

# Projektdokumentation «Lifeguard»

## Team

- Aregger, Thomas
- Daniel, David
- Gutknecht, Jürg
- Kälin, Christof

# Inhaltsverzeichnis

Team.....	1
Anwendungsfälle.....	3
Use Case Diagramm.....	3
Use Case Beschreibungen.....	4
Architektur.....	8
Verteilungsdiagramm.....	8
System und Services.....	8
Zustände des Alarm Service.....	8
Persistenz.....	11
Database-Helper.....	11
Table Data Gateway.....	11
Implementierung.....	11
Problemdomäne.....	12
Serialisierung.....	12
Klassendiagramm.....	12
Bedienkonzept.....	14
Anforderungen an das Bedienkonzept.....	14
Hierarchie.....	15
Hauptansicht.....	15
Ansicht.....	16
Kontaktliste.....	16
Ansicht.....	17
Kontakt Detailansicht.....	17
Ansicht.....	18
Konfiguration.....	18
Ansicht.....	18
Testkonzept.....	19
Verwendete Tests.....	19
Automatisierte Tests.....	19
Manuelle Tests.....	19
Bugtracking.....	19
Testphasen.....	19
Entwicklertests.....	19
Benutzertests vor Abnahme.....	20
Abnahmetests.....	20
Vollständige Liste der Testfälle (automatisch und manuell).....	20
Automatische Tests.....	20
Manuelle Tests.....	22
Changelist Dokumentation.....	23

# Anwendungsfälle

## Use Case Diagramm

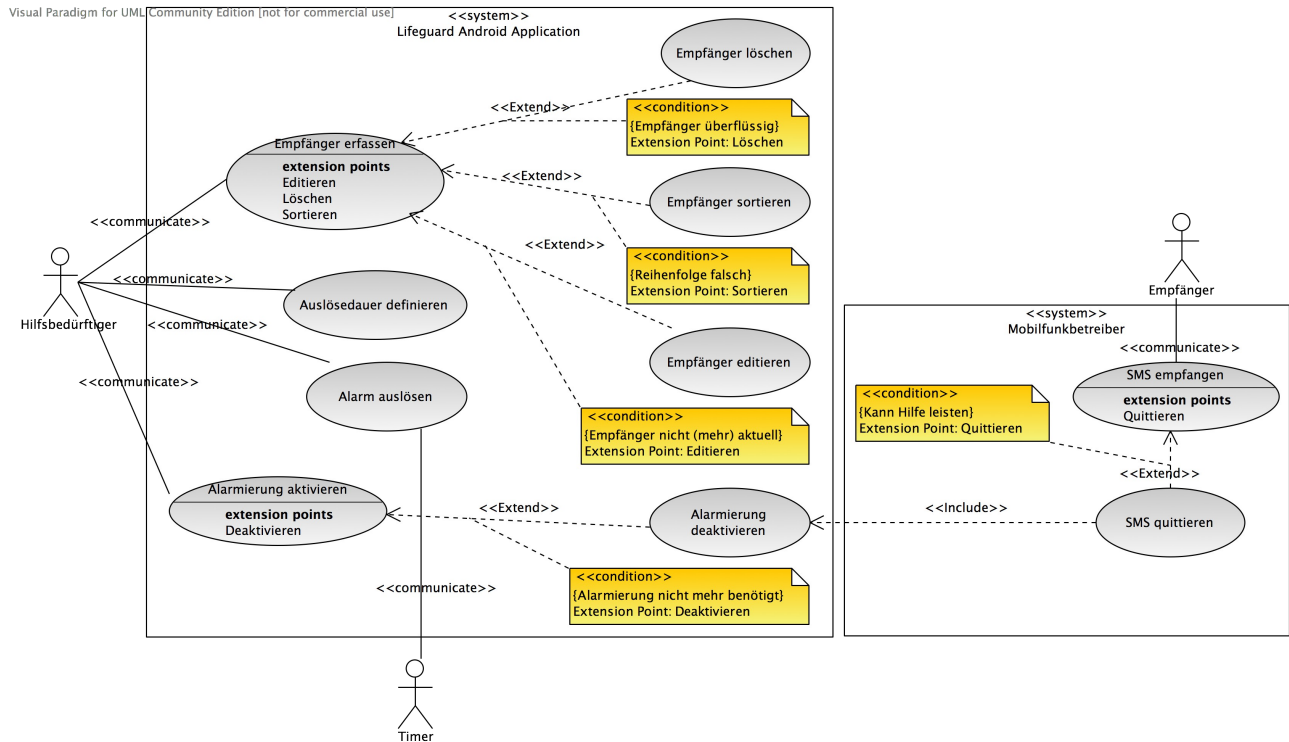


Abbildung 1: Use Case Diagramm

## Use Case Beschreibungen

<b>Name</b>	<b>UC1 Empfänger erfassen</b>
<b>Kurzbeschreibung</b>	Es wird eine Nummer und der Name eines möglichen Empfängers erfasst.
<b>Akteure</b>	Hilfsbedürftiger
<b>Auslösendes Ereignis</b>	Der Hilfsbedürftige möchte einen möglichen Empfänger erfassen
<b>Vorbedingung</b>	-
<b>Eingehende Informationen</b>	Nummer des Empfängers Name des Empfängers
<b>Ablauf (essentielle Schritte)</b>	Name erfassen Nummer erfassen
<b>Ausnahmefälle</b>	Es wurden bereits 5 Empfänger definiert Der gewünschte Empfänger wurde bereits erfasst
<b>Nachbedingung</b>	Der Empfänger wurde erfasst
<b>Zeitverhalten</b>	-
<b>Verfügbarkeit</b>	-
<b>Fragen, Kommentare</b>	-

<b>Name</b>	<b>UC2 Empfänger editieren</b>
<b>Kurzbeschreibung</b>	Ein bereits im System erfasster Empfänger wird editiert
<b>Akteure</b>	Hilfsbedürftiger
<b>Auslösendes Ereignis</b>	Der Name oder die Nummer des Empfängers hat sich geändert oder sind falsch eingegeben worden
<b>Vorbedingung</b>	Der Empfänger wurde bereits erfasst
<b>Eingehende Informationen</b>	Name des Empfängers Nummer des Empfängers
<b>Ablauf (essentielle Schritte)</b>	Der gewünschte Empfänger wird ausgewählt Der Empfänger wird editiert
<b>Ausnahmefälle</b>	Die Nummer oder der Name stimmt mit einem anderen, bereits vorhandenen Benutzer überein
<b>Nachbedingung</b>	Der Empfänger wurde aktualisiert, die neuen Angaben wurden gespeichert.
<b>Zeitverhalten</b>	-
<b>Verfügbarkeit</b>	-
<b>Fragen, Kommentare</b>	-

<b>Name</b>	<b>UC3 Empfänger löschen</b>
<b>Kurzbeschreibung</b>	Ein bereits im System erfasster Empfänger wird gelöscht
<b>Akteure</b>	Hilfsbedürftiger
<b>Auslösendes Ereignis</b>	Der Hilfsbedürftige möchte den Empfänger nicht mehr benachrichtigen
<b>Vorbedingung</b>	Der Empfänger wurde bereits erfasst
<b>Eingehende Informationen</b>	-
<b>Ablauf (essentielle Schritte)</b>	Der gewünschte Empfänger wird ausgewählt Der Empfänger wird gelöscht (bestätigt)
<b>Ausnahmefälle</b>	Es handelte sich um den letzten Empfänger – dies deaktiviert die Alarmierung.
<b>Nachbedingung</b>	Der gelöschte Empfänger wurde aus der Liste der Empfänger gelöscht und nicht mehr benachrichtigt.
<b>Zeitverhalten</b>	-
<b>Verfügbarkeit</b>	-
<b>Fragen, Kommentare</b>	-

<b>Name</b>	<b>UC4 Alarmierung aktivieren / deaktivieren</b>
<b>Kurzbeschreibung</b>	Die Alarmierung soll an- resp. ausgeschaltet werden
<b>Akteure</b>	Hilfsbedürftiger
<b>Auslösendes Ereignis</b>	Der Aktivitäts-Zustand des Hilfsbedürftigen hat sich geändert.
<b>Vorbedingung</b>	Es wurde mind. 1 Empfänger definiert
<b>Eingehende Informationen</b>	De-/Aktivierung
<b>Ablauf (essentielle Schritte)</b>	Der Alarm wird aktiviert oder deaktiviert
<b>Ausnahmefälle</b>	-
<b>Nachbedingung</b>	Der Alarm wurde aktiviert oder deaktiviert
<b>Zeitverhalten</b>	-
<b>Verfügbarkeit</b>	-
<b>Fragen, Kommentare</b>	-

<b>Name</b>	<b>UC5 Auslösedauer definieren</b>
<b>Kurzbeschreibung</b>	Die Dauer, bis wann ein Alarm ausgelöst wird, wird konfiguriert
<b>Akteure</b>	Hilfsbedürftiger
<b>Auslösendes Ereignis</b>	Die Alarmdauer ist zu kurz oder zu lang
<b>Vorbedingung</b>	-
<b>Eingehende Informationen</b>	Die Auslösedauer
<b>Ablauf (essentielle Schritte)</b>	Die Dauer wird konfiguriert
<b>Ausnahmefälle</b>	-
<b>Nachbedingung</b>	Die Auslösedauer wurde eingestellt
<b>Zeitverhalten</b>	-
<b>Verfügbarkeit</b>	-
<b>Fragen, Kommentare</b>	-

<b>Name</b>	<b>UC6 Alarm auslösen</b>
<b>Kurzbeschreibung</b>	Der Alarm wird ausgelöst, eine oder mehrere Nachrichten werden versandt
<b>Akteure</b>	Hilfsbedürftige / Timer
<b>Auslösendes Ereignis</b>	Die Auslösedauer wurde überschritten oder der Alarm wurde manuell ausgelöst
<b>Vorbedingung</b>	Die Auslösedauer wurde überschritten / der Alarm wurde manuell ausgelöst
<b>Eingehende Informationen</b>	Alarm
<b>Ablauf (essentielle Schritte)</b>	Alarm auslösen / der Timer löst den Alarm aus
<b>Ausnahmefälle</b>	-
<b>Nachbedingung</b>	Mind. Eine Nachricht wurde versandt
<b>Zeitverhalten</b>	-
<b>Verfügbarkeit</b>	-
<b>Fragen, Kommentare</b>	-

<b>Name</b>	<b>UC7 SMS empfangen</b>
<b>Kurzbeschreibung</b>	Der Empfänger erhält eine Alarm-Nachricht
<b>Akteure</b>	Empfänger
<b>Auslösendes Ereignis</b>	Alarm
<b>Vorbedingung</b>	Der Alarm wurde ausgelöst
<b>Eingehende Informationen</b>	Wer hat den Alarm von wo ausgelöst
<b>Ablauf (essentielle Schritte)</b>	Das SMS wird empfangen
<b>Ausnahmefälle</b>	Der Alarm wurde aus Versehen ausgelöst
<b>Nachbedingung</b>	Der Empfänger wurde über den Alarm informiert
<b>Zeitverhalten</b>	-
<b>Verfügbarkeit</b>	-
<b>Fragen, Kommentare</b>	-

<b>Name</b>	<b>UC8 SMS quittieren</b>
<b>Kurzbeschreibung</b>	Der Empfänger der Alarm-Nachricht quittiert den Empfang der Nachricht
<b>Akteure</b>	Empfänger, Timer
<b>Auslösendes Ereignis</b>	Empfang einer Alarm Nachricht
<b>Vorbedingung</b>	Es wurde eine Alarm-Nachricht empfangen
<b>Eingehende Informationen</b>	-
<b>Ablauf (essentielle Schritte)</b>	Das SMS wird mit entsprechendem Vermerk an den Absender retourniert
<b>Ausnahmefälle</b>	Timer: Falls keine Quittung erhalten und Empfänger-Nr. < 5: SMS an nächsten Empfänger senden Warten auf Quittierung
<b>Nachbedingung</b>	Der Hilfsbedürftige / der Timer empfängt die Rückmeldung
<b>Zeitverhalten</b>	-
<b>Verfügbarkeit</b>	-
<b>Fragen, Kommentare</b>	-

# Architektur

## Verteilungsdiagramm

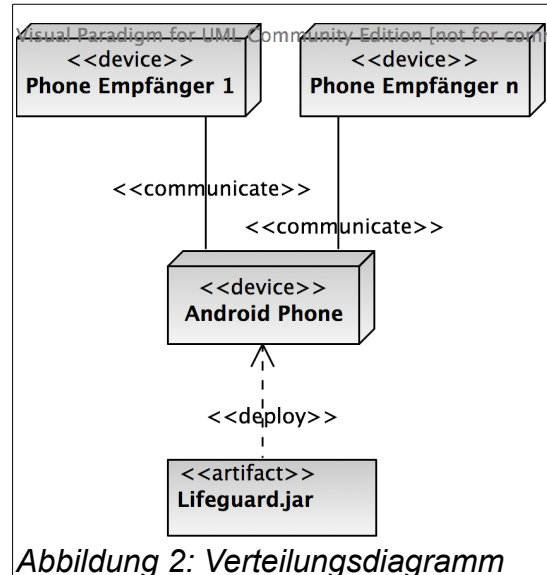


Abbildung 2: Verteilungsdiagramm

## System und Services

Es sind keine besonders rechenintensive Aufgaben zu erledigen. Es wird jedoch ein Service benötigt, welcher auch bei geschlossener Anwendung sicherstellt, dass

- Die Bewegungen des Gerätes registriert werden und der Zeitgeber dementsprechend seinen Zustand anpasst.
- Der Alarm ausgelöst wird.
- Die Quittierung des Hilfestellenden empfangen wird.

## Zustände des Alarm Service

Grundsätzlich muss zwischen unterschiedlichen Zuständen unterschieden werden, je nachdem in welchem Zustand sich die Anwendung befindet, verrichtet der laufende Service unterschiedliche Aufgaben. Folgender Ablauf kann festgehalten werden:

1. Der Service wird neu erstellt.
2. Entweder startet der Benutzer den Zählvorgang (zu Schritt 3) oder er verlässt die Applikation wieder ohne den Zählvorgang zu starten. Der Benutzer kann den Alarm auch manuell sofort auslösen (zu Schritt 4).
3. Der Zählvorgang startet und die Benutzer-Aktivität wird überwacht.
  - Der Benutzer bricht den Vorgang ab (zu Schritt 2)
  - Der Benutzer löst den Alarm manuell aus (zu Schritt 4)
  - Der Zähler erreicht die nötige Grösse (zu Schritt 4)
4. Der Alarm wird ausgelöst, resp. es wird ein SMS versandt.
5. Es wird auf eine Antwort gewartet.



- Das SMS wird vom Empfänger quittiert (zu Schritt 6).
- Nach gewisser Zeit wird nicht mehr mit einer Rückmeldung gerechnet (zu Schritt 4).

6. Der Alarm wird zurückgesetzt, der Benutzer wird benachrichtigt.

Das obige Status-Modell ist in Abbildung 3 visualisiert.

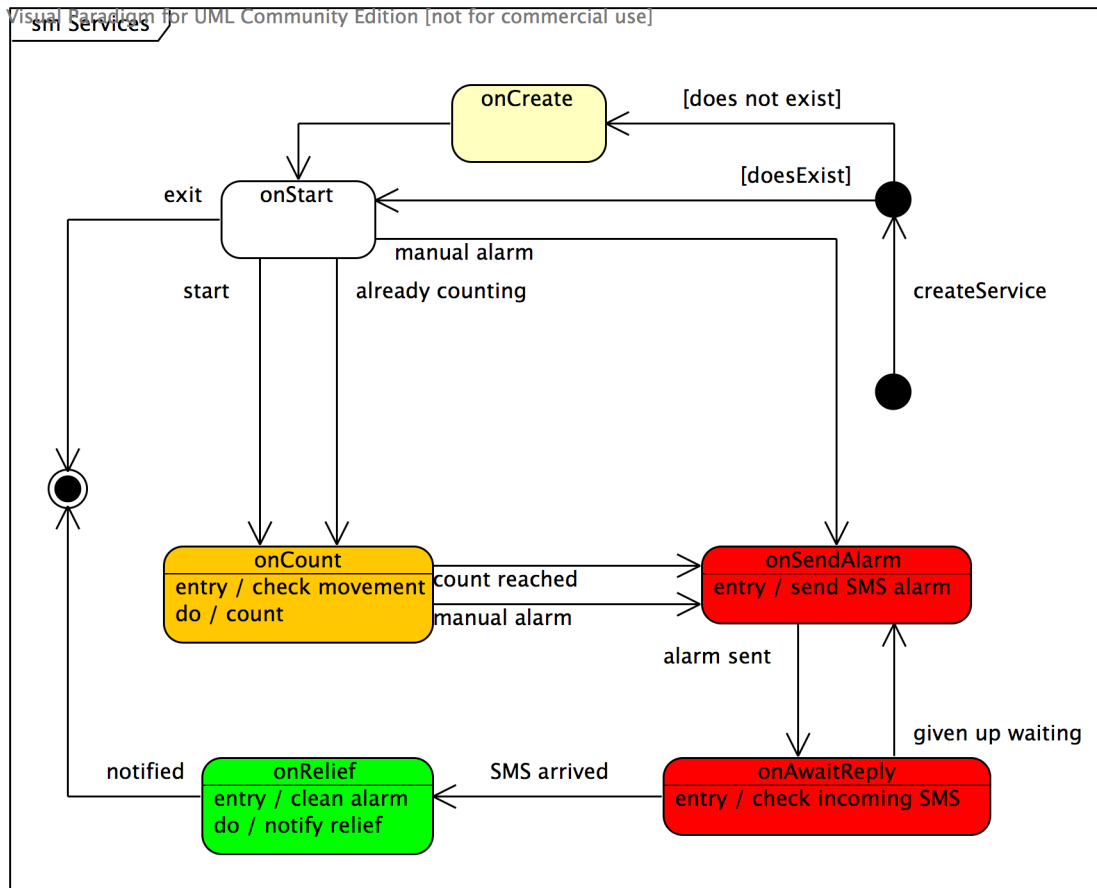


Abbildung 3: Zustandsdiagramm des Alarm-Service Lifecycle

Das obige Modell eignet sich hervorragend für die Implementierung mittels dem Zustands-Muster. Unter Vernachlässigung der Start- und End-Zustände können die folgenden für die Anwendung relevanten Zustände festgehalten werden:

- Initial
- Zählend (Bewegungen beobachten, Zähler überwachen)
- Alarmierend (SMS wird versandt)
- Auf Rückmeldung wartend
- Die Rückmeldung propagierend

Da die jeweiligen Zustände auch das zugehörige Verhalten kapseln sollen, wird auch die Entscheidung der Übergänge zwischen den Zuständen an die Zustände selbst übergeben. Ein Zustand muss daher über den Kontext den nächsten Zustand setzen können. Hierzu muss der jeweilige Zustand die möglichen Folgezustände kennen und diese bei Bedarf anwenden.

Der Alarm-Service hält sich schliesslich eine Referenz zu dem Kontext und lauscht auf dessen Zustandsänderungen. Bei jedem Zustandswechsel und beim Start des Services benachrichtigt dieser die Anwendung, welche dem Benutzer den Zustand des Services präsentieren kann.

Das Klassendiagramm in Abbildung 4 zeigt das Alarm Zustands-Modell.

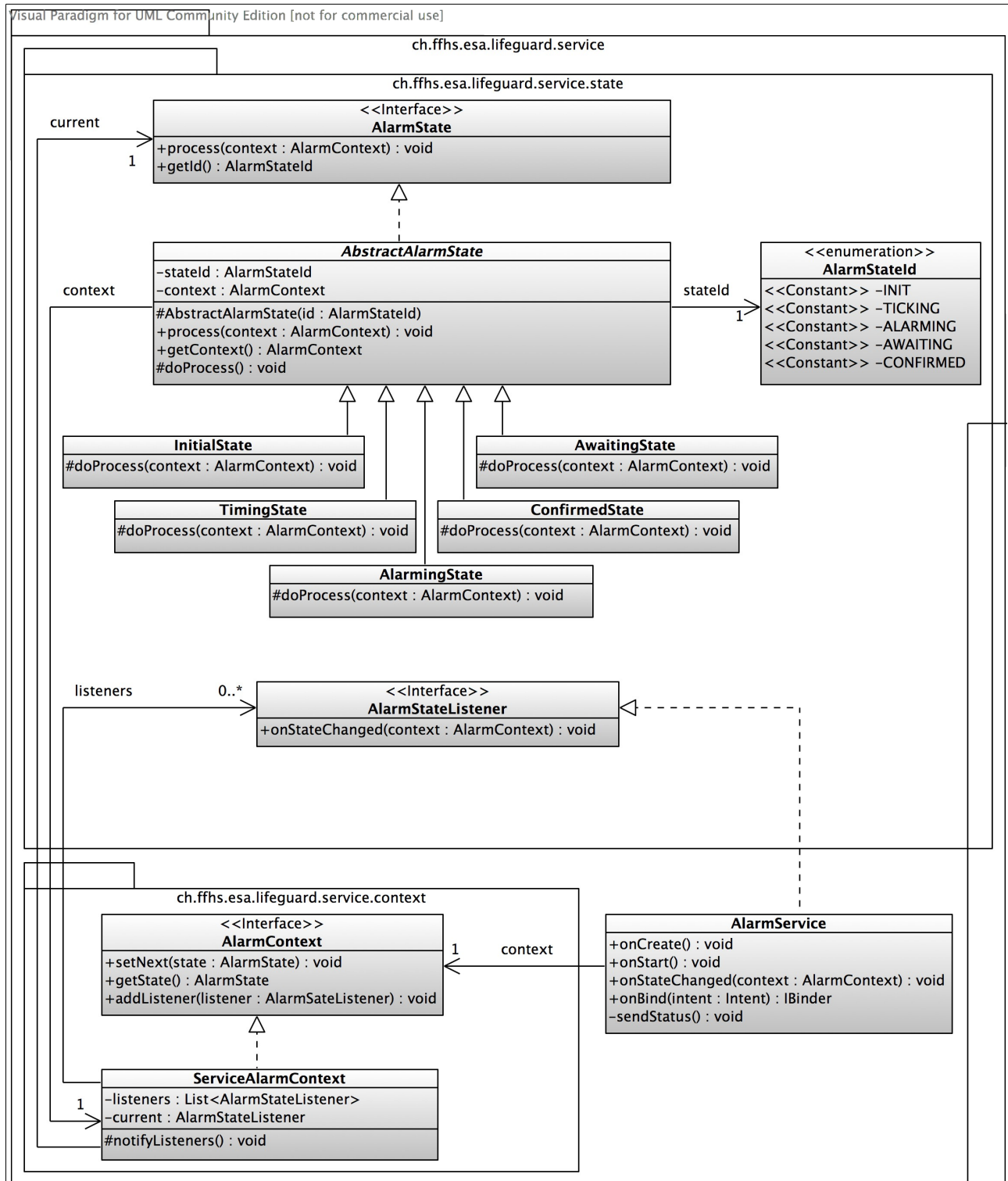


Abbildung 4: Klassendiagramm des Statusmodells

## Persistenz

Zur Speicherung der Applikationsdaten kommt das Datenbanksystem SQLite zum Einsatz, welches eine gute Integration in Android besitzt.

Der eingeschränkten Ressourcen mobiler Geräte wegen, wird auf die Verwendung einer vollumfänglichen ORM/DBAL-Lösung wie bspw. Hibernate verzichtet. Die geringe Anzahl vorhandener Entitäten in der Problemdomäne liesse grundsätzlich die Ausführung einzelner Datenbankstatements innerhalb der Activities zu. Im Sinne loser Entkopplung, und damit einhergehender erhöhter Testbarkeit und Wartbarkeit der Anwendung, wird jedoch eine dünne Schicht an Persistenzlogik eingeführt.

## Database-Helper

Um den Zugriff auf die Datenbank in der Anwendung zu ermöglichen, wird eine statische Referenz zum Database-Helper in der Applikationsklasse Lifeguard hinterlegt.

## Table Data Gateway

Die implementierte Persistenzlösung basiert auf dem Entwurfsmuster *Table Data Gateway*<sup>1</sup>, bei welchem sogenannte Gateway-Klassen jeweils eine Datenbanktabelle repräsentieren und sämtlichen Zugriff auf die persistierten Daten regeln.

Einzelne Datensätze werden vom Gateway in einfache Model-Objekte überführt, welche selber keinerlei Geschäfts- oder Persistenzlogik enthalten.

## Implementierung

Das Table-Data-Gateway-Pattern wird im Package `ch.ffhs.esa.lifeguard.persistence` implementiert. Die Interfaces `TableGatewayInterface` und `Persistable` definieren die Schnittstellen für Implementierungen der Gateway- resp. Modell-Klassen.

Weiter existiert eine abstrakte `TableGateway`-Klasse, welche sämtlichen Gateway-Implementierungen gemeinsame Logik zur Verfügung stellt. Eine konkrete Gateway-Klasse, bspw. `Contacts`, erbt folglich von `TableGateway` und implementiert gleichzeitig das Interface `TableGatewayInterface` bzw. eine Spezialisierung derselben.

Die Table-Gateways halten eine Referenz zum Database-Helper, welche im Sinne von Dependency Injection an den Gateway-Konstruktor übergeben wird. Dadurch lässt sich im Testfall ein Database-Helper übergeben, welcher auf einer Testdatenbank operiert und die Applikationsdatenbank nicht tangiert.

Abfragen an SQLite liefern in der Regel ein Resultateobjekt vom Typ `Cursor`. Für die Anzeige von Datensätzen in Listen wird dann ein `CursorAdapter` verwendet, welchem ein Mapping von Datenfeldern auf Views übergeben wird. Um die `ListView`s nicht eng an diese `Cursor`, und somit die Datenbank, zu koppeln, wird von Query-Methoden wie `TableGatewayInterface.findById()` oder `TableGatewayInterface.getAll()` kein `Cursor` sondern Instanzen der entsprechenden Modellklassen zurückgegeben. Um solche Instanzen ebenfalls in Listen anzeigen zu können, sind im `Persistence`-Package angepasste Adapter definiert, welche für die konkreten Datentypen spezialisiert werden. Ein Beispiel ist der `TableGatewayTwoLineAdapter`, mit welchem Datensätze in zweizeiligen `ListItemViews` dargestellt werden können.

---

<sup>1</sup> Patterns of Enterprise Application Architecture: Table Data Gateway,  
<http://martinfowler.com/eaCatalog/tableDataGateway.html> (abgerufen: 15.11.2013)

## Problemdomäne

Während im Persistence-Package die generischen Interfaces und Klassen zum Lesen und Schreiben von Daten definiert sind, werden die konkreten Datentypen der Problemdomäne im Package `ch.ffhs.esa.lifeguard.domain` realisiert. Am Beispiel der Kontaktdaten wird ersichtlich, dass ein Datentyp durch zwei Schnittstellen und zwei Klassen realisiert wird:

1. `ContactsInterface`: Eine Spezialisierung des `TableGatewayInterface`, welches in der Regel eine reines Marker-Interface ist.
2. `Contacts`: Die Table-Gateway-Implementierung für die Kontaktdaten
3. `ContactInterface`: Eine Spezialisierung des `Persistable-Interface`, welches die Schnittstelle des Datentyps definiert.
4. `Contact`: Die Modellklasse zur Repräsentation eines einzelnen Datensatzes.

Zudem können Listen-Adapter, wie der `ContactsListAdapter` für jeden Datentyp implementiert werden.

## Serialisierung

Das `Persistable-Interface` erweitert das `Serializable-Interface`, wodurch Instanzen der Modellklassen serialisiert und zwischen Activities übertragen werden können.

Nicht zuletzt deswegen empfiehlt sich die Verwendung spezialisierter List-Adapter anstelle von Standard-Adapttern wie `ArrayAdapter` oder `SimpleCursorAdapter`. Während der `ArrayAdapter` die `toString`-Methode jedes Objekts im Array aufruft und somit die String-Repräsentation des Objekts in der Liste anzeigt, wird bspw. beim `ContactsListAdapter` eine `Contact`-Instanz mit jeder `ListItemView` assoziiert, welche bei einem allfälligen Klick (resp. Tippen) referenziert und serialisiert an die `ContactDetailActivity` übermittelt werden kann.

## Klassendiagramm

Das nachfolgende Klassendiagramm visualisiert das Zusammenspiel von `ch.ffhs.esa.lifeguard.persistence` und `ch.ffhs.esa.lifeguard.domain`.

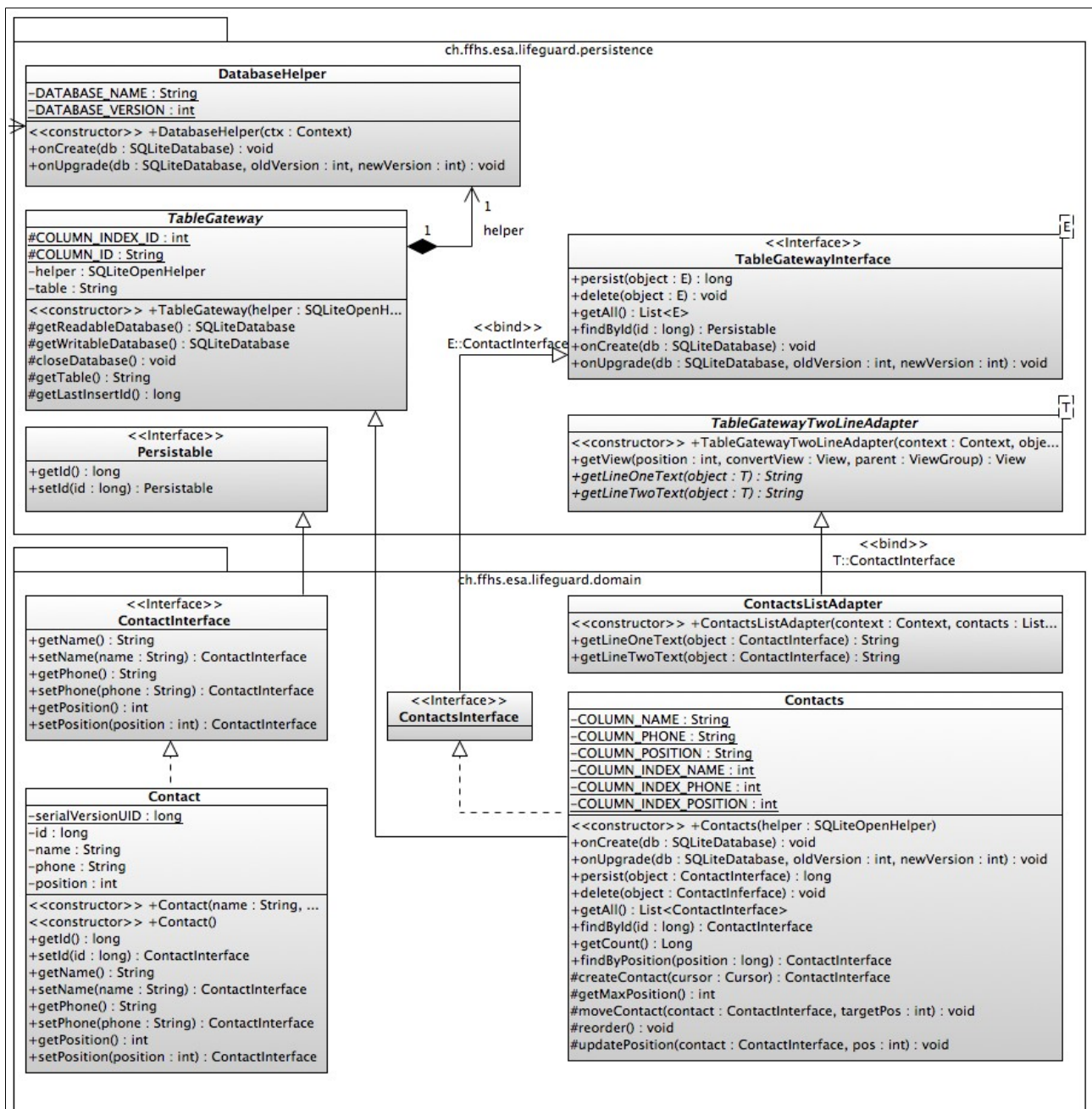


Illustration 5: Klassendiagramm Persistenz

# Bedienkonzept

---

## Anforderungen an das Bedienkonzept

Das Zielpublikum sind ältere oder hilfsbedürftige Menschen, die Fähigkeiten mit technologischen Hilfsmitteln umzugehen muss daher in Frage gestellt werden. Somit soll die Benutzerschnittstelle so einfach wie möglich gehalten werden.

- Maximal 4-5 von Steuerelementen pro Anzeige.
- Maximal 3-4 Ebenen in der View-Hierarchie.
- Grosse Bedienelemente.

## Hierarchie

Die Hierarchie soll gemäss den Anforderungen eine maximale Tiefe von 3-4 Ebenen aufweisen. Die einzige Aktion, welche mehr als 1 Ebene von der Haupt-Aktivität (Hauptansicht) entfernt ist, ist das Erfassen resp. Editieren von Empfängern. Somit kann eine Hierarchie mit lediglich 3 Stufen erstellt werden.

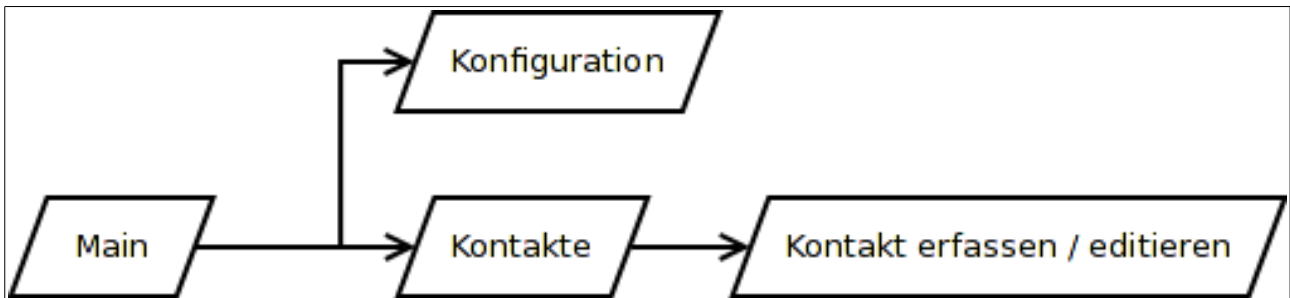


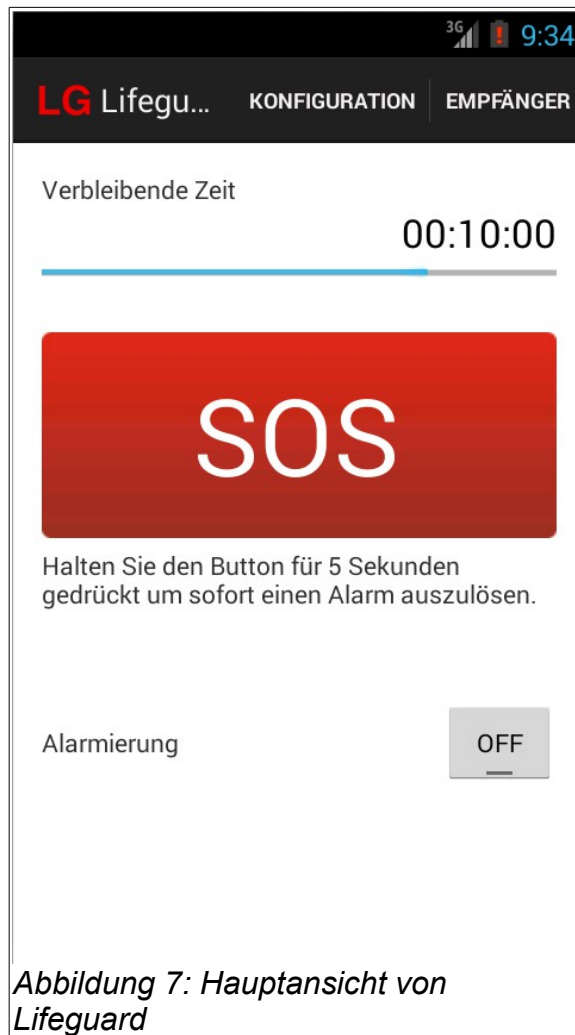
Abbildung 6: Hierarchie der Views

## Hauptansicht

Die Hauptansicht soll den Kern der Applikation, resp. dessen allernötigsten, resp. das Allernötigste Steuerelement präsentieren. In minimaler Betrachtung, um den Anforderungen gerecht zu werden können die folgenden drei unterschiedlichen Aktivitäten genannt werden:

- Verbleibende Zeit darstellen (visualisiert in Form eines Fortschrittsbalkens).
- Steuerung zum sofortigen Auslösen des Alarms (UC6 Alarm auslösen).
- Alarmierung ein- oder ausschalten (UC4 Alarmierung aktivieren oder deaktivieren).

## Ansicht



*Abbildung 7: Hauptansicht von Lifeguard*

## Kontaktliste

Die Kontaktliste soll das Verwalten der Kontakte ermöglichen. Somit soll es möglich sein, einen bestehenden Kontakt zu editieren, einen neuen Kontakt zu erstellen sowie die Reihenfolge der Kontakte zu verändern. In der Darstellung der Liste selbst sollen daher Möglichkeiten bestehen, das Erzeugen oder Editieren eines Kontaktes auszulösen.

- Wechsel zu Kontakt editieren (Klick) (UC2 Empfänger editieren).
- Neuen Kontakt erstellen (UC1 Empfänger erfassen).
- Priorität festlegen (Umsortieren).



## Ansicht

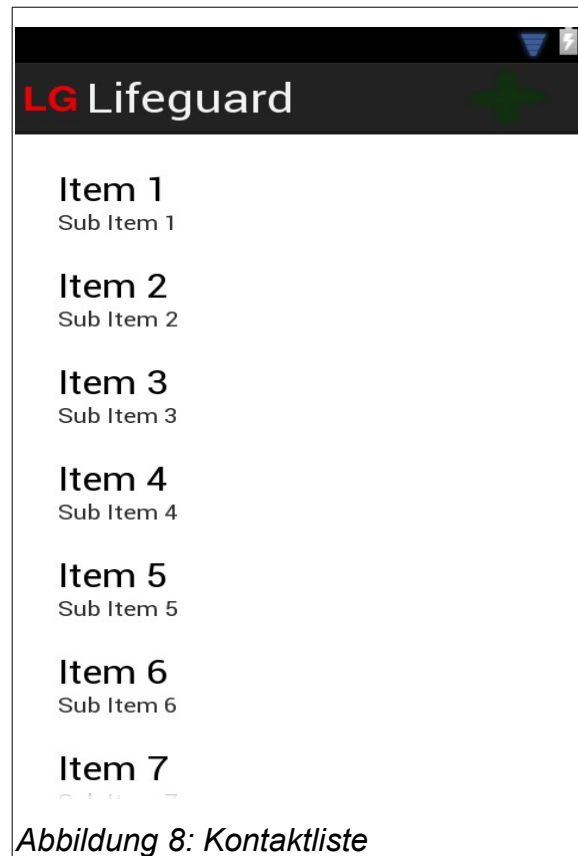


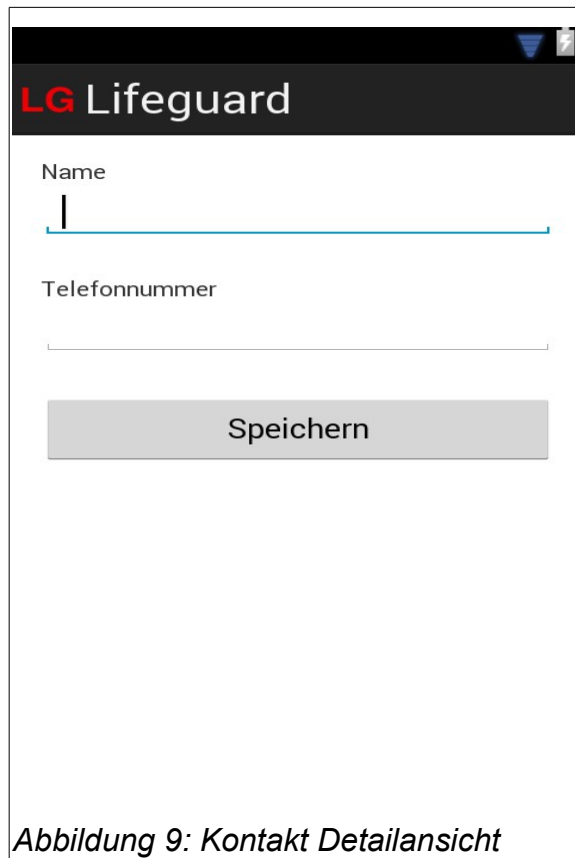
Abbildung 8: Kontaktliste

## Kontakt Detailansicht

Die Detailansicht soll es ermöglichen, einen bestehenden Kontakt zu editieren oder einen neuen Kontakt zu erfassen, für die beiden verwandten Aufgaben kann sehr gut dieselbe Ansicht verwendet werden, da genau dieselben Informationen hinterlegt werden. Die folgenden Attribute werden in dieser Ansicht gesetzt:

- Name des Empfängers
- Telefonnummer

## Ansicht



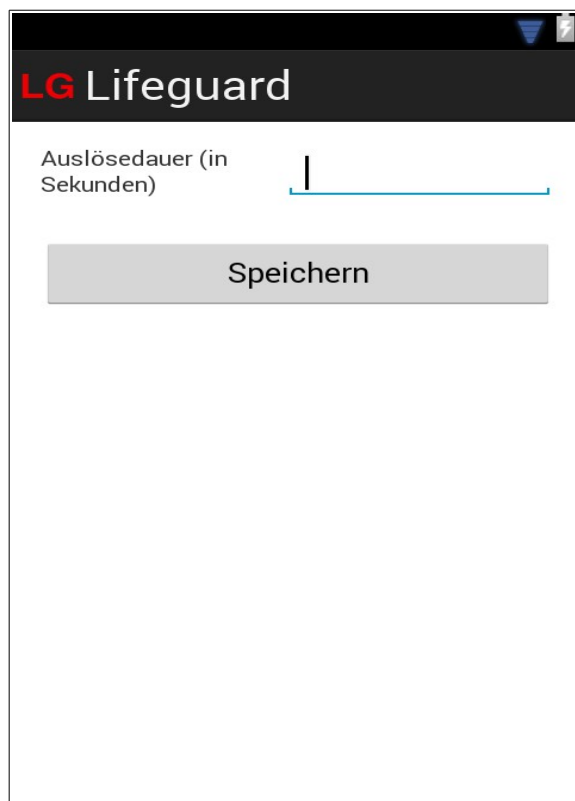
The screenshot shows the 'LG Lifeguard' app interface. At the top is a black header with the 'LG Lifeguard' logo in red and white. Below the header, there are two input fields: 'Name' and 'Telefonnummer'. The 'Name' field contains a single character 'I'. Below these fields is a grey button labeled 'Speichern'.

Abbildung 9: Kontakt Detailansicht

## Konfiguration

Die Applikation weist lediglich einen Parameter auf, welcher das Verhalten der Applikation verändert. Es handelt sich um die Auslösedauer. Die Auslösedauer gibt an, wie lange der Benutzer (Hilfsbedürftige) inaktiv sein kann, ohne dass der Alarm ausgelöst wird. Sobald der Benutzer länger als die konfigurierte Auslösedauer inaktiv ist, wird der Alarm ausgelöst.

## Ansicht



The screenshot shows the 'LG Lifeguard' app interface. At the top is a black header with the 'LG Lifeguard' logo in red and white. Below the header, there is a single input field labeled 'Auslösedauer (in Sekunden)'. The field contains a single character 'I'. Below this field is a grey button labeled 'Speichern'.

# Testkonzept

---

## Verwendete Tests

### Automatisierte Tests

Der offiziellen Empfehlung<sup>2</sup> folgend, werden die programmierten Tests in einem eigenen Projekt<sup>3</sup> umgesetzt, welches das Applikations-Projekt als Abhängigkeit enthält.

Das Android-Testing-Framework unterstützt vielfältige Tests, insbesondere das Testing von Activities und Services, sowie automatisiertes UI-Testing mittels uiautomator-Framework. Domänen-Modelle sind POJOs und werden mit JUnit-Test-Cases geprüft.

Die Test-Cases werden der Paketstruktur des Hauptprojekts folgend im Testprojekt abgelegt und können einzeln oder in Suiten gruppiert ausgeführt werden. Eine automatisierte Ausführung der Tests mit einem Continuous-Integration-Server wäre möglich, wird in diesem Projekt jedoch nicht umgesetzt.

### Manuelle Tests

Das Vorgehen beim manuellen Testing wird durch die Anwendungsfallbeschreibungen vorgegeben. Ausschlaggebend für das Bestehen eines Tests ist die Erfüllung der jeweiligen Nachbedingungen.

### Bugtracking

Während der Testphase «Abnahmetests» aufgedeckte Fehler werden im Issue-Tracker des Git-Repositories des Hauptprojekts<sup>4</sup> eingetragen.

## Testphasen

Aufgrund des geringen Projektumfangs werden lediglich zwei Testphasen definiert:

### Entwicklertests

Die Entwickler sind dazu angehalten, den Testcode zeitnah zu ihren Implementierungen zu schreiben. Somit lässt sich der Zustand der Applikation laufend überprüfen.

Gesonderte Integrationstests sind vorerst wegen der geringen Anzahl an Komponenten nicht vorgesehen. Durch die Verwendung von Git ist jeder Entwickler im Besitz einer vollwertigen Kopie der Codebase und kann die Applikation jederzeit lokal testen. Allfällige Feature-Branches anderer Entwickler können nach deren Freigabe in die eigenen Branches gemerged werden.

Sollte es wider Erwarten zu Integrationsproblemen kommen, wird das Testkonzept entsprechend erweitert.

---

<sup>2</sup> Testing from Eclipse with ADT, [http://developer.android.com/tools/testing/testing\\_eclipse.html](http://developer.android.com/tools/testing/testing_eclipse.html) (abgerufen 16.11.2013)

<sup>3</sup> <https://github.com/FFHS-Inf-2010-students/LifeguardTest>

<sup>4</sup> <https://github.com/FFHS-Inf-2010-students/Lifeguard/issues>

## Benutzertests vor Abnahme

Um das Testverfahren nicht unnötig kompliziert zu gestalten, werden die Use Cases im echten Einsatz getestet. Vor allem der Umgang mit SMS Ein- und Ausgängen lässt sich so einfacher testen.

## Abnahmetests

Ist die Applikation *feature complete*, wird sie anhand der definierten Use Cases abschliessend geprüft. Die Abnahme erfolgt, wenn:

- Alle Testfälle ohne Fehler durchlaufen
- Keine fehlerhaften Abweichungen von den in den Anwendungsfällen definierten Abläufen festzustellen sind.

Solange nicht beide Bedingungen erfüllt sind, wird die Applikation durch Bugfixing iterativ an die *general availability* herangeführt.

## Vollständige Liste der Testfälle (automatisch und manuell)

### Automatische Tests

- ConfigurationActivityTest
  - testLayout
  - testForm
  - testCancel
  - testSave
- ContactDetailActivityTest
  - testLayout
  - testContact
  - testValidate
  - testCancelButton
  - testFailedSaveAttempt
  - testSaveContact
- ContactListActivityTest
  - testItemClick
- ContactsTest
  - testPersist
  - testDelete
  - testGetAll
  - testFindById
- ContactTest

- testToString
- testGetSetId
- testGetSetName
- testGetSetPhone
- testGetPosition
- LifeguardTest
- MainActivityFunctionalTest
  - testStartOfConfigurationActivity
  - testStartOfContactListActivity
- MainActivityTest
  - testLayout
  - testOpenConfiguration
  - testViewContacts
- DatabaseHelperTest
  - testOnCreateSQLiteDatabase
  - testOnUpgradeSQLiteDatabaseIntInt
- AlarmServiceTest
  - testOnCreate
  - testOnStart
  - testOnStateChanged
  - testOnBindIntent
- ServiceAlarmContextTest
  - testSetNext
  - testGetState
  - testAddListener
- AlarmingStateTest.java
  - testGetId
- AwaitingStateTest.java
  - testGetId
- ConfirmedStateTest.java
  - testGetId
- InitialStateTest.java
  - testGetId
- TickingStateTest.java

- testGetId

## Manuelle Tests

Um den Aufwand der Testentwicklung auf ein vertretbares Mass zu beschränken, werden die „high-level“ Benutzertests manuell durchgeführt:

- **UC6 Alarm auslösen**

Der Benutzer/Hilfsbedürftige löst selber einen Alarm aus oder der Timer läuft ab, wenn der Bewegungssensor keine Aktivität meldet

- **UC7 SMS empfangen**

Der erste konfigurierte Empfänger erhält ein SMS. Dieses wird nicht quittiert, damit ein weiterer konfigurierter Empfänger (kann auch die gleiche Handynummer sein) die SMS erhält.

- **UC8 SMS quittieren**

Der letzte Empfänger quittiert die SMS und die Alarmierung stoppt.

## Changelist Dokumentation

Datum	Task	Autor	Reviewer	Bemerkungen
07.09.2013	Dokument erstellen	David Daniel		
07.09.2013	Use-Case-Beschreibungen verfassen und einfügen	David Daniel	Alle	
11.09.2013	Use-Case-Diagramm erstellen und einfügen	Jürg Gutknecht	Alle	
19.09.2013	Verteilungsdiagramm erstellen	Christof Kälin	Alle	
19.09.2013	Verteilungsdiagramm einfügen und Dokument formatieren	Jürg Gutknecht		
20.09.2013	Verteilungsdiagramm überarbeiten	Christof Kälin		
13.10.2013	Changelist erstellen; Feedback vom 5.10.2013 umsetzen	Jürg Gutknecht		
18.10.2013	Bedienkonzept erstellen	David Daniel	Alle	
5.11.2013	Architektur System und Services erstellen (Entwerfen, implementieren, dokumentieren)	David Daniel	Alle	
16.11.2013	Architektur Datenpersistenz erstellen (Entwerfen, implementieren, dokumentieren)	Jürg Gutknecht	Alle	
16.11.2013	Testkonzept schreiben	Jürg Gutknecht		
17.11.2013	SVGs durch PNGs aus Visual Paradigm ersetzen	Jürg Gutknecht		1. Einheitlichkeit 2. Darstellung der SVGs mit riesiger Schrift
21.12.2013	Ergänzung mit kompletter Testfallliste und kleinere Korrekturen	Christof Kälin		