- Anschrift (Zeile 14 – 17),
- Herkunftsland (Zeile 18),
- Telefonnummer (Zeile 19),
- Geschlecht (Zeile 20),
- Anzahl der Kinder (Zeile 21).

Das folgende Listing 4 zeigt beispielhaft die Abfrage aller gewünschten Nachrichten aus der Datenbank.

```
1     public List<Message> getAllMessagesSend(String username) {
2             List<Message> MessagesList = new ArrayList<Message>();
3             SQLiteDatabase db = this.getWritableDatabase();
4             Cursor cursor = db.query(TABLE_MESSAGES, new String[] {
                      KEY_ID_MESSAGE, KEY_SENDER, KEY_RECEIVER,
                      KEY_SUBJECT, KEY_DATE, KEY_CONTENT, KEY_READ },
                      KEY_SENDER + "=?", new String[] { username }, null,
                      null, null, null);
5             // looping through all rows and adding to list
6             if (cursor.moveToFirst()) {
7                     do {
8                             Message now = new Message();
9                             now.setID(Integer.parseInt(
                                      cursor.getString(0)));
10                            now.setSender(cursor.getString(1));
11                            now.setReceiver(cursor.getString(2));
12                            now.setSubject(cursor.getString(3));
13                            now.setDate(cursor.getString(4));
14                            now.setContent(cursor.getString(5));
15                            now.setRead(cursor.getInt(6));
16                            // Adding Message to list
17                            MessagesList.add(now);
18                    } while (cursor.moveToNext());
19            }
20            db.close();
21            // return Message list
22            return MessagesList;
23    }
```

Listing 4: Beispiel DB-Abfrage

Die Datenbank-Query aus Listing 4 (Zeile 4) ist im Prinzip eine einfache SQL-Select Abfrage. Die Query führt im Grunde die folgende SQL-Anweisung durch „SELECT * FROM Messages WHERE sender='username'". Die Ergebnisse der Query werden in einem sogenannten Cursor (Zeile 4) zwischen gespeichert. In der If-Abfrage (Zeile 11) wird überprüft, ob der Cursor Daten enthält. Sind Daten vorhanden, dann werden diese Reihe für Reihe in einer Do-While-Schleife (Zeile 8 – 19)

zur Nachrichtenliste (Zeile 2 und 18) hinzugefügt. Zum Schluss wird die fertige Nachrichtenliste zurückgeliefert (Zeile 23). Die Abfragen für alle anderen Tabellen unterscheiden sich nur minimal von der angegebenen.

5.1.3. Aufbau Listen

Wie schon in Abschnitt 5.1.2 angesprochen musste zur Darstellung der Nachrichten- und Geräteliste ein eigener List-Adapter entwickelt werden. Der generelle Aufbau dieses Adapters ist in Listing 5 zu finden.

```
1     public class MessageAdapter extends BaseAdapter {
2             private Context;
3             private List<Message> data;
4             private int selectednav;
5             static class MessageHolder {
6                     ImageView imgIcon;
7                     TextView txtName;
8                     TextView txtSubject;
9                     TextView txtDate;
10            }
11            public View getView(int position, View convertView,
                                  ViewGroup parent) {
12                    MessageHolder holder = new MessageHolder();
13                    Message = data.get(position);
14                    LayoutInflater inflater = (LayoutInflater) context
                              .getSystemService(Context
                              .LAYOUT_INFLATER_SERVICE);
15                    View row = inflater.inflate(
                              R.layout.message_listview_layout, null);
16                    holder.imgIcon = (ImageView)
                              row.findViewById(R.id.imgIcon);
17                    holder.txtName = (TextView)
                              row.findViewById(R.id.txtName);
18                    holder.txtSubject = (TextView)
                              row.findViewById(R.id.txtSubject);
19                    holder.txtDate = (TextView)
                              row.findViewById(R.id.txtDate);

20                    holder.imgIcon.setImageResource(
                              R.drawable.ic_launcher);
21                    holder.txtName.setText(message.getSender());
22                    holder.txtSubject.setText(message.getSubject());
23                    holder.txtDate.setText(message.getDate());

24                    return row;
25            }
26    }
```

Listing 5: List-Adapter

Der Base-Adapter (Zeile 1) ist die Basis-Klasse zur Implementierung von Adaptern. Der Typ Context (Zeile 2) spezifiziert die Umgebung der App näher, genauer gesagt ist es die aktuell geöffnete Activity. Selectednav (Zeile 4) ist im Grunde nur

ein Indikator, der aussagt, welche Nachrichtenliste angezeigt werden soll, also entweder alle empfangenen Nachrichten oder alle gesendeten Nachrichten. Die Liste (Zeile 3) enthält alle darzustellenden Datensätze, welche im Vorfeld aus der Datenbank bezogen wurden. Die Klasse MessageHolder (Zeile 5 – 10) beinhaltet die UI-Komponenten die zur Darstellung der Liste nötig sind. Mit Hilfe der *getView()*-Methode (Zeile 11 – 24) wird Zeile für Zeile die Liste aufgebaut. Zuerst wird die dafür benötigte View aus einem Layout erzeugt (Zeile 14 und 15). Anschließend wird die erzeugte View mit der MessageHolder-Klasse verknüpft (Zeile 12, 16 – 19). Zum Schluss werden die UI-Komponenten mit den Daten aus der Datenbank gefüllt (Zeile 13, 20 – 24). Dieser Vorgang wird Zeile für Zeile der Liste wiederholt, bis alle Datensätze verarbeitet wurden. Die Adapter-Klasse erstellt also bei deren Initialisierung entsprechend der Datensätze die visuelle Auflistung, wie in Abbildung 5.2 dargestellt.

Abbildung 5.2: Nachrichtenliste

Das angezeigte Android-Icon in jeder Zeile der Liste steht für einen Avatar des jeweiligen Nutzers. Dieser wurde jedoch in diesem Prototyp nicht implementiert. Eine Nachricht wurde wie folgt definiert (Listing 6).

```
1     public class Message {
2             private int _id;
3             private String _sender;
4             private String _receiver;
5             private String _subject;
6             private String _date;
7             private String _content;
8             private int _read;
9     }
```

Listing 6: Nachrichten-Klasse

Eine Nachricht besteht also aus den folgenden Komponenten:

- ihre eindeutige ID (Zeile 2),
- dem Sender (Zeile 3),

- dem Empfänger (Zeile 4),
- einem Betreff (Zeile 5),
- dem Verfassungsdatum (Zeile 6),
- der Inhalt der Nachricht (Zeile 7),
- einem Status (Zeile 8), der beschreibt ob die Nachricht schon gelesen wurde oder nicht.

Die Listen werden für den Nutzer jedoch erst sichtbar, wenn der zugehörige List-Adapter für die zugehörige Komponente gesetzt wird. Für die Geräteliste wird dasselbe Prinzip angewendet, nur mit dem Unterschied, dass es sich um ein anderes vorgefertigtes Layout und andere Daten handelt. Der Vollständigkeit halber ist in Listing 7 die Definition eines Gerätes zu finden.

```
1     public class Device {
2             private int _id;
3             private String _owner;
4             private String _typ;
5             private String _name;
6             private String _manufactor;
7             private String _model;
8             private String _os;
9             private String _version;
10            private String _provider;
11    }
```

Listing 7: Geräte-Klasse

Ein Gerät hat demnach die folgenden Eigenschaften:

- eine eindeutige ID (Zeile 2),
- einen Besitzer (Zeile 3),
- einen Typ (Smartphone oder Tablet) (Zeile 4),
- einen selbstvergebenen Namen (nicht der Modellname) (Zeile 5),
- einen Hersteller (Zeile 6).
- ein Modellname (Zeile 7),
- ein Betriebssystem (Zeile 8),
- die Version des Betriebssystems (Zeile 9),
- der Netzprovider (Zeile 10).

Der List-Adapter für die dazugehörige Geräteliste hat einen ähnlichen Aufbau wie der einer Nachricht. Der Aufbau stimmt insoweit mit dem aus Listing 5 überein, lediglich die Datenquelle und die Darstellung sind eine andere.

5.1.4. Aufbau der Hauptseiten

Nach dem Login und dem Laden der Daten wird der Nutzer zum Hauptmenü weitergeleitet, über welches er Zugriff auf alle Funktionen der App hat. Auf dem Nachrichten-Button werden außerdem die Anzahl der ungelesenen Nachrichten angezeigt. Der Prototyp beinhaltet jedoch noch nicht alle geplanten Features. Bis zu diesem Zeitpunkt wurde das Nachrichtensystem, der Gerätemanager sowie die Account- und Profilseite implementiert. Nachfolgend wird auf all diese Seiten genauer eingegangen und deren grundsätzlicher Aufbau näher erläutert. Die angezeigten Daten wurden im Vorfeld aus der Datenbank geladen und wie in 5.1.2 beschrieben in eine Instanz der User-Klasse abgelegt.
Die Profilseite besteht aus einer Liste zur Navigation und zwei Unterseiten (Abbildung 5.3).

Abbildung 5.3: Profilseite

Die Unterseiten „Beruf“, „Nutzungsverhalten“ und „Bankverbindung“ wurden in diesem Prototyp noch nicht implementiert. Die Unterseite Account, wie in Abbildung 5.4 zu sehen, zeigt dem Nutzer seinen Usernamen und seine Email-Adresse.

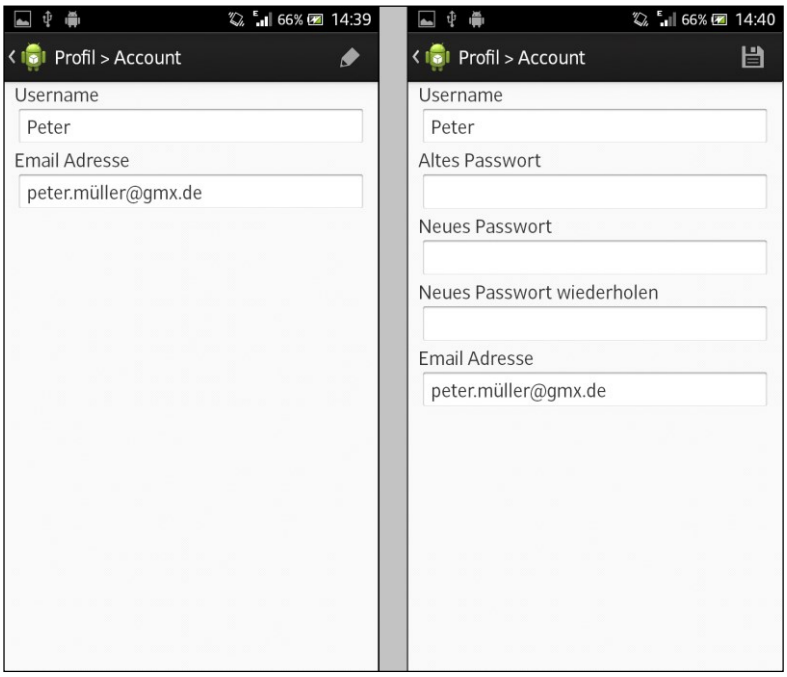

Abbildung 5.4: Accountseite

Betätigt der Nutzer den Edit-Button in der Action Bar, dann werden drei zusätzliche Eingabefelder sichtbar. Hier kann dieser nun sein Passwort ändern. Dafür muss zunächst das alte Passwort und ein neues Passwort eingegeben werden. Das neu eingegebene Passwort muss bestätigt werden, in dem es zwei Mal eingegeben wird. Betätigt der Nutzer den Button der Action Bar, welcher nun ein Speichern-Symbol hat, erneut, dann wird der Datenbankabgleich des Passwortes gestartet. Dazu wird aus dem eingegebenen, alten Passwort ein Hash erstellt und mit dem in der Datenbank abgelegten Hash verglichen. Stimmen diese sowie das doppelt eingegebene neue Passwort überein, dann wird das neue Passwort für diesen Account gesetzt. Die Unterseite, die die Profildaten des Nutzers enthält ist ähnlich, wie die Accountseite aufgebaut, sie ist in Abbildung 5.5 zu finden.

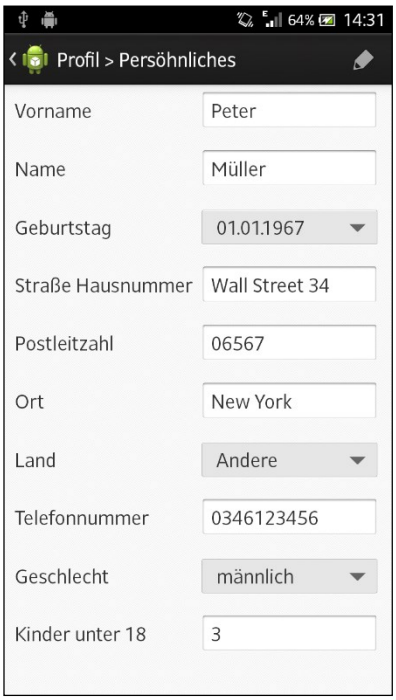

Abbildung 5.5: Profildaten

Es werden alle Daten aus der Datenbank in den Textfeldern angezeigt. Mittels des Bearbeitungs-Buttons in der Action Bar können diese verändert werden. Hierfür ist jedoch keine Bestätigung notwendig. Die Daten werden nach Betätigung des Speichern-Buttons sofort in der Datenbank abgespeichert.
Nachfolgend wird das Nachrichtensystem und die Darstellung der Nachrichten genauer erklärt werden. In Abbildung 5.6 ist ein Überblick über das Nachrichtensystem zu finden.

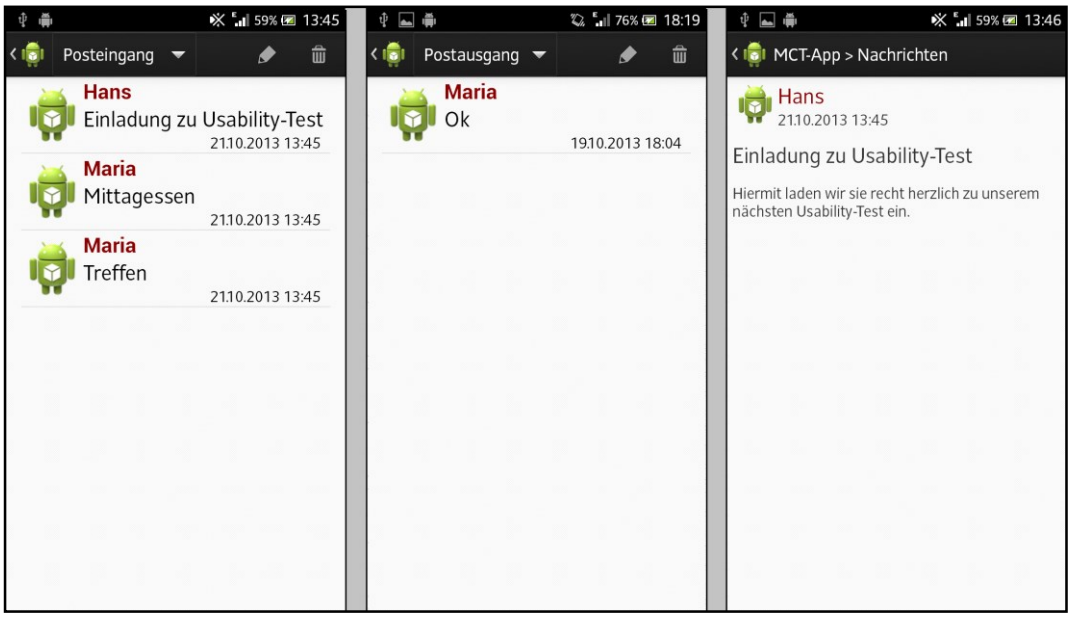

Abbildung 5.6: Nachrichtenliste

Wie bereits in Abschnitt 5.1.2 angesprochen, werden die Nachrichten in zwei verschiedene Listen unterteilt. Diese Unterteilung dient der einfacheren Darstellung für den Nutzer. Ihm kann nun eine Liste von empfangenen oder eine von gesendeten Nachrichten angezeigt werden. Der Benutzer wechselt mittels des View Controls in der Action Bar zwischen seinen empfangenen und gesendeten Nachrichten hin und her. In Listing 8 ist der zugehörige Quellcode zum Wechseln der Listen zu finden.

```
1     public boolean onNavigationItemSelected(int position, long id) {
2             ListFragment lFrag;
3             List<Message> msgList;
4             if (MainActivity.isTablet() == true) {
5                     lFrag = (ListFragment)
                      getSupportFragmentManager().findFragmentById(
                      R.id.item_list_profile);
6             } else {
7                     lFrag = (ListFragment)
                      getSupportFragmentManager().findFragmentById(
                      R.id.item_list_message);
8             }
9             if (position == 1) {
10                    msgList = MainActivity.user.getMessageSendList();
11            } else {
12                    msgList =
                      MainActivity.user.getMessageReceivedList();
13            }
14            MessageAdapter adapter = new MessageAdapter(this, msgList,
                      position);
15            lFrag.setListAdapter(adapter);
16            return false;
17    }
```

Listing 8: View Control

Wie in Listing 8 zu erkennen ist, wird bei der Auswahl eines neuen Items, des View Controls, der List-Adapter des zugehörigen ListFragments immer wieder neu gesetzt (Zeile 14 und 15). Dadurch wird dem Nutzer immer eine aktualisierte Liste angezeigt. Die Komponente „ListFragment“ (Zeile 2) ist hier für die korrekte Darstellung der Layouts verantwortlich. Dafür wird zuerst überprüft, ob es sich bei dem Gerät um ein Smartphone oder ein Tablet handelt (Zeile 4). Entsprechend dem Ergebnis dieser Überprüfung wird das Tablet-Layout (Zeile 5) oder das Smartphone-Layout (Zeile 7) genutzt.

Will sich der Nutzer eine Nachricht genauer anschauen, dann wird mittels des Klicks auf eine Nachricht eine Detailansicht geöffnet. Sobald eine Nachricht geöffnet wurde, wird sie als „gelesen“ markiert. Dadurch kann dem Nutzer angezeigt werden, welche Nachrichten er schon gelesen hat und welche noch nicht. Des Weiteren wird dem Nutzer die Option angeboten neue Nachrichten zu schreiben

oder seine Nachrichten zu löschen. Um diese zu nutzen muss er lediglich den entsprechenden Button der Action Bar betätigen. Daraufhin öffnet sich der entsprechende Dialog und der Nutzer kann die gewünschte Aktion durchführen. Abbildung 5.7 enthält diese Dialoge.

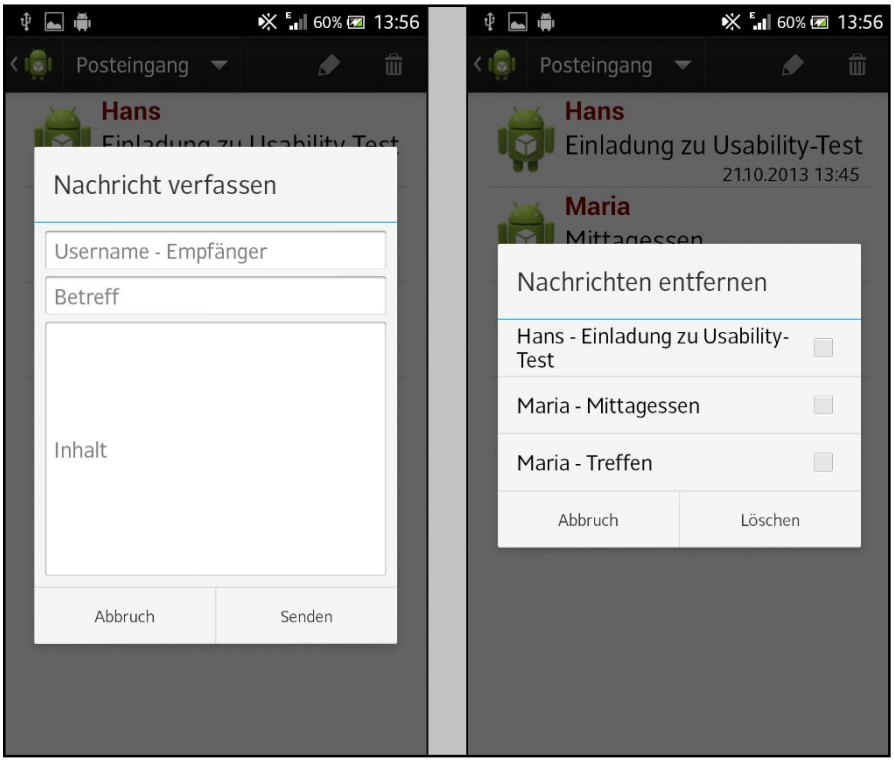

Abbildung 5.7: Nachrichten-Dialoge

Auf der linken Seite ist der Dialog zu finden, mit dem der Nutzer einem anderen Nutzer eine neue Nachricht senden kann. Die einzig nötigen Daten, die hier eingeben werden müssen sind der Empfängername, ein Betreff und natürlich der Inhalt der zu versendenden Nachricht. Bei Betätigen des Versenden-Buttons wird eine neue Instanz der Nachrichten-Klasse (Listing 6, Abschnitt 5.1.3) erstellt. Listing 9 beinhaltet den dazugehörigen Quellcode.

```
1     Message msg = new Message(username, ReceiverUser, Sbjct, Date,
      Content, 0);
2     db.addMessage(msg);
3     ListMsgSend.add(msg);
4     ListMsgSend.get(ListMsgSend.size() - 1)
              .setID(db.getMessage(MaxId).getID());
```

Listing 9: Nachrichteninstanz

Nachdem eine neue Instanz der Nachrichtenklasse erstellt wurde (Zeile 1), wird die Nachricht in die Datenbank und in die Liste der gesendeten Nachrichten übertragen (Zeile 2 und 3). Die Nachrichten ID der Liste muss jedoch manuell gesetzt

werden (Zeile 4), da diese beim Hinzufügen zur Liste nicht automatisch wie in der Datenbank gesetzt wird.
Im rechten Teil von Abbildung 5.7 ist der Dialog zum Löschen von einer oder mehrerer Nachrichten dargestellt. In Listing 10 ist außerdem der wichtige Teil des Quellcodes zum Löschen der Nachrichten zu finden.

```
1       private ArrayList<Integer> mSelectedItems = new
                ArrayList<Integer>();
2       Integer[] ToRemove = new Integer[mSelectedItems.size()];
3       for (int i = 0; i < mSelectedItems.size(); i++) {
4               ToRemove[i] = mSelectedItems.get(i);
5       }
6       Arrays.sort(ToRemove);
7       for (int h = mSelectedItems.size() - 1; h >= 0; h--)
8       {
9               db.deleteMessageID(
                        ListMsgReceived.get(ToRemove[h]).getID());
10              ListMsgReceived.remove(ToRemove[h]);
11      }
```

Listing 10: Nachrichten entfernen

Dem Nutzer wird eine einfache Liste aller Nachrichten der derzeit ausgewählten Kategorie angezeigt. Er markiert einfach die Nachrichten, die er löschen will und bestätigt seine Auswahl mit dem Löschen-Button. Alle ausgewählten Nachrichten werden in einer Integer-ArrayList (Zeile 1) gespeichert. Diese Liste beinhaltet die Position der ausgewählten Nachrichten in der Liste, angefangen bei 0. Listen haben jedoch den Nachteil, dass diese keine leeren Einträge, wie Arrays, erlauben. Dadurch kann beim Löschen der Listenelemente ein Fehler auftreten. Beispielsweise kann es passieren, dass Elemente gelöscht werden, welche gar nicht gelöscht werden sollten. Um diesen Fehler zu vermeiden findet eine Sortierung (Zeile 2 – 6) der ausgewählten Nachrichten sowie eine Löschung beginnend bei der größten Position statt (Zeile 7 – 11).
Die letzte Seite ist der Gerätemanager. Hier kann der Nutzer seine Geräte angeben, für die er später gerne Tests erhalten würde. Ein Überblick über den Manager ist in Abbildung 5.8 zu finden.

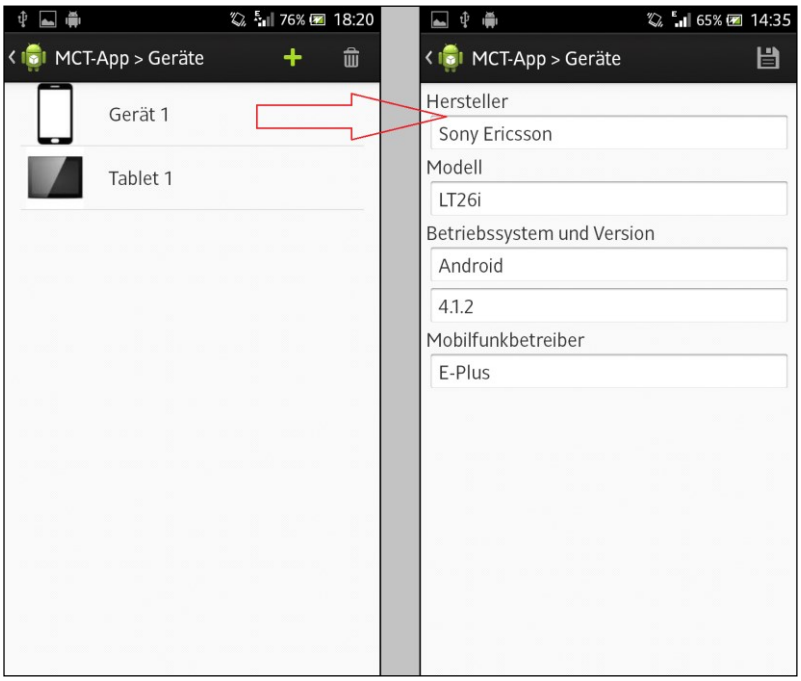

Abbildung 5.8: Gerätemanager

Die Detailseite eines Gerätes enthält alle technischen Daten des jeweiligen Gerätes. Diese können im Nachhinein über den Bearbeitungs-Button in der oberen rechten Ecke verändert werden. Die Action Bar der Geräteseite ist, abgesehen von dem fehlenden View Control, genauso wie die der Nachrichtenseite aufgebaut. Diese verfügt ebenso über einen Button zum Hinzufügen und einen zum Löschen von Geräten. Der Dialog zum Löschen unterscheidet sich nicht weiter, von dem der Nachrichtenseite. Der Dialog zum Hinzufügen von Geräten hingegen enthält eine automatische Erkennung des momentan verwendeten Gerätes. Diese kann der Nutzer auf Wunsch durchführen lassen. Abbildung 5.9 enthält den Aufbau dieses Dialoges.

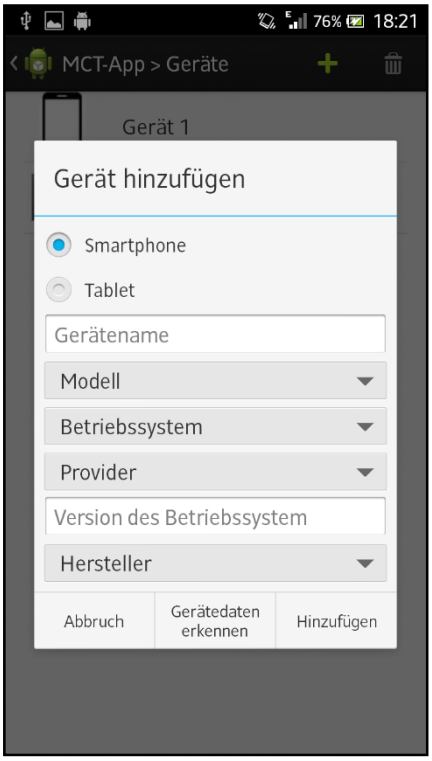

Abbildung 5.9: Gerät hinzufügen

Der Dialog enthält zwei Radio-Buttons zur Auswahl des Typs des Gerätes, zwei Textfelder für einen Namen des Gerätes und die Betriebssystemversion (Zeile 3) sowie vier Combo-Boxen zur Auswahl von Modell (Zeile 5), Betriebssystem, Netz-Provider (Zeile 1 und 2) und Hersteller (Zeile 4). Mit den in Listing 11 dargestellten Befehlen lassen sich die benötigten Systemdaten auslesen.

```
1    TelephonyManager manager = (TelephonyManager) Activity
            .getSystemService(Context.TELEPHONY_SERVICE);
2    manager.getNetworkOperatorName();
3    Build.VERSION.RELEASE;
4    android.os.Build.MANUFACTURER
5    android.os.Build.MODEL
```

Listing 11: Systembefehle

Neben den aufgeführten Eigenschaften ist es außerdem noch mögliche Daten wie SDK-Version, industrieller Name des Gerätes, Produktname oder Displaygröße auslesen. Prinzipiell lassen sich nahezu alle Hardware spezifischen Daten des Gerätes erfassen. Nach dem Betätigen des „Hinzufügen“-Buttons wird ähnlich wie in Listing 9 eine neue Geräteinstanz erstellt und in Datenbank sowie Liste abgespeichert.

5.1.5. Unterschiede der Layouts

Nachfolgend wird auf die genaue Umsetzung des Smartphone- bzw. Tablet-Layouts eingegangen. Wie in Abschnitt 4.2 bereits erwähnt, wird für Smartphone-Nutzer ein gewöhnliches Layout, bestehend aus einer Activity genutzt. Fragmente werden hier nicht weiter verwendet. Der Seitenaufbau erfolgt immer mit Hilfe einer Activity für die jeweilige Seite. Tablets werden das in Kapitel 4.1.2 und 4.1.5 angesprochene Zwei-Fragmente-Layout nutzen. Nichtsdestotrotz muss in beiden Layouts der vorhandene Platz zur Darstellung optimal genutzt werden. Um dies zu realisieren wird die XML-Eigenschaft „Layout-Weight“ genutzt. Diese Eigenschaft sorgt dafür, dass die entsprechenden UI-Komponenten den vorhanden Platz in ihrem Layout im angegebenen Verhältnis unter sich aufteilen. Um dies so nutzen zu können müssen zwei Voraussetzungen vorhanden sein. Zum einen müssen sich die UI-Komponenten in einem Linear-Layout befinden und zum anderen muss dieses Layout sowie die vorhandenen Komponenten die gesamte verfügbare Breite oder Höhe einnehmen. Dafür muss die XML-Eigenschaft „Layout_Width“ (für die Breite) bzw. „Layout_Height“ (für die Höhe) den Wert „fill_parent“ annehmen, sprich es wird die komplett verfügbare Breite bzw. Höhe genutzt. Die folgende Abbildung 5.10 und das folgende Listing 12 zeigen diese Eigenschaften an einem Beispiel des Prototyps genauer auf.

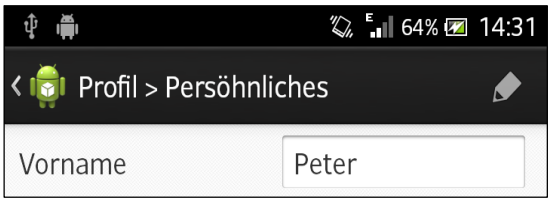

Abbildung 5.10: Layout_weight Beispiel

```
1           <LinearLayout
2               android:layout_width="fill_parent"
3               android:layout_height="wrap_content" >
4                <TextView
5                    android:id="@+id/textView1"
6                    android:layout_width="match_parent"
7                    android:layout_height="wrap_content"
8                    android:layout_weight="1"/>
9                <EditText
10                   android:id="@+id/editText1"
11                   android:layout_width="match_parent"
12                   android:layout_height="wrap_content"
13                   android:layout_weight="1"/>
14          </LinearLayout>
```

Listing 12: Eigenschaft - Layout_weight

Das in Abbildung 5.10 dargestellte Beispiel zeigt ein Linear-Layout, welches ein TextView und ein EditText beinhaltet. Das Linear-Layout nimmt dabei die volle Breite des Displays ein (Zeile 2). TextView und EditText nehmen auch die komplette verfügbare Breite ein (Zeile 7 und 13). Das Verhältnis der Breiten soll 1:1 betragen (Zeile 9 und 15), d.h. TextView und EditText nehmen beide 50% der Displaybreite ein. Diese Art der Darstellung wird in Layouts der Smartphone- und Tablet-App in verschiedenen Arten verwendet.

5.2. UI-Test

5.2.1. Testumgebung

Für die Programmierung der automatisierten Tests wurde Eclipse zusammen mit dem Android-SDK von Google benutzt. Als Testframework wird im weiteren Vorgehen Robotium benutzt. Dies liegt hauptsächlich an der Einfachheit der Programmierung sowie dem Erstellen von Testfällen. Des Weiteren verfügte das benutzte Testgerät während der Entwicklung noch nicht über die nötige Android-Version von 4.1, um das ADF verwenden zu können. Daher hätte zur Ausführung der Tests ein Android-Emulator benutzt werden müssen. Dieser ist aber einem echten Gerät leistungstechnisch sehr weit unterlegen und konnte deshalb nicht verwendet werden. Beispielsweise nimmt das einfache Öffnen einer neuen Activity schon mehrere Sekunden in Anspruch. Für die Durchführung des UI-Tests wurde ein automatisierter Testfall erstellt, der alle möglichen Anwendungsszenarien beinhaltet. Die in Abschnitt 5.2.2 aufgeführten Quellcodefragmente sind Ausschnitte aus diesem komplexen Testfall. Listing 13 beinhaltet den allgemeinen Grundaufbau eines Testfalles in Robotium.

```
1       private static final String
                LAUNCHER_ACTIVITY_FULL_CLASSNAME = "de.softwareforen.mct.
                prototyp.main.LoginActivity";
2       private static Class launcherActivityClass;
3       static {
4               try {
5                       launcherActivityClass = Class
                                .forName(LAUNCHER_ACTIVITY_FULL_CLASSNAME);
6               } catch (ClassNotFoundException e) {
7                       throw new RuntimeException(e);
8               }
9       }
10
11      private Solo solo;
12      public TestApk() throws ClassNotFoundException {
13              super(launcherActivityClass);
14      }
15      protected void setUp() throws Exception {
16              solo = new Solo(getInstrumentation(), getActivity());
17      }
18      public void TestThis() {
19              // Testfall
20      }
21      public void tearDown() throws Exception {
22              solo.finishOpenedActivities();
23      }
```

Listing 13: Robotium Testfallaufbau

Zuerst findet eine Überprüfung statt, ob die angegebene Start-Activity überhaupt gefunden werden kann (Zeile 1 - 9 und 12 -14). Ist dies der Fall wird in der *setUp()*-Methode die Solo-Klasse initialisiert (Zeile 15 - 17). Dabei werden ihr die Ausgangsactivity und der Paketname des Manifests der zu testenden App zugewiesen (Zeile 16). Dieses Manifest beinhaltet alle wichtigen Daten über die App. Dazu zählen unteranderem alle verwendeten Activies, App-Icon und die zulässigen Android-Versionen dieser Applikation. Im Prinzip findet hier also eine Verlinkung auf die zu testende App statt, in der der Test durchgeführt werden soll. Daraufhin wird mit der Abarbeitung des Testfalles begonnen (Zeile 18 - 20). Sobald dieser beendet wurde wird in der *tearDown()*-Methode abgesichert, dass alle Ressourcen wieder freigegeben werden, bevor ein neuer Testfall durchgeführt wird (Zeile 21 - 23).

5.2.2. Testfälle

Im Folgenden werden einige selbsterstellte Testfälle beschrieben und durch entsprechenden Quellcode für zwei Testfälle dargestellt. Im Anschluss wird das Ergebnis der Testfälle kurz vorgestellt und ausgewertet.

Testfall 1: Login - Listing 14

```
1     public void testDisplayBlackBox() {
2             // Edittextfelder leeren
3             solo.clearEditText(0);
4             solo.clearEditText(1);
5             // Username eingeben
6             solo.typeText(0, "Franz-Josef");
7             // Passwort eingeben
8             solo.typeText(1, "1234");
9             // Login button klicken
10            solo.clickOnButton("Login");
11            // Fehlgeschlagenen Login überprüfen
12            solo.assertCurrentActivity("Login erfolgreich",
                      LoginActivity.class);
13            solo.clearEditText(0);
14            solo.clearEditText(1);
15            solo.typeText(0, "Peter");
16            solo.typeText(1, "1234");
17            solo.clickOnButton("Login");
18            // Erfolgreichen Login überprüfen
19            solo.assertCurrentActivity("Login fehlgeschlagen ",
                      MainActivity.class);
20    }
```

Listing 14: Login Testfall

Das Framework soll automatisiert zuerst eine ungültige Username – Passwort Kombination eingeben (Zeile 3 – 8) und den Login-Button betätigen (Zeile 10). Unter Zuhilfenahme der assertCurrentActivity()-Methode wird der Login-Versuch überprüft (Zeile 12). Diese Methode überprüft dabei nicht direkt, ob die eingegebene Username – Passwort Kombination gültig ist, sondern in welcher Activity sich die App befindet. Befindet sich die App noch in der LoginActivity, dann ist der Login fehlgeschlagen. Wäre der Login an dieser Stelle (Zeile 10) jedoch erfolgreich gewesen, dann würde sich die Applikation in der MainActivity befinden. Dadurch wäre der Test fehlgeschlagen und würde mit einer Fehlermeldung beendet werden. Der in der assertCurrentActivity()-Methode vorhandene String würde in der entstandenen Fehlermeldung angegeben werden.
Im Anschluss soll eine gültige Kombination aus Username und Passwort eingegeben werden (Zeile 13 - 16). Nachfolgend soll wieder der Login-Button betätigt werden (Zeile 17). Dieses Mal wird abgesichert, das sich die Applikation in der MainActivity befindet, d.h. es wird überprüft, ob der Login erfolgreich war (Zeile 19).

Für die Abarbeitung der nun folgenden Testfälle wird ein erfolgreich durchgeführter Login als gegeben angenommen.

Testfall 2: Geräteliste bearbeiten

Das Framework navigiert in die Geräteliste. Dort sollen neue Geräte hinzugefügt sowie ein Gerät entfernt werden. Das zuerst hinzugefügte Gerät soll automatisch erkannt werden und das zweite soll komplett manuell eingetragen werden. Danach soll das zweite Gerät aus der gegebenen Liste entfernt werden.

Testfall 3: Nachricht versenden - Listing 15

```
1     public void testDisplayBlackBox() {
2             // Login
3             // Nachrichtenseite öffnen
4             solo.clickOnView(solo
              .getView(de.softwareforen.mct
                     .prototyp.R.id.imageViewMessage));
5             // Nachricht schreiben
6             solo.clickOnView(solo
                     .getView(de.softwareforen.mct.prototyp.R.id.write));
7             // Nachrichtendetails eingeben
8             solo.typeText(0, "Hans");
9             solo.typeText(1, "Meeting");
10            solo.typeText(2, "Sorry ich habe heute keine Zeit!");
11            // Nachricht senden
12            solo.clickOnView(solo.getView(android.R.id.button1));
13            // Button nicht direkt ansteuerbar -> Klick über Pixel
14            solo.clickOnScreen(300, 100);
15            // zu gesendeten Nachrichten wechseln
16            solo.clickInList(2);
17            // eben geschriebene Nachricht auswählen
18            solo.clickInList(2);
19            // Überprüfung ob Nachricht vorhanden
20            Assert.assertTrue(solo.searchText("Meeting"));
21    }
```

Listing 15: Nachrichten Testfall

Im nächsten Testfall soll die Nachrichtenseite geöffnet (Zeile 4) und eine Nachricht mit Inhalt und Betreff an einen vorhandenen Account geschrieben und versendet werden (Zeile 6 – 12). Daraufhin soll der Tab in der Action Bar von „Posteingang“ auf „Postausgang“ geändert werden (Zeile 14 und 16) und die Nachricht in der Liste angeklickt werden, um sie auf der Detailseite anzuzeigen (Zeile 18). Abschließend soll mit Hilfe der assertTrue()-Methode überprüft werden, ob die momentan geöffnete Activity den Text „Meeting“ beinhaltet. Ist dies der Fall, dann wurde der Testfall korrekt durchlaufen und es kam zu keinen Fehler.

Testfall 4: Profildaten bearbeiten

Hier soll die Profilseite geöffnet werden und im Bereich „Persönliches“ Vorname, Nachname, Geburtsdatum sowie das Herkunftsland geändert werden. Dazu muss zunächst in der Action Bar der Button zum Bearbeiten angeklickt werden, woraufhin die Daten geändert werden. Nach dem Ändern werden die Daten durch das erneute Klicken des Buttons gespeichert. Als nächstes soll mittels des Hardware-Zurück-Buttons in die Auswahlliste zurückgewechselt werden. Dort angekommen wird das Listenitem „Account“ angeklickt. Auf der Accountseite soll nun das Passwort von „1234“ auf „123“ geändert werden. Dabei soll beabsichtigt zuerst das alte Passwort falsch angegeben werden.

Testfall 5: Beabsichtigter Fehler im Test - Listing 16

```
1     // Login
2     // Add-Button suchen
3     solo.clickOnView(solo.getView(
            de.softwareforen.mct.prototyp.R.id.add));
```

Listing 16: Fehler Testfall

Im letzten Testfall soll ein beabsichtigter Fehler im UI-Test auftreten. Dabei wird dem Framework gesagt es soll auf einen Button klicken, welcher aber nicht vorhanden ist (Zeile 3). Das Framework versucht hier, den Add-Button in der Action Bar zu suchen und danach zu betätigen. Da sich das Framework jedoch noch im Hauptmenü befindet und dort der Add-Button natürlich nicht vorhanden ist, entsteht hier ein Fehler

5.2.3. Ergebnisse

Die ersten beiden durchgeführten Testfälle aus Listing 14 und Listing 15 verliefen wie zu erwarten ohne Probleme. Eclipse meldete nach 15,26 bzw. 13,57 Sekunden, dass der jeweilige Test erfolgreich durchgeführt wurde. Abbildung 5.11 und Abbildung 5.12 zeigen die Ergebnisse, wie sie in Eclipse dargestellt werden.

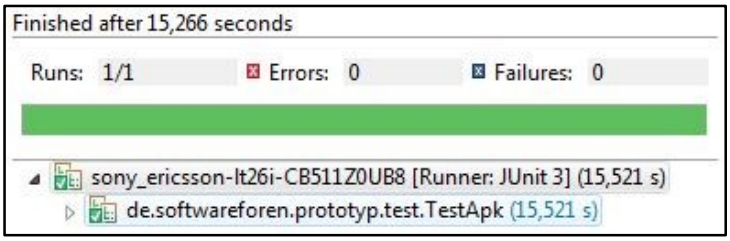

Abbildung 5.11: Ergebnis Login Testfall

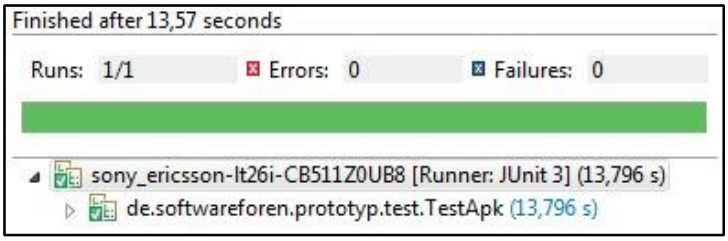

Abbildung 5.12: Ergebnis Nachrichten Testfall

Der mit Absicht eingebaute Fehler aus Listing 16 hat natürlich ein anderes Ergebnis zur Folge. Dieses ist in Abbildung 5.13 zu finden.

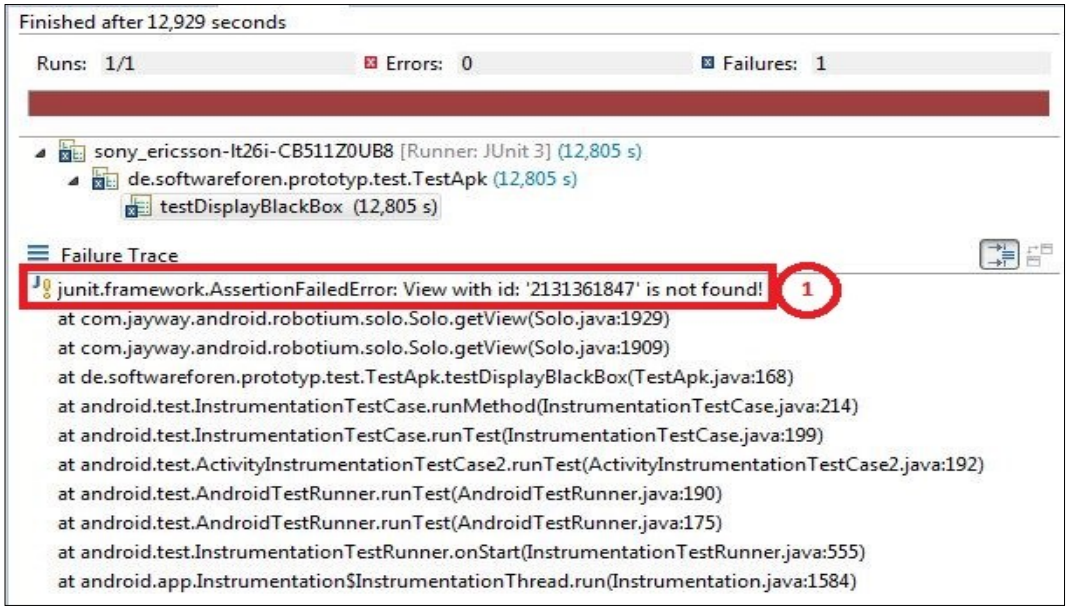

Abbildung 5.13: Ergebnis Fehler Testfall

Wie im Failure Trace zu erkennen ist, ist ein Fehler aufgetreten, da die UI-Komponente mit der ID „2131361847“ nicht gefunden werden konnten (1). Der UI-Test wurde daraufhin abgebrochen und die dargestellte Auswertung wurde angezeigt.

5.3. Usability-Test

5.3.1. Testfälle

Die folgenden Testfälle wurden dazu erstellt, um alle wichtigen Funktionen des Prototyps auf deren Usability zu überprüfen. Die Testfälle werden zwei Personen vorgelegt, welche diese dann durchführen und ein kurzes Feedback geben. Der Login wird im Weiteren nicht berücksichtigt und als bereits erfolgreich durchgeführt angenommen. Alle Testfälle gehen davon aus das sich der Tester bereits im

Hauptmenü befindet. Die Tests werden alle auf demselben Gerät durchgeführt, in diesem Fall ist dies ein Smartphone. Die Form des Feedbacks ist ein kurzes Auswertungsgespräch. Der Usability-Test selbst ist noch ein erster sehr früher Test der Grundfunktionalitäten. Die Tester sollen dabei gezielt auf die folgenden Kriterien achten:

- Bedienbarkeit,
- Allgemeiner Aufbau der App,
- Menüführung,
- Intuitiver Aufbau,
- Dialogaufbau.

Testfall 1: Account-Daten ändern
Der Tester soll in sein Profil navigieren und dort unter „Account“ sein Passwort und unter „Persönliches“ einige Daten des Test-Accounts ändern. Die Daten sollen nach deren Veränderung gespeichert werden.

Testfall 2: Nachricht versenden und Nachricht aus Postausgang löschen
Nach der Navigation in das Nachrichtensystem wird der Tester zunächst eine neue Nachricht verfassen und versenden. Danach soll er in seinen Postausgang wechseln, wo die neu versendete Nachricht aufgelistet ist. Zum Schluss wird diese vom Tester gelöscht.

Testfall 3: Zwei Geräte hinzufügen und entfernen
Im letzten Testfall soll der Tester vom Hauptmenü in den Gerätemanager wechseln. Dort angekommen fügt er zwei neue Geräte hinzu. Das erste Gerät wird dabei das verwendete Smartphone sein. Der Tester benutzt also den Dialog-Button „Gerätedaten erkennen“ um die Geräteigenschaften zu erfassen. Im Anschluss soll ein Gerät hinzugefügt werden, bei dem der Tester alle Daten selbst einträgt. Sind beide Geräte hinzugefügt, muss verifiziert werden, dass alle Daten korrekt in die Datenbank übernommen wurden und in den Detailseiten der Geräte richtig angezeigt werden. Zum Schluss werden diese Geräte wieder entfernt.

5.3.2. Feedback durch Tester und Anpassung der App

Das Feedback der Tester wird nicht für jeden einzelnen Testfall separat dargestellt werden, sondern kompakt für alle Testfälle.
Tester 1 empfand die einfache Steuerung aller Aktionen über die Action Bar als sehr positiv. Des Weiteren sagte ihm die Menüführung durch ein Hauptmenü zu, welches auf alle anderen Unterseiten mit ihren jeweiligen Funktionen verweist. Die Dialoge zur Datenabfrage wurden auch sehr positiv angenommen, besonders die Möglichkeit das eigene Gerät automatisch erkennen zu lassen. Außerdem gefiel ihm das Anpassen des Layouts beim Drehen des Gerätes. Ein aufgefallener Kritikpunkt ist der Passwort-Ändern-Dialog. Dieser erwartet zwangsweise die richtigen Eingaben aller drei Passwörter oder es wird eine Fehlermeldung angezeigt und es passiert nichts. Der Nutzer hat also nur die Möglichkeit die Änderung des Passwortes abzubrechen, indem er zurück auf eine vorher aufgerufene Seite wechselt. Des Weiteren stellte der Tester fest, dass ein Home-Button auf der Profilseite nicht funktioniert. Außerdem wird nach Betätigen eines Home-Buttons die Anzahl der ungelesenen Nachrichten im Hauptmenü nicht mehr angezeigt.
Der zweite Tester empfand ebenfalls die einfache Menüsteuerung sowie das Hauptmenü als positiv. Die Steuerung über die Action Bar und Dialoge sei sehr intuitiv aufgebaut und handlich. Außerdem wurde die Möglichkeit zur automatischen Geräteerkennung als sehr gut wahrgenommen. Die Kritikpunkte decken sich weitestgehend mit denen des ersten Testers. Hierzu zählen der nicht funktionierende Home-Button und die nicht mehr dargestellte Anzahl der ungelesen Nachrichten im Hauptmenü, nachdem ein Home-Button betätigt wurde.

Das Feedback beider Tester machte deutlich, dass die App noch kleinere Fehler aufweist. Diese wurden nach ihrer Lokalisierung im Quellcode behoben, speziell der Passwort-Ändern-Dialog wurde angepasst. Die Usability der App wurde ansonsten von beiden Testern als gut befunden, d.h. es sind keine weiteren Anpassungen notwendig.

6. Abschließende Bemerkungen

Ziel der Arbeit war es, einen Überblick über die Thematik des Crowdsourcing und Crowdtesting zu geben, die Grundlagen der Bedienkonzepte in Android darzustellen sowie deren prototypische Umsetzung in einer Android-Applikation. Außerdem sollten die Möglichkeiten des UI-Testing in Android näher erläutert und mit Hilfe eines Frameworks automatisiert durchgeführt werden. Des Weiteren sollte die entwickelte Applikation durch mehrere Personen auf ihre Usability getestet werden. Zusammenfassend wurde all dies erfolgreich umgesetzt und durchgeführt. Es wurden Grundfunktionalitäten implementiert, die die UI-Gestaltung in Android demonstrieren sollen und einen Ausblick auf mögliche komplexere Layouts geben. Die zu Beginn festgelegten Anforderungen an den Prototyp wurden alle erfüllt. Die UI-Evaluierung beschreibt detailliert, aus welchen Einzelteilen eine Android-App bestehen kann und wie diese funktionieren. Die Umsetzung der Kompatibilität der App zwischen Smartphone und Tablet konnte auch gewährleistet werden. Geräte ab einer Bildschirmgröße von mehr als sieben Zoll nutzen das für Tablets optimierte Layout, andernfalls wird das normale Smartphone-Layout genutzt. Die Datenspeicherung aller Beispieldaten erfolgt in einer SQLite-Datenbank direkt auf dem Gerät. Mit Hilfe des Usability-Tests wurden einige Schwachstellen der App aufgedeckt. Diese konnten im Nachhinein beseitigt werden. Der automatisiert durchgeführte UI-Test deckte im Gegenzug keine Fehler auf.

Der erste Schritt zur Weiterentwicklung des Prototyps wäre es, die fehlende Testfallverwaltung sowie alle weiteren Features zu implementieren. Ein weiterer Schritt zur Verbesserung ist die Zusammenführung von App und der für die Anwendung vom PC ausgelegten Webplattform. Dazu zählt auch die Umsetzung der Schnittstelle zwischen App und Webplattform, damit die App ihre benötigten Daten von der Plattform bereitgestellt bekommt.

Glossar

Crowdsourcing: Crowdsourcing beschreibt die Auslagerung interner Unternehmensaufgaben an Gruppen von arbeitswilligen Leuten, die über das Internet organisiert sind. Diese Gruppen werden als „Crowd“ bezeichnet. Der Begriff „Crowdsourcing“ ist an das aus der Ökonomie stammende Outsourcing angelehnt.

Crowdtesting: Unter Crowdtesting ist die Anwendung des Crowdsourcing zum Testen von Software zu verstehen. Softwarehersteller geben dabei ihre zu testende Software an die Crowd weiter, welche dann die gewünschten Tests durchführt.

Mobile Crowdtesting: Mobile Crowdtesting bezeichnet die Anwendung des Crowdtesting auf mobile Anwendungen für Smartphones und Tablets.

Activity: Eine Activity ist die Komponente einer Android-App, welche das User Interface beinhaltet. Eine Android-App besteht in der Regel aus einer Vielzahl von Activities.

Fragment: Fragmente stellen Teile des UI einer einzelnen Activity dar. Durch die Verwendung mehrerer Fragmente in einer Activity kann ein mehrteiliges UI erzeugt werden. Fragmente wurden mit der Android-Version 3.0 eingeführt.

Dialog: Dialoge sind Fenster, die dazu dienen zusätzliche Nutzereingaben anzufordern. Der Aufbau dieser Fenster reicht von simplen Ja-Nein-Dialogen bis hin zu selbstentwickelten, komplexen Layouts.

Action Bar: Action Bars helfen dem Nutzer wichtige und häufig verwendete Funktionen schnell aufzurufen. Oftmals werden diese im Header der App angelegt.

Android Developer Framework: Das im Android-SDK enthalte Framework dient der automatisierten Durchführung von UI-Tests. Dieses Framework besteht aus zwei Tools, dem „uiautomatorviewer“ und dem „uiautomator“. Es wird eine Android-Version von mindestens 4.1 vorausgesetzt, um das Framework benutzen zu können.

Robotium: Robotium ist ein weiteres Framework zur automatisierten Durchführung von UI-Tests. Es ist nicht im Android-SDK enthalten und muss extern von der Entwicklerseite bezogen werden. Das Framework ist ein Open Source Projekt.

Quellenverzeichnis

[Andr13a] ANDROID DEVELOPERS: *Activities | Android Developers*. URL http://developer.android.com/guide/components/activities.html. - abgerufen am 2013-07-08

[Andr13b] ANDROID DEVELOPERS: *Tasks and Back Stack | Android Developers*. URL http://developer.android.com/guide/components/tasks-and-back-stack.html. - abgerufen am 2013-10-24

[Andr13c] ANDROID DEVELOPERS: *Fragments | Android Developers*. URL http://developer.android.com/guide/components/fragments.html#Design. - abgerufen am 2013-07-08

[Andr13d] ANDROID DEVELOPERS: *Swipe Views | Android Developers*. URL http://developer.android.com/design/patterns/swipe-views.html. - abgerufen am 2013-10-19

[Andr13e] ANDROID DEVELOPERS: *Dialogs | Android Developers*. URL http://developer.android.com/design/building-blocks/dialogs.html. - abgerufen am 2013-11-03

[Andr13f] ANDROID DEVELOPERS: *Action Bar | Android Developers*. URL http://developer.android.com/design/patterns/actionbar.html. - abgerufen am 2013-10-19

[Andr13g] ANDROID DEVELOPERS: *UI Testing | Android Developers*. URL http://developer.android.com/tools/testing/testing_ui.html. - abgerufen am 2013-07-08

[Andr13h] ANDROID DEVELOPERS: *Toasts | Android Developers*. URL http://developer.android.com/guide/topics/ui/notifiers/toasts.html. - abgerufen am 2013-10-19

[Borl13] BORLAND: *Performance Testing*. URL http://www.borland.com/sdlc/performance-testing/. - abgerufen am 2013-10-28

[Clic13] CLICKWORKER: *Crowdsourcing Definition*. URL https://www.clickworker.com/crowdsourcing-glossar/crowdsourcing-definition/. - abgerufen am 2013-07-01

[Grim12] GRIMME-INSTITUT: *Crowdsourcing*. URL http://www.grimme-institut.de/imblickpunkt/pdf/IB-Crowdsourcing.pdf. - abgerufen am 2013-10-21

[Guru13] GURU99: *Functional Testing - Stuff You Must Know!* URL http://www.guru99.com/functional-testing.html. - abgerufen am 2013-07-05

[Java13] JAVABEGINNERS: *Einfach verkettete Liste*. URL http://www.javabeginners.de/Sammlungen_und_Listen/Einfach_verkettete_Liste.php. - abgerufen am 2013-10-19

[Ptex12] PTEXT: *Trend Crowdtesting | Apps und mobile Anwendungen*. URL http://www.ptext.de/pressemitteilung/trend-crowdtesting-311758. - abgerufen am 2013-10-23

[Reda13] REDA, RENES: *QuestionsAndAnswers - robotium - Common Robotium Questions & Answers - The world's leading Android™ test automation framework - Google Project Hosting*. URL https://code.google.com/p/robotium/wiki/QuestionsAndAnswers. - abgerufen am 2013-08-28

[Sigg13] SIGGE: *Robotium: Android automation testing taken one step further | Happytesting*. URL http://happytesting.wordpress.com/2010/01/27/robotium-android-automation-testing-taken-one-step-further/. - abgerufen am 2013-08-28

[Test13] TESTBIRDS.DE: *Frequently Asked Questions | Testbirds.de*. URL https://nest.testbirds.com/de/faq#t152n3991. - abgerufen am 2013-09-30

[Utes13a] UTEST: *Security Testing for Web, Mobile and Desktop Applications | uTest*. URL https://www.utest.com/security-testing. - abgerufen am 2013-07-05

[Utes13b] UTEST: *Usability Testing for Web, Desktop and Mobile Applications | uTest*. URL https://www.utest.com/usability-testing. - abgerufen am 2013-10-19

[Utes13c] UTEST: *Localization Testing | uTest*. URL https://www.utest.com/localization-testing. - abgerufen am 2013-07-05

[Wire06] WIRED MAGAZINE: *The Rise of Crowdsourcing*. URL http://www.wired.com/wired/archive/14.06/crowds.html. - abgerufen am 2013-09-09

[Zieg13] ZIEGLER, WIBKE: *Crowdtesting - Bugtracking via Crowd*. URL http://blog.volksbank-buehl.de/2013/01/17/crowdtesting-bugtracking-crowd/. - abgerufen am 2013-10-23

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbstständig und nur unter Zuhilfenahme der angegebenen Quellen erstellt habe.

_____	_____
Ort,Datum	Unterschrift