CENTRADO Android App

Studienarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Herbstsemester 2011

Autoren: Marco Pfiffner, Mathias Fasser
Betreuer: Prof. Hans Rudin
Projektpartner: CENTRADO GmbH, Crealogix AG
Experte: Marcel Keller, Crealogix AG
Gegenleser: -

HSR Hochschule für Technik Rapperswil
Oberseestrasse 10
Postfach 1475
CH-8640 Rapperswil
www.hsr.ch

Erklärung der Eigenständigkeit

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben haben.

Ort, Datum: Rapperswil, 19.12.2011

Name, Unterschrift: Marco Pfiffner 

Mathias Fasser 

1 Inhaltsverzeichnis

2 Abstract

Die Firma CENTRADO bietet ein System an, welches Menschen hilft, ihre Work-Life-Balance zu verbessern. Um dies zu erreichen muss ein Kunde im Vorfeld ein Screening ausfüllen, welches Aufschluss über verschiedene Bereiche seines Lebens festhält. Anhand dieses Screenings werden dem Kunden passende Impulse zugesendet. Diese Arbeit ermöglicht es CENTRADO Kunden in Zukunft, ihre Impulse auch auf einem Android-Phone zu empfangen.

Die CENTRADO App ist einem Mail App sehr ähnlich. Sie stellt eine Plattform zur Verfügung, auf der Impulse verwaltet und gelesen werden können. In einer Impulsliste werden alle, bis anhin empfangenen, Impulse dargestellt. In einer Detail Ansicht kann der Inhalt des jeweiligen Impulses gelesen werden. Zusätzlich zu einem Text kann der Impuls Bilder, Bildfolgen und Videos enthalten.

Impulse treffen per Push auf dem Smartphone ein, dazu wird das Android Cloud to Device Messaging (C2DM) Framework verwendet. C2DM ist ab Android Version 2.2 verfügbar und die derzeit einzig verfügbare Push-Technologie unter Android.

Von einem Server können Impulse an das Android App gesendet werden. Dazu werden die Impulse in ein JSON-Format gebracht. Die JSON-Dateien werden über eine REST-Schnittstelle an das App gesendet. Der Server wird nicht bis zur Produktreife entwickelt, er dient während der Arbeit zu Testzwecken.

3 Management Summary

3.1 Ausgangslage

Die Firma Centrado hat ein Work-Life-Balance System entwickelt, welches Menschen dabei unterstützt, ihr Leistungspotenzial besser auszunutzen. In Zusammenarbeit mit der Firma Crealogix wurde ein Webcockpit entwickelt, auf dem die Kunden ein persönliches Screening ausfüllen können. Dabei müssen Fragen zu Familie, Ernährung, Sport, Spiritualität und anderen Bereichen beantwortet werden. Mithilfe dieses Screenings können die Problemgebiete der einzelnen Kunden festgestellt werden. Anhand des Screenings wählt das System passende Impulse aus, welche den Kunden in seiner Work-Life-Balance unterstützen und sendet die Impulse an den Kunden. Die Impulse können aus Text, Bildern, Bildfolgen und Videos bestehen.

In einer Bachelorarbeit an der HSR wurde eine iPhone App entwickelt, welche das Empfangen von Impulsen auf dem iPhone ermöglicht. Ebenfalls wurde ein Testserver implementiert, welcher die Webschnittstelle zur iPhone App simuliert.

In dieser Semesterarbeit wurde eine App entwickelt, welche das Empfangen von Impulsen auf Android Geräten ermöglicht. Die Android App sollte den gleichen Funktionsumfang wie die iPhone App bieten. Die REST-Schnittstelle des Testservers konnte von der Bachelorarbeit übernommen und angepasst werden.

3.2 Vorgehen

Während des gesamten Projekts wurde ein iterativer Entwicklungsprozess verfolgt. Zu Beginn der Arbeit wurden Use Cases definiert, welche den Umfang der Arbeit aufzeigten.
Während der Arbeit wurden optionale Anforderungen definiert, welche jeweils in einer Diskussion mit dem Auftraggeber priorisiert wurden. Anhand der Priorisierung wurden diese Anforderungen umgesetzt. Der Auftraggeber legte jedoch viel Wert auf „Qualität vor Quantität“. Dadurch wurden die optionalen Anforderungen erst umgesetzt, sobald die spezifizierten Use Cases Grossteils abgeschlossen waren.

Während der gesamten Arbeit wurden wöchentliche Meetings mit dem Auftraggeber abgehalten, in denen der aktuelle Stand diskutiert und das weitere Vorgehen besprochen wurde. Zusätzlich wurde das Erscheinungsbild der App laufend verbessert, da dies eine wichtige Anforderung an die App war.

Um der Problematik der Gerätevielfalt in Android entgegen zu wirken, wurden umfangreiche Systemtests auf Geräten mit unterschiedlichen Auflösungen und Android Versionen durchgeführt.

3.3 Technologie

Bei Projektbeginn war die Einarbeitung in Android sehr wichtig. Anschliessend wurde der Push-Mechanismus von Android analysiert. Denn sobald für den Kunden ein passender Impuls auf dem Server generiert wurde, wird dieser per Push-Mechanismus dem Kunden gesendet.

Die persistente Speicherung von Impulsen auf dem App wurde mit SQLite realisiert. Der Testserver wurde parallel mit dem App entwickelt. Das Versenden von Impulsen auf dem Server wurde mit Servlets umgesetzt.

3.4 Ergebnisse

Das Endprodukt der Arbeit ist eine Android App, welche den Empfang, das Verwalten sowie die Betrachtung von Impulsen bietet. Es können Impulse mit unterschiedlichen Medieninhalten angezeigt werden.

Die App ist ab Android Version 2.2 auf allen Geräten ausführbar und bietet auf allen Auflösungen ein ansprechendes externes Design. Als optionale Anforderungen wurde der Umgang mit der Impulsliste verbessert. Der Benutzer des Apps kann nach Impulsen suchen, Impulse als Favoriten markieren, sowie die Impulse filtern.

Nebst der App wurde ein Testserver entwickelt, sodass die Kommunikation von App und CENTRADO-Business-Logik, sowie das Senden von Impulsen simuliert werden konnte.

Die Abbildung 1: Gesamtübersicht zeigt eine Übersicht, der bereits bestehenden Komponenten (grau). Die neu entwickelten Komponenten und Schnittstellen sind in grün dargestellt. Dabei ist zu beachten, dass die REST-Schnittstelle, der Media Server, sowie die Android Push Technologie (C2DM) auf dem Testserver simuliert wurden.

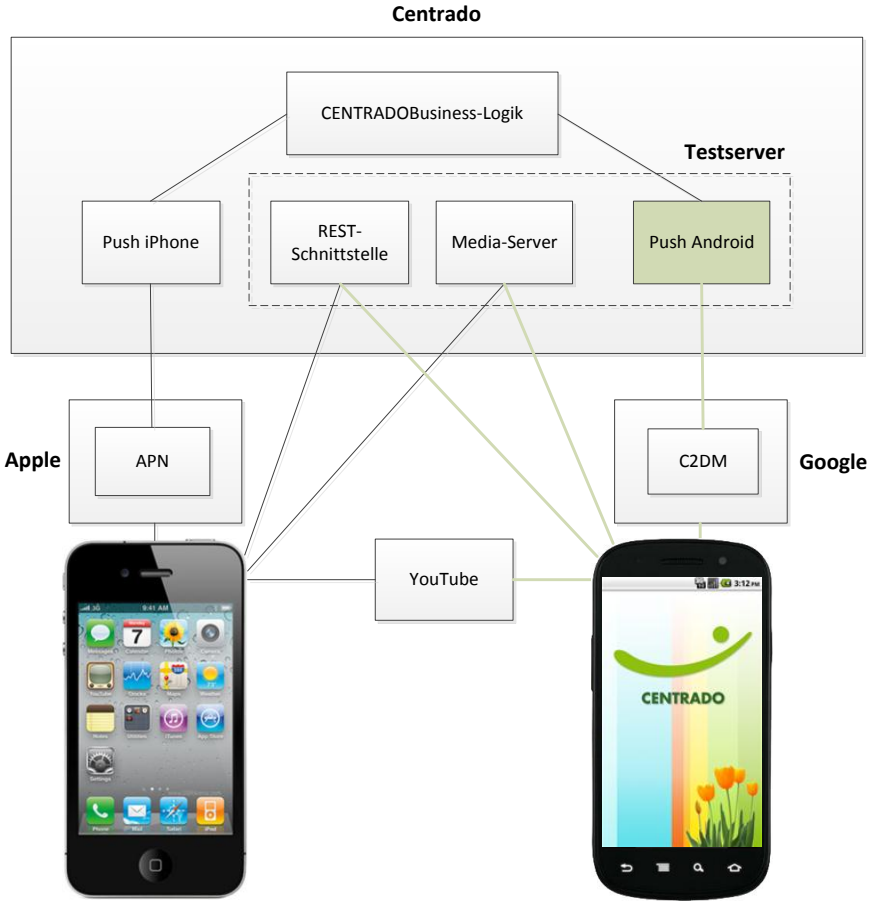

Abbildung 1: Gesamtübersicht

3.5 Ausblick

Aufgrund der Verwendung des Testservers müssen die Schnittstellen zwischen App und CENTRADO-Business-Logik für den produktiven Einsatz angepasst und getestet werden.
Ebenfalls muss die App von der Firma Crealogix auf das Deployment in den Android Market vorbereitet werden. Dies umfasst das Signieren der App mit einem Zertifikat, sowie das Einbinden von Herstellerinformationen mit Screenshots der App.

4 Anforderungsanalyse

4.1 Einleitung

Grosse Teile der Anforderungsanalyse wurden aus der Bachelorarbeit CENTRADO App für iPhone übernommen und, wo nötig, angepasst. Die kursiven Absätze stammen aus der Bachelorarbeit [BAiPhone].

4.2 Personas & Szenarios

Bei dieser Arbeit wurden Designvorschläge der GUIs von Crealogix zur Verfügung gestellt. Die Anforderungsanalyse konnte zu grossen Teilen von der iPhone App übernommen werden.

4.2.1.1 Person 1

- *Er nutzt nur das Auto - auch für den Arbeitsweg, er fährt kaum mit der Bahn.*
- *Das iPhone ist immer an. An Wochenenden ist sein iPhone jedoch immer öfter abgeschaltet.*
- *Nutzt die iPhone Code-Sperre nicht - er fand's jedoch einen guten Tipp und wird es in nächster Zeit anschauen.*
- *Störendes an Apps: 20min App hat zu viele News, die mit Push Notifications mitgeteilt werden: Das dauernde Piepsen stört ihn ziemlich.*

4.2.1.2 Person 2

- *Er benutzt für unterwegs jeweils das Auto (Arbeitsweg und Kundentermine).*
- *Störendes an Apps: Swisstraffic: Sie ist penetrant, bringt Meldungen und Geräusche, Popups. Er hat die Push Notifications ausgeschaltet, das hat aber nur teilweise etwas gebracht.*
- *Push Notifications hat er auch nicht sehr gerne. Hat lieber selbst die Kontrolle, als sich vom iPhone steuern zu lassen.*

4.3 Use Cases

4.3.1 Übersicht Grundfunktionalitäten

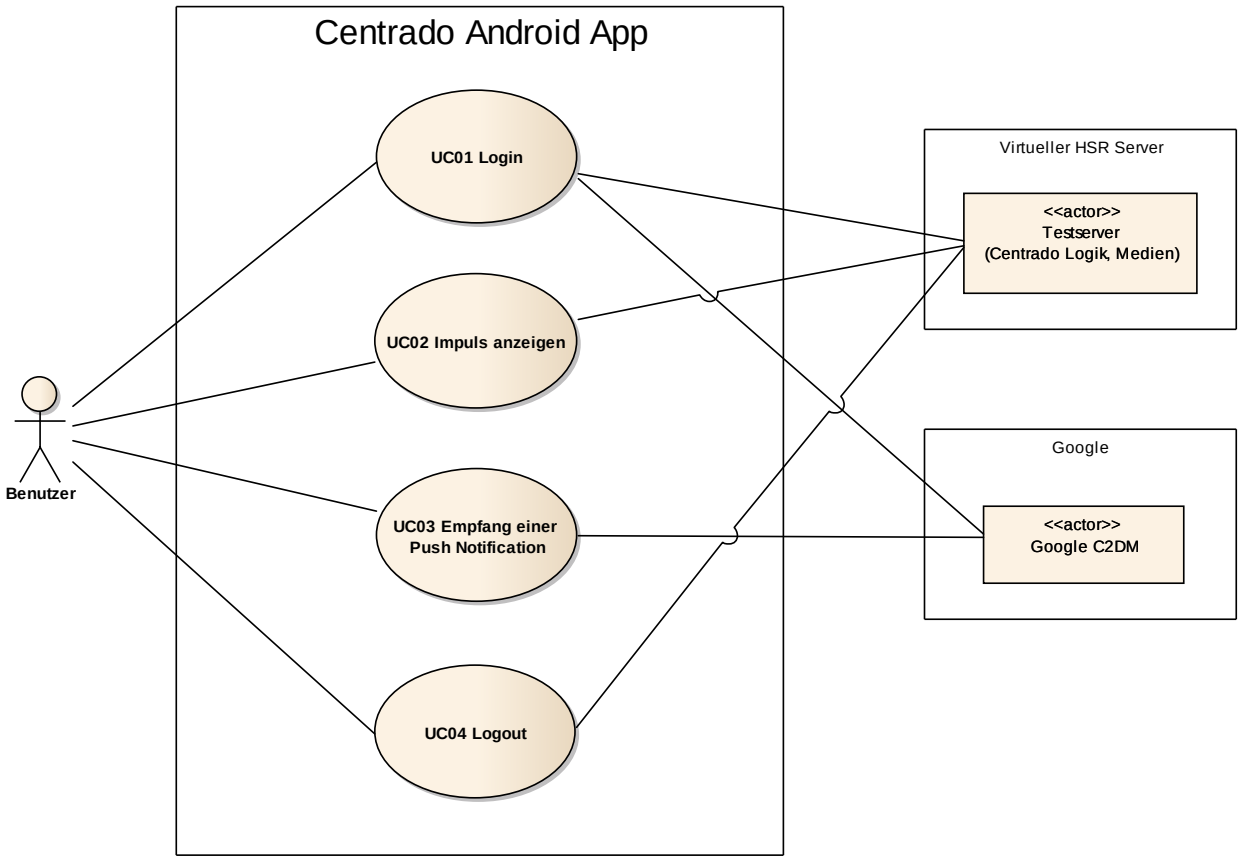

Abbildung 2: Use Case Diagramm

4.3.2 Aktoren

Aktor	Beschreibung
Benutzer	Benutzer der CENTRADO Android App
Test-Server	Server mit Impulslogik und Medieninhalten (Bilder, Videos usw.)
Google C2DM	Google C2DM Server Infrastruktur für Push-Notifications

Tabelle 1: Aktoren

4.3.3 Use Case Beschreibungen

4.3.3.1 UC01: Login

Use Case ID	UC01
Use Case Name	Login
Umfang	Android App, Test-Server, Google C2DM
Ebene	Benutzer
Aktor	Benutzer
Überblick	Der Benutzer startet die App und gibt seine Login-Daten ein.
Stakeholder und Interessen	Der Benutzer will Zugriff auf die App und seine Inhalte.
Vorbedingungen	Benutzer hat einen Centrado-Account
Nachbedingungen	Die Login-Daten sind gültig und in der App verfügbar.
Auslöser	Start der App
Standardablauf	1. App überprüft, ob Login-Daten gespeichert sind (ist der Benutzer bereits am App angemeldet) 2. App fordert Login-Daten an 3. App überprüft bei CENTRADO, ob Login-Daten OK sind 4. App fordert Device-Token bei Google an (anhand Google-Account) 5. App teilt den Device-Token dem Test-Server mit 6. App zeigt Impuls-Inbox an
Bemerkungen	Die App muss keine Neuregistrierungen von Benutzern unterstützen.
Erweiterungen	**3a) Login-Daten falsch** 1. App zeigt Hinweis, dass Login-Fehler aufgetreten 2. Hauptablauf weiter bei Schritt 1 **4a) Google-Services nicht verfügbar** 1. App zeigt Hinweis, dass Push nicht funktioniert 2. Hauptablauf weiter bei Schritt 6 **4b) Kein Google-Konto vorhanden** 1. App zeigt Hinweis, dass Push nicht funktioniert 2. Hauptablauf weiter bei Schritt 6

Tabelle 2: UC01:Login

4.3.3.2 UC02: Impuls anzeigen

Use Case ID	UC02
Use Case Name	Impuls anzeigen
Umfang	Android App, Test-Server
Ebene	Benutzer
Aktor	Benutzer
Überblick	Der Benutzer wählt einen Impuls aus seiner Liste aus und möchte die Detailansicht des Impulses sehen.
Stakeholder und Interessen	Der Benutzer will Zugriff auf die Inhalte eines Impulses
Vorbedingungen	Benutzer hat sich erfolgreich angemeldet
Nachbedingungen	Der Benutzer hat die Inhalte des Impulses gesehen.
Auslöser	Auswahl eines Impulses
Standardablauf	1. App lädt Impulse von der REST-Schnittstelle herunter 2. App zeigt Liste empfangener Impulse an 3. Benutzer tippt ein Impuls aus der Liste an 4. App lädt Detailansicht mit passendem Logo (Bild, Video, Text) 5. App zeigt Detailansicht
Bemerkungen	Die App muss keine Neuregistrierungen von Benutzern unterstützen.
Erweiterungen	**6) Benutzer klickt auf Medien-Button (Bild, Video)** 1. Medien-Inhalt wird wiedergegeben

Tabelle 3: UC02 Impuls anzeigen

4.3.3.3 UC03: Empfangen einer Push-Notification

Use Case ID	UC03
Use Case Name	Empfangen einer Push-Notification
Umfang	Android App, Test-Server, Google C2DM
Ebene	Benutzer
Aktor	Benutzer
Überblick	Der Benutzer erhält eine Benachrichtigung, dass ein neuer Impuls für ihn bereit steht.
Stakeholder und Interessen	Der Benutzer will benachrichtigt werden, wenn es neue Impulse für ihn gibt.
Vorbedingungen	• Benutzer hat einen Google-Account registriert auf dem Device • Benutzer hat das App bereits einmal erfolgreich gestartet
Nachbedingungen	Der Benutzer hat den neuesten Impuls gesehen.
Auslöser	• Zuweisung eines neuen Impulses auf dem Server. • Push initiieren bei C2DM
Standardablauf	1. Gerät zeigt Notification in der Status-Bar an 2. Benutzer klickt auf die Notification 3. Impuls-Liste wird neu geladen 4. App zeigt neuesten Impuls UC02
Bemerkungen	Die App muss keine Neuregistrierungen von Benutzern unterstützen.
Erweiterungen	**2a) Benutzer ignoriert Notification** 1. Notification bleibt in der Status-Bar 2. Hauptablauf weiter bei 1 **2b) Benutzer löscht Notification** 1. Abbruch von UC03

Tabelle 4: UC03 Empfangen einer Push-Notification

4.3.3.4 UC04: Logout

Use Case ID	UC04
Use Case Name	Logout
Umfang	Android App, Testserver
Ebene	Benutzer
Aktor	Benutzer
Überblick	Der Benutzer beendet die App und loggt sich explizit aus.
Stakeholder und Interessen	Der Benutzer will den Zugriff auf die App und seine Inhalte sperren.
Vorbedingungen	Benutzer hat sich erfolgreich eingeloggt.
Nachbedingungen	Die App und seine Inhalte sind vor fremden Zugriffen geschützt
Auslöser	Logout-Aktion wird ausgeführt.
Standardablauf	1. App löscht Login-Daten von Android Device 2. App wird beendet
Bemerkungen	Die Logout-Aktion kann im Menü getätigt werden.

Tabelle 5: UC04:Logout

4.4 Optionale Anforderungen

Während der Arbeit sind laufend neue Ideen vom Auftraggeber eingegangen. Diese Ideen wurden jeweils diskutiert und in die Liste der optionalen Anforderungen aufgenommen. Es war wichtig, dass die optionalen Anforderungen priorisiert wurden. Zudem wurde jeweils vom Projektteam eine Aufwandschätzung abgegeben. Anforderungen, bei denen der Aufwand mit einem „?“ markiert wurde, konnten nicht geschätzt werden, da dem Projektteam die Erfahrung in diesem Bereich fehlte.

Optionale Anforderung	Aufwand	Priorität	Grund für Priorität
Favoriten / Merkliste	Hoch	1	Wurde vom Auftraggeber am höchsten priorisiert
Suche	Mittel	1	Ist bei iPhone App auch vorhanden (jedoch nicht als Use Case aufgeführt)
Gelesen / Ungelesen	Hoch	1	Ist bei iPhone App auch vorhanden (jedoch nicht als Use Case aufgeführt)
Liste sortieren	Gering	1	Hilft dem User beim Arbeiten mit der Liste
Filtern (Favorite, Ungelesene / Gelesene)	Hoch	1	Hilft dem User beim Arbeiten mit der Liste
Horizontale Ansicht der App	Hoch	1	Macht App flexibler und somit ansprechender
Jingle	Hoch	1	Macht die App spezieller
Hilfe Icon (auf LoginScreen)	Gering	1	Unterstützt User bei Problemen
Impulse als Termine im Kalender eintragen	?	2	Technik für diese optionale Anforderung ist dem Projektteam noch nicht bekannt.
Erinnerung für Impuls (Snooze)	?	2	Technik für diese optionale Anforderung ist dem Projektteam noch nicht bekannt.
Bewertung von Impulsen („Gefällt mir“, +1, Smiley)	Hoch	3	Schnittstelle müsste geändert werden
Wie fühlst du dich im Moment?	Hoch	3	Schnittstelle müsste geändert werden
Notizen, Tagebuch Funktionalität	Hoch	3	Schnittstelle müsste geändert werden
Widget (Zähler für ungelesene Impulse)	Hoch	3	Technik für diese optionale Anforderung ist dem Projektteam noch nicht bekannt.
Tablet-Unterstützung (Fragments, Action-Bars usw.)	Hoch	3	Würde Rahmen der SA sprengen

Tabelle 6: Optionale Anforderungen

4.4.1 Optionale Anforderungen (Priorität 1)

4.4.1.1 Favoriten / Merkliste

Das Prinzip der Favorisierung ist aus vielen Anwendungen bekannt. Es ermöglicht dem Benutzer, meistens Listeneinträge, speziell zu markieren, um sie somit vom Rest abzuheben. Dem Benutzer des CENTRADO Android Apps sollte die Möglichkeit geboten werden, seine Lieblingsimpulse speziell markieren zu können.

4.4.1.2 Suche

Wie in den meisten Apps mit einer Liste sollte auch hier die Möglichkeit bestehen, die Liste nach speziellen Impulsen durchsuchen zu können.

4.4.1.3 Gelesen / Ungelesen

Da die App vom Erscheinungsbild her sehr stark einem Mailclient gleicht, sollte dem Benutzer die Information angezeigt werden, ob ein Impuls bereits gelesen, oder noch ungelesen ist.

4.4.1.4 Liste sortieren

Eine weitere Möglichkeit, um dem Benutzer die Arbeit mit einer Liste zu erleichtern, ist die Sortierung. Diese Idee stammt vom Projektteam.

4.4.1.5 Filterung

Durch die Möglichkeit Impulse als Favoriten zu markieren und zusätzlich anzuzeigen, ob der Impuls gelesen/ungelesen ist, bietet sich die Möglichkeit an, die Liste nach diesen Kriterien zu Filtern. Dies unterstützt den Benutzer, um schneller an die richtigen Impulse zu gelangen.

4.4.1.6 Horizontale Ansicht der App

Viele Apps bieten die Möglichkeit, auch im horizontalen Modus benutzt zu werden.

4.4.1.7 Jingle

Ein neuer CENTRADO Impuls, welcher per Push-Mechanismus beim Kunden angezeigt wird, sollte von anderen Meldungen, welche auf dem Android Gerät eingehen, abgehoben werden. Durch einen Jingle, der beim Empfang des Impulses abgespielt wird, weiss der Kunde, dass er einen CENTRADO Impuls erhalten hat.

4.4.1.8 Hilfe Icon (auf Login Screen)

Um den CENTRADO Dienst nutzen zu können, muss nur auf der Firmenwebseite dafür bezahlt werden. Die App wird in naher Zukunft auf dem Android Market gratis angeboten. Somit könnte der Fall eintreten, dass jemand die App herunterlädt, ohne als Benutzer registriert zu sein. Folglich hätten diese Personen keinen Login und könnten die App nicht verwenden. Für diesen Fall sollte auf dem Login Screen die Möglichkeit bestehen, sich Hintergrundinformationen zur heruntergeladenen App zu beschaffen.

4.4.2 Optionale Anforderungen (Priorität 2 und 3)

Die optionalen Anforderungen mit Priorität 2 sind als Vision einzustufen. Bei einigen dieser optionalen Anforderungen hätte die Schnittstelle angepasst werden müssen, was während der Arbeit sowieso nicht möglich war. Diese Ideen wurden jedoch trotzdem dokumentiert, da sie die Grundlage für allfällige Erweiterungen des CENTRADO Android Apps darstellen.

4.5 Nichtfunktionale Anforderungen

Die nichtfunktionalen Anforderungen wurden ebenfalls aus der Anforderungsanalyse der iPhone App übernommen. Nicht relevante Anforderungen wurden mit „*nicht relevant* „ gekennzeichnet. Die Nummerierung erfolgt analog zur iPhone App.

4.5.1 Benutzbarkeit

- NF01: Die Android App soll in der Bedienung intuitiv sein. Es darf keine Schulung erforderlich sein. Wenn nötig, werden Hinweise und Kontexthilfe in *der App eingebaut.*
- NF02: Best Practice: Anlehnung an bestehende/bewährte Designs für Android Apps

4.5.2 Attraktivität

- *NF03: Die App soll ansprechend sein.*
- *NF04: Die App wird nur von Benutzern verwendet, die ein Abo haben. Die App wird nicht verwendet, um neue Kunden anzuwerben.*

4.5.3 Sicherheit

- *NF05: Alle Zugriffe auf persönliche Daten des Benutzers sollen nur mit Login möglich sein.*
- NF06: Übertragungen zwischen Server und Android Phone sollen immer verschlüsselt werden.
- *NF07: Der Zugriff auf Medien (Bilder, Videos etc.) muss weder verschlüsselt noch autorisiert (Login) erfolgen.*
- *NF08: Nach einem Logout darf nicht mehr auf persönliche Daten zugegriffen werden können.*
- NF09: Die persönlichen Daten (Benutzername und Passwort) sollen angemessen geschützt sein, auch wenn das Android Phone verloren oder gestohlen wird. Impulse sind nicht speziell gesichert, es sind keine „persönlichen Daten“.

4.5.4 Reife

- *NF10: Der Funktionssatz „Anzeigen von Impulsen inklusive deren lokale Speicherung“ soll bis zur Produktionsreife entwickelt werden.*
- *NF11: Die Qualität ist wichtiger als der Funktionsumfang.*

4.5.5 Wartbarkeit

- *NF12: Der gesamte Programm-Code, sowie die Dokumentation im Code sollen in Englisch verfasst sein.*
- *NF13: Es soll, soweit vorhanden, der Coding Style von Crealogix übernommen werden. Ansonsten sind die üblichen Standards zu verwenden.*
- *NF14: Wo sinnvoll (Aufwand und Nutzen), sollen Unit-Tests geschrieben werden.*
- NF22: Einfaches Hinzufügen zusätzlicher Sprachfiles. (Standardsprache: Deutsch)

4.5.6 Konformität

- NF15: nicht relevant

4.5.7 Portabilität

- NF16: Die App soll für das Android Phone erstellt werden. Eine Tablet-Version ist nicht gefordert.
- NF17: Für den Datenaustausch zwischen Android Phone und Serverkomponenten soll JSON verwendet werden. (Gleich wie beim iPhone)
- *NF18: Für die Anmeldung am Server soll http Basic Authentication1 (mit SSL) verwendet werden.*

4.5.8 Umgebung (Vorgaben)

- NF19: Die App soll auf Android Geräten mit Version 2.2 (C2DM) oder höher ausführbar sein.
- NF20: nicht relevant
- *NF21: Alle Serverkomponenten sollen in Java 1.6 geschrieben werden.*

5 Domainanalyse

Die Domainanalyse wurde bereits in der Bachelorarbeit gemacht [BAiPhone] und betrifft hauptsächlich den Server. Da die Serverschnittstelle im Rahmen dieser Arbeit nicht angepasst werden konnte, waren keinen Änderungen an der bestehenden Domainanalyse notwendig.

6 Android Grundkonzepte

Dieses Kapitel gibt einen kurzen Einblick in Android, sowie in die Android-Programmierung. Es werden die wichtigsten Komponenten kurz erläutert und der Lebenszyklus einer Android App aufgezeigt.

6.1 Android-Facts

Android ist eine Software-Plattform für mobile Geräte. Seit Oktober 2008 ist Android offiziell verfügbar. Entwickelt wird die Software-Plattform von Google und anderen Mitgliedern der Open Handset Alliance.

Abbildung 3: Android Market

Im Android Market haben Entwickler die Möglichkeit, ihre Software allen Android Benutzern anzubieten. Der Android Market ist der am schnellsten wachsende Software-Markt. Heute besteht der Market aus 580‘000 Apps, im Juli 2010 waren es noch 40‘000.

Laut Google werden pro Tag 400‘000 Android Smartphones ausgeliefert.

Im dritten Quartal 2011 erreicht Android einen Marktanteil von 52,5%.

Marktanteile der Smartphone-Betriebssysteme laut Gartner Inc. für das 3. Quartal 2011 (Verkäufe)[68]

Betriebssystem	Prozent
Android	52,5%
Symbian	16,9%
Apple iOS	15,0%
Blackberry OS	11,0%
Bada	2,2%
Windows Mobile	1,5%
Andere	0,9%

Abbildung 4: Android Marktanteile

Business Applikationen für die Android Plattform werden in Java geschrieben. Bei zeitkritischen Abläufen kann auf C oder C++ Bibliotheken zugegriffen werden.
Die Informationen und Bilder für dieses Kapitel stammen aus [WikiAndroidDE].

6.2 Versionen

Am 19. Oktober 2011 wurde die neuste Android Version „Ice Cream Sandwich“ veröffentlicht. „Ice Cream Sandwich“ ist somit die 8. Android Version seit dem Februar 2009.

In Tabelle 7: Android Versionen wird die Verwendung aller Versionen seit Cupcake dargestellt (November 2011).

Distribution	API level	%
4.0 *Ice Cream Sandwich*	14-15	0%
3.x.x *Honeycomb*	11-13	1.9%
2.3.x *Gingerbread*	9-10	44.4%
2.2 *Froyo*	8	40.7%
2.1 *Eclair*	7	10.7%
1.6 *Donut*	4	1.4%
1.5 *Cupcake*	3	0.9%

Tabelle 7: Android Versionen

In Abbildung 5: Android Versionen ist die Verwendung der einzelnen Versionen grafisch dargestellt.

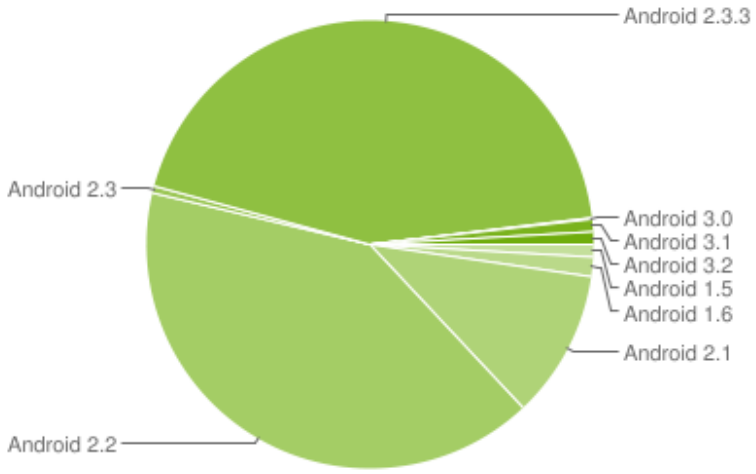

Abbildung 5: Android Versionen

Diese Grafik zeigt, dass Apps mit Mindest-Version 2.2, von den meisten Kunden genutzt werden können.

Welche Auswirkungen die unterschiedlichen Versionen auf die Tests hatten, sind im Anhang im Dokument Systemtest-Protokoll genauer beschrieben.
Die Informationen, Bilder und Tabellen für dieses Kapitel stammen aus [WikiAndroidEN].

6.3 Auflösungen

Da das Android Betriebssystem nicht auf eine spezielle Hardware zugeschnitten ist, haben viele Smartphone Hersteller die Chance genutzt und liefern ihre Geräte mittlerweile mit Android aus. Zum Leidwesen der Android-Entwickler muss eine App auf unterschiedliche Geräte angepasst werden.

Um den Entwicklern die Gestaltung des GUI's zu vereinfachen, wurden vier verschiedene Bildschirmgrössen, sowie vier verschiedene Auflösungen definiert.

- Bildschirmgrössen: small, normal, large, xlarge
- Auflösungen: low, medium, high, extra high

Im Dezember 2011 wurden Informationen über die verschiedenen Bildschirmgrössen sowie Auflösungen gesammelt. Eine grafische Auswertung der verschiedenen Kombinationen ist in Abbildung 6: Android Auflösungen/Grössendargestellt.

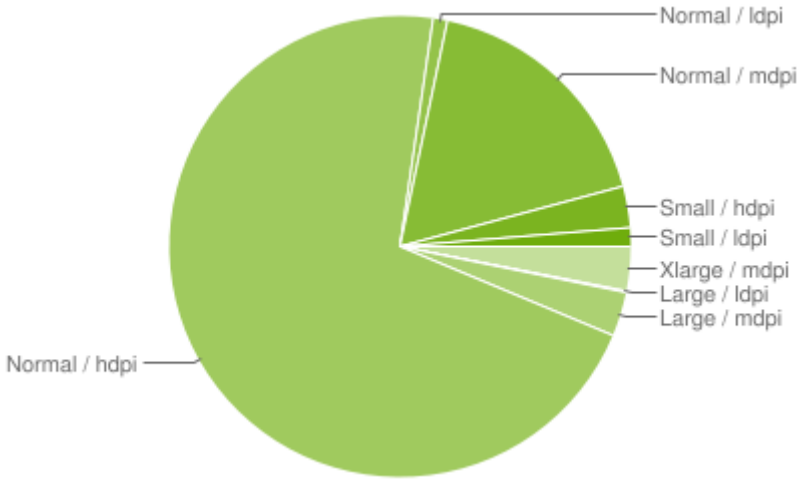

Abbildung 6: Android Auflösungen/Grössen

Die Problemlösung, um die App auf verschiedene Auflösungen vorzubereiten, ist im Kapitel 7.7 beschrieben. Welche Auswirkungen die unterschiedlichen Auflösungen auf die Tests hatten sind im Anhang im Dokument Systemtest-Protokoll genauer beschrieben.

In der Liste unterhalb sind jeweils bekannte Geräte aus den jeweiligen Kategorien aufgelistet:

- **ldpi** Samsung Galaxy Mini, HTC Wildfire
- **mdpi** HTC Magic, HTC Hero
- **hdpi** HTC Desire, Samsung Galaxy S2, Nexus S
- **xhdpi** Samsung Galaxy Nexus

Die Informationen und Bilder für dieses Kapitel stammen von [AndroidDev].

6.4 Architektur

Die Informationen und Bilder für die folgenden Kapitel stammen von [AndroidDev].

Die folgende Übersicht veranschaulicht die Hauptkomponenten des Android Betriebssystems.

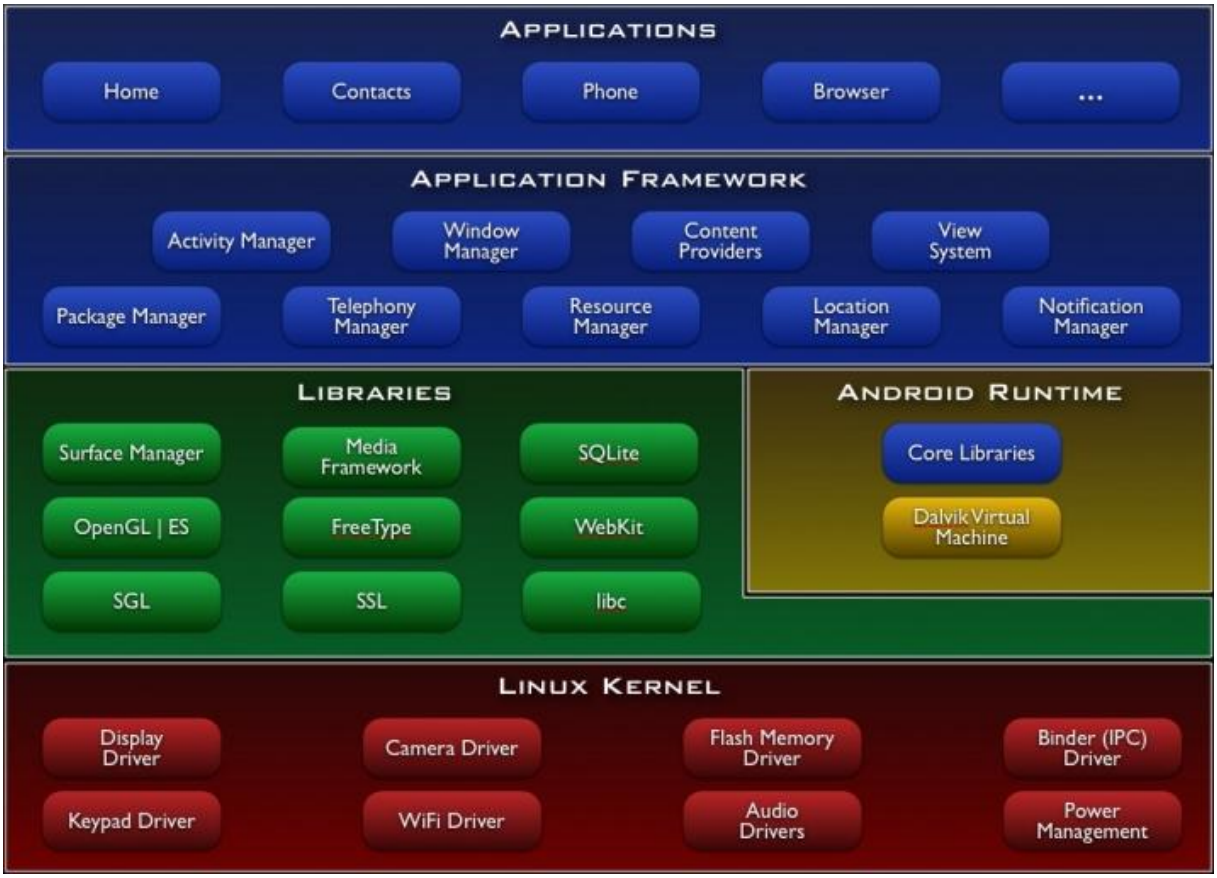

Abbildung 7: Android Architektur-Übersicht

- ***Applications*** sind die vorinstallierten Standardprogramme von Android (E-Mail, Kalender, SMS, Maps usw.)
- Das ***Application Framework*** bietet den Android Entwicklern zahlreiche APIs für verschiedenste Applikationen.
- Android umfasst diverse C/C++ ***Libraries*** (System C Library, OpenGL, SQLite usw.). Diese stehen dem Entwickler über das ***Application Framework*** zur Verfügung.
- Die ***Android Runtime*** umfasst einen Grossteil der Java Core Library, sowie die Dalvik Virtual Machine, welche zu einem späteren Zeitpunkt erläutert wird.
- Der ***Kernel*** dient als Abstraktionsschicht zwischen Hard- und Software. Er beinhaltet die Treiber für die Hardware-Komponenten und ist verantwortlich für Memory und Prozess-Management, Netzwerk und so weiter.

6.5 Grundlagen

Android Applikationen werden in Java geschrieben. Das Android SDK kompiliert den Source-Code, zusammen mit allen benötigten Daten (Bilder usw.) in ein Android-Package (*.apk). Dieses Android-Package wird von den Geräten verwendet um die Applikation zu installieren.

6.5.1 Dalvik Virtual Machine

Sobald die Applikation auf dem Gerät installiert ist, lebt sie in einer eigenen virtuellen Umgebung DVM (Dalvik Virtual Machine), diese läuft in einem eigenen Prozess. Die DVM stellt die Schnittstelle zu den Java Core Libraries dar und verfügt über einen Garbage Collector.
Das Android-System ist ein Multi-User Linux System, auf dem jede Applikation einem eigenen Benutzer zugewiesen wird. Die Benutzer-IDs sind allerdings nur im System bekannt. Das System setzt die Zugriffsberechtigungen so, dass nur die Applikation selbst Zugriff auf ihre Dateien hat.
Android implementiert somit das „**Least-Privilege Principle**“, was sich in einer sicheren Umgebung auszahlt, da standardmässig nicht auf Dateien zugegriffen werden kann, für die man keine Berechtigung hat.

6.5.2 Prozesse

Jede Android Applikation läuft normalerweise in einem eigenen Prozess. Das System startet den Prozess, wenn eine Komponente der Applikation gebraucht wird und beendet diesen wieder, wenn er nicht mehr gebraucht wird oder das System die Ressourcen für andere Applikationen benötigt.
Es gibt fünf verschiedene Prozesstypen, diese sind nach absteigender Priorität aufgelistet (Der erste Prozesstyp ist der wichtigste und wird zuletzt terminiert):

1. Foreground process
2. Visible process
3. Service process
4. Background process
5. Empty process

6.5.3 Threads

Wenn die Applikation ausgeführt wird, startet das System einen sogenannten „main“ Thread oder auch UI Thread genannt. Dieses Konzept unterscheidet sich von der Java Virtual Machine, wo es einen Main und UI Thread gibt. Der UI Thread ist aber, analog zu Java, der einzige Thread, der das GUI verändern darf. Bei zeitintensiven Aufgaben macht es Sinn, diese Aufgaben in Worker-Threads auszulagern. Damit der UI Thread nicht blockiert.
Android bietet dafür eine abstrakte Klasse *AsyncTask<Param, Progress, Result>* an. Dieses Konzept wird in der CENTRADO Android App verwendet um unterschiedlichste Tasks durchzuführen. Eine genaue Beschreibung befindet sich im Architektur und Design Abschnitt.

6.6 Komponenten

Die Applikationskomponenten sind die Bausteine einer Android App. Jede Komponente ist ein separater Einstiegspunkt, durch welchen das System die Möglichkeit hat, sich in die App einzuklinken. Natürlich sind nicht alle Komponenten auch Einstiegspunkte für die Benutzer der App, aber jede Komponente hat eine spezifische Rolle und ist ein Baustein, der das Gesamtverhalten der Applikation beeinflusst.
Im Ganzen gibt es vier verschiedene Komponenten-Typen.

- Activity
- Service
- Broadcast-Receiver
- Content-Provider

Jeder dieser vier Typen hat einen unterschiedlichen Nutzen, einen eigenen Lebenszyklus und definiert zusätzlich wie die Komponente erstellt und gelöscht werden kann.

Ein spezieller Aspekt von Android ist, dass grundsätzlich jede Applikation die Komponenten anderer Applikationen starten kann. Durch diesen Mechanismus kann man bestehende Komponenten (z.B. Kamera) einfach in die eigene Applikation einbinden. Dem Benutzer wird somit das Gefühl vermittelt, dass die Kamera Bestandteil der App ist.

6.6.1 Activity

Eine Activity repräsentiert genau eine Benutzeroberfläche. Zum Beispiel beinhaltet eine Mail-Applikation: eine Listenansicht der Mails, eine Detailansicht und eine Ansicht zum Verfassen neuer Nachrichten. Jede dieser Ansichten wäre eine eigene Activity und möglicherweise auch ein eigener Einstiegspunkt in die Applikation. Es wäre zum Beispiel denkbar, dass eine Galerie-App die „Ansicht zum Verfassen neuer Nachrichten“ aufrufen kann, um ein Bild direkt aus der Galerie zu versenden.

Eine solche Ansicht (Activity) wird immer als Unterklasse von *android.app.Activity* implementiert.

6.6.2 Service

Ein Service ist eine Komponente, die im Hintergrund läuft und zeitintensive Arbeiten oder Remote-Prozeduren aufführt. Ein Service besitzt keine Benutzeroberfläche. Beispielsweise kann ein Musik-Player als Service implementiert sein, dies ermöglicht es, während dem Surfen im Internet auch Musik zu hören.

Ein Service wird von einer anderen Komponente (z.B. Activity) gestartet und leitet von der *android.app.Service* Klasse ab.

6.6.3 Content Provider

Wie im Kapitel 6.5 Grundlagen erwähnt, sind die Anwendungsdaten vor fremden Zugriff geschützt. Mit Content Providern kann anderen Applikationen die Möglichkeit geboten werden, auf die Daten der eigenen App zuzugreifen.
Ein Content Provider leitet von der *android.content.ContentProvider* Klasse ab. Für die Centrado Android App wurde kein Content Provider gebraucht.

6.6.4 Broadcast Receiver

Ein Broadcast Receiver ist eine Komponente, die auf systemweite Broadcast-Meldungen reagiert. Eine solche Meldung könnte zum Beispiel sein, dass der Screen ausgeschaltet wurde. Broadcast Receiver besitzen keine Benutzeroberfläche, sie können jedoch Benachrichtigungen erstellen (Status-Bar). In der Regel ist ein Broadcast Receiver eine Art Gateway zu anderen Komponenten. In der CENTRADO App instanziiert ein Broadcast Receiver (*C2DMBroadcastReceiver*) einen Service, sobald eine Push-Nachricht empfangen wurde.
Ein Broadcast Receiver leitet von der *android.content.BroadcastReceiver* ab. Eine Broadcast-Meldung ist ein *android.content.Intent*.

6.6.5 Komponenten aktivieren (Intents)

Drei der vier Komponenten-Typen(Activities, Services und Broadcast Receivers) werden durch sogenannte Intents aktiviert. Ein Intent ist eine asynchrone Nachricht, welche die unterschiedlichen Komponenten zur Laufzeit aktivieren kann.
Ein Intent leitet von der *android.content.Intent* Klasse ab und beschreibt eine Nachricht, welche entweder eine spezifische Komponente (expliziter Intent) oder einen spezifischen Typ einer Komponente (impliziter Intent) aktiviert. Die impliziten Intents werden anhand von Intent-Filtern beschrieben. Wird ein Intent mit einem solchen Intent-Filter gestartet, sucht das System automatische alle passenden Komponenten und bietet dem Benutzer eine Auswahl, falls mehrere Komponenten diesen Intent-Filter erfüllen.

6.7 Android Manifest

Bevor das Android System eine Komponente starten kann, muss das System wissen, dass die Komponente überhaupt existiert und wo sie zu finden ist. Innerhalb des Android Manifest (AndroidManifest.xml) müssen alle Komponenten deklariert werden.
Im Android Manifest werden ausser den Komponenten noch andere Dinge konfiguriert:

- Berechtigungen, welche die App benötigt (z.B. Internetzugang)
- Benötigtes API Level (min. SDK Version)
- Deklaration von Hard- und Software Features (Multitouch, Kamera usw.)
- Third-Party Libraries

In Code-Snippet 1: AndroidManifest.xml ist ein Beispiel eines Android Manifests zu sehen.

6.7.1 Beispiel

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="MyListApp.ch"
      android:versionCode="1"
      android:versionName="1.0">
    <uses-sdk android:minSdkVersion="8" />

    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".MyListAppActivity"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".MyDetailActivity"></activity>
    </application>
</manifest>
```

Code-Snippet 1: AndroidManifest.xml

6.8 Ressourcen

Alle Ressourcen (Bilder, Audio-Files, Animationen, Binary-Files und XML-Dateien) werden im Ordner /res oder einem Unterordner davon abgelegt.
Auch Animationen, Menüs, Styles, Farben und Layouts werden in XML Dateien definiert und in den entsprechenden Unterordnern von /res abgelegt.
Für jede Ressource wird automatische eine eindeutige ID generiert (in der R Klasse), mit der die Ressource aus dem Programmcode referenziert werden kann (z.B: R.drawable.icon).
Werden die Ressourcen in den dafür vorgesehenen Verzeichnissen abgelegt, kann das System automatisch die richtigen Sprachfiles (Internationalisierung) oder die für das Display optimierten Bilder (DPI) laden.

6.8.1 Ordnerstruktur

```
MyProject/
    src/
        MyActivity.java
    res/
        drawable-xhdpi/
            icon.png
        drawable-hdpi/
            icon.png
        drawable-mdpi/
            icon.png
        drawable-ldpi/
            icon.png
        layout/
            main.xml
        values/
            strings.xml
        values-en/
            strings.xml
    AndroidManifest.xml
```

Abbildung 8: Projektstruktur

6.9 Layouts

Android-Anwendungen beschreiben ihre Benutzeroberflächen mittels deklarativen, XML-basierten Layout-Dateien. Das System instanziiert zur Laufzeit die Objektstrukturen anhand der XML-Dateien. Der Aufbau der GUI wird somit sauber von der Programm-Logik (Controller) getrennt.
Die Activity-Klassen können über einen einzigen Befehl die Layout-Strukturen in ihren Inhaltsbereich laden. [AndroidGalileo]

Das Code-Snippet 2: Layout main.xml zeigt die Deklaration eines Layouts in XML Form. Code-Snippet 3: Laden eines Layouts zeigt den Java-Code, wie das zuvor erstellte Layout eingebunden werden kann.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
       android:orientation="vertical" android:layout_width="fill_parent"
       android:layout_height="fill_parent" android:background="@drawable/background">
       <TextView android:layout_width="fill_parent"
              android:layout_height="wrap_content" android:text="@string/app_name"
              android:textColor="#008080" android:textSize="18dp" android:textStyle="bold"
/>
       <ListView android:id="@android:id/list" android:layout_height="fill_parent"
              android:layout_width="fill_parent"></ListView>
</LinearLayout>
```

Code-Snippet 2: Layout main.xml

```
setContentView(R.layout.main);
```

Code-Snippet 3: Laden eines Layouts

7 Implementation

In diesem Kapitel wird die Umsetzung von einzelnen Android Technologien vorgestellt. Ebenfalls werden Abläufe und konkrete Implementationen mit Code-Beispielen aus der CENTRADO Android App erklärt.

7.1 Android Cloud to Device Messaging (C2DM)

Das Centrado Impuls Prinzip basiert auf Push Events, welche der Abonnent auf seinem Android Gerät empfängt. Android Entwickler haben erst seit Android Version 2.2 die Möglichkeit, einen Push-Mechanismus zu benutzen. Dieser Push-Mechanismus nennt sich Android Cloud to Device Messaging, kurz C2DM. C2DM ist bis jetzt die einzige Möglichkeit, Push Events auf ein Android Gerät zu senden. Ausführlichere Informationen sind im Anhang C2DM-Spezifikation dokumentiert.
Dieses Kapitel zeigt, wie C2DM im CENTRADO Android App integriert wurde.

7.1.1 Registrierung des Servers

Damit der Push-Mechanismus getestet werden konnte, wurde ein entsprechender Testserver mit C2DM implementiert. Da von Crealogix kein Testserver zur Verfügung gestellt wurde, wurde ein Server von den HSR Informatik-Diensten beantragt.
Dieser Abschnitt soll die Implementation des Servers kurz erläutern.

Damit der Server überhaupt Push-Notifications an seine Clients schicken kann, muss er sich zuerst beim Google-Service ein Authentication-Token besorgen. Dies geschieht über einen speziell dafür eingerichteten Google-Account. Ein solcher Account kann unter http://code.google.com/intl/de-DE/android/c2dm/signup.html registriert werden. Für die Erläuterung dieser Problematik wird ein fiktiver Google-Account mit dem folgenden Benutzerangaben verwendet:

E-Mail:	MyC2DMAccount@gmail.com
Password:	qwer1234

Tabelle 8: Beispiel C2DM Account

Die Freischaltung dieses Accounts kann bis zu zwei Tagen dauern. Sobald das Bestätigungs-Mail empfangen wurde, können unter diesem Benutzer-Account C2DM-Nachrichten verschickt werden. Der Server muss dazu ein sogenanntes Authentication-Token für diesen Benutzer-Account erlangen. Die Klasse *AuthenticationUtil* im Server Projekt bietet diese Funktionalität an. Dieses Token muss einmalig vom Google-Service (https://www.google.com/accounts/ClientLogin) geholt werden.
Das Code-Snippet 4: AuthenticationUtil C2DM-Server zeigt wie das Authentication-Token beim Google-Server geholt wird.

```
public class AuthenticationUtil {
	public static String getToken(String email, String password) throws IOException{

		byte[] data = formDataString(email,password).getBytes();
		URL url = new URL("https://www.google.com/accounts/ClientLogin");
		HttpURLConnection con = (HttpURLConnection) url.openConnection();
		con.setUseCaches(false);
		con.setDoOutput(true);
		con.setRequestMethod("POST");
		con.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
		con.setRequestProperty("Content-Length", Integer.toString(data.length));

		OutputStream output = con.getOutputStream();
		output.write(data);
		output.close();

		BufferedReader reader = new BufferedReader(new
		    InputStreamReader(con.getInputStream()));
		String[] split = reader.readLine().split("=");
		String authToken = split[1];
		return authToken;
	}

	private static String formDataString(String email, String password){
		StringBuilder builder = new StringBuilder();
		builder.append("Email="+email);
		builder.append("&Passwd="+password);
		builder.append("&accountype=GOOGLE_OR_HOSTED");
		builder.append("&source=Centrado-C2DM-Example");
		builder.append("&service=ac2dm");
		return builder.toString();
	}
}
```

Code-Snippet 4: AuthenticationUtil C2DM-Server

Falls dieses Token einmal von Google geändert wird, wird es beim Versenden der Push-Nachrichten gemeldet (Header-Feld „Update-Client-Auth“ in der HTTP-Response). Dieser Update-Mechanismus wurde auf dem Testserver jedoch nicht implementiert.

7.1.2 Registrierung des Clients

Damit der Client Push-Notifications vom Google-Service erhalten kann, muss er zuerst einen Push-Service abonnieren. Grundvoraussetzung dafür ist ein registrierter Google-Account in den Account-Einstellungen des Android-Phones.
Sobald sich der Benutzer erfolgreich an der App angemeldet hat, wird das C2DM-Enabling durchgeführt. Wie auf der Skizze Abbildung 9: Skizze C2DM Registrierung erläutert, wird zuerst eine Device-Registration-ID (registrationID) angefordert. Die Anfrage beinhaltet die beiden Parameter Sender-ID und Application-ID. Die Sender-ID ist dabei der vom C2DM-Server verwendete Benutzer-Account (z.B. MyC2DMAccount@gmail.com).

7.1.2.1 Empfangen der Device-Registration-ID

Die CENTRADO Android App installiert einen eigenen Broadcast-Receiver, welcher Push-Notifications, sowie die Device-Registration-ID empfängt. Diese Informationen werden mittels Intents geschickt. Sobald ein solcher Intent vom Google-Service empfangen wird, instanziiert der Broadcast-Receiver einen entsprechenden Service, welcher dann die Notification in der Status-Bar anzeigt, oder die Device-Registration-ID an den C2DM-Server übermittelt und so die C2DM-Registrierung abschliesst. (Abbildung 9: Skizze C2DM Registrierung)

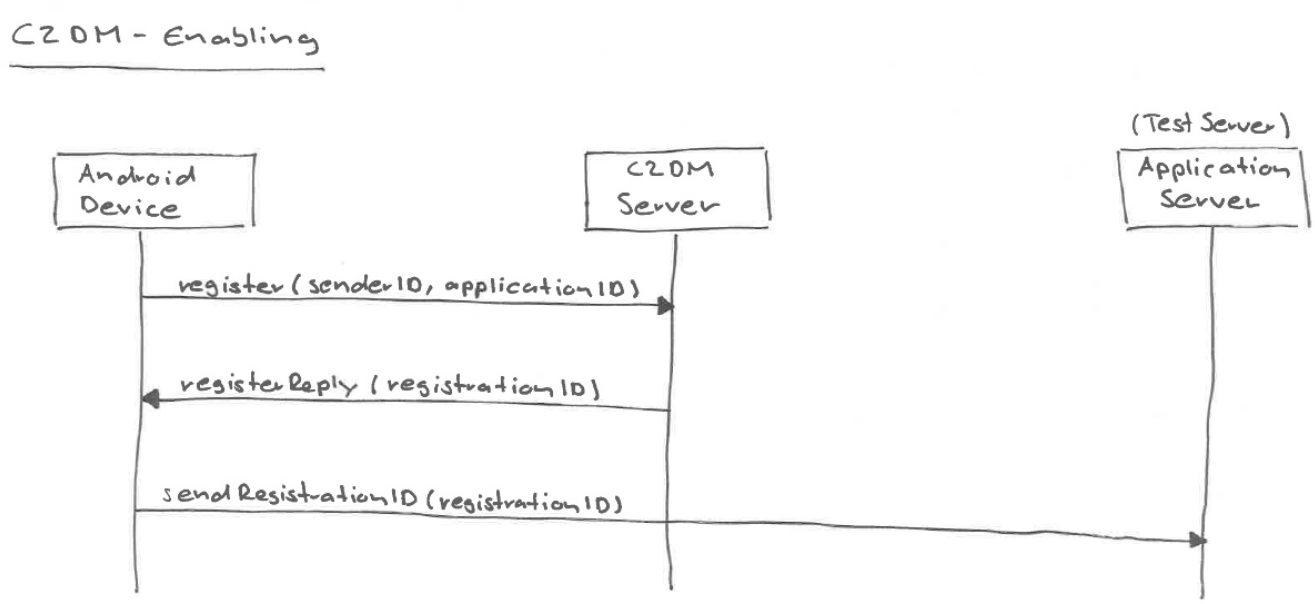

Abbildung 9: Skizze C2DM Registrierung

Das untenstehende Sequenzdiagramm Abbildung 10: C2DM Registrierung zeigt den detaillierten Ablauf der C2DM-Registrierung auf dem Client.

Ablauf:

1. Sobald die Listen-Ansicht (*ImpulseListActivity*) gestartet wird, registriert sich die App für das C2DM.
2. Die *C2DMessaging* Klasse schickt die Anforderung zu Google
3. Google löst einen sog. *REGISTRATION_CALLBACK_INTENT* aus, welcher vom *C2DMBroadcastReceiver* empfangen wird.
4. Der *C2DMBroadcastReceiver* übergibt den Intent an den *C2DMBaseReceiver* (Service), welcher zusammen mit dem *C2DMReceiver* für die Verarbeitung des Intents (Anzeigen einer Benachrichtigung o.ä.) verantwortlich ist.

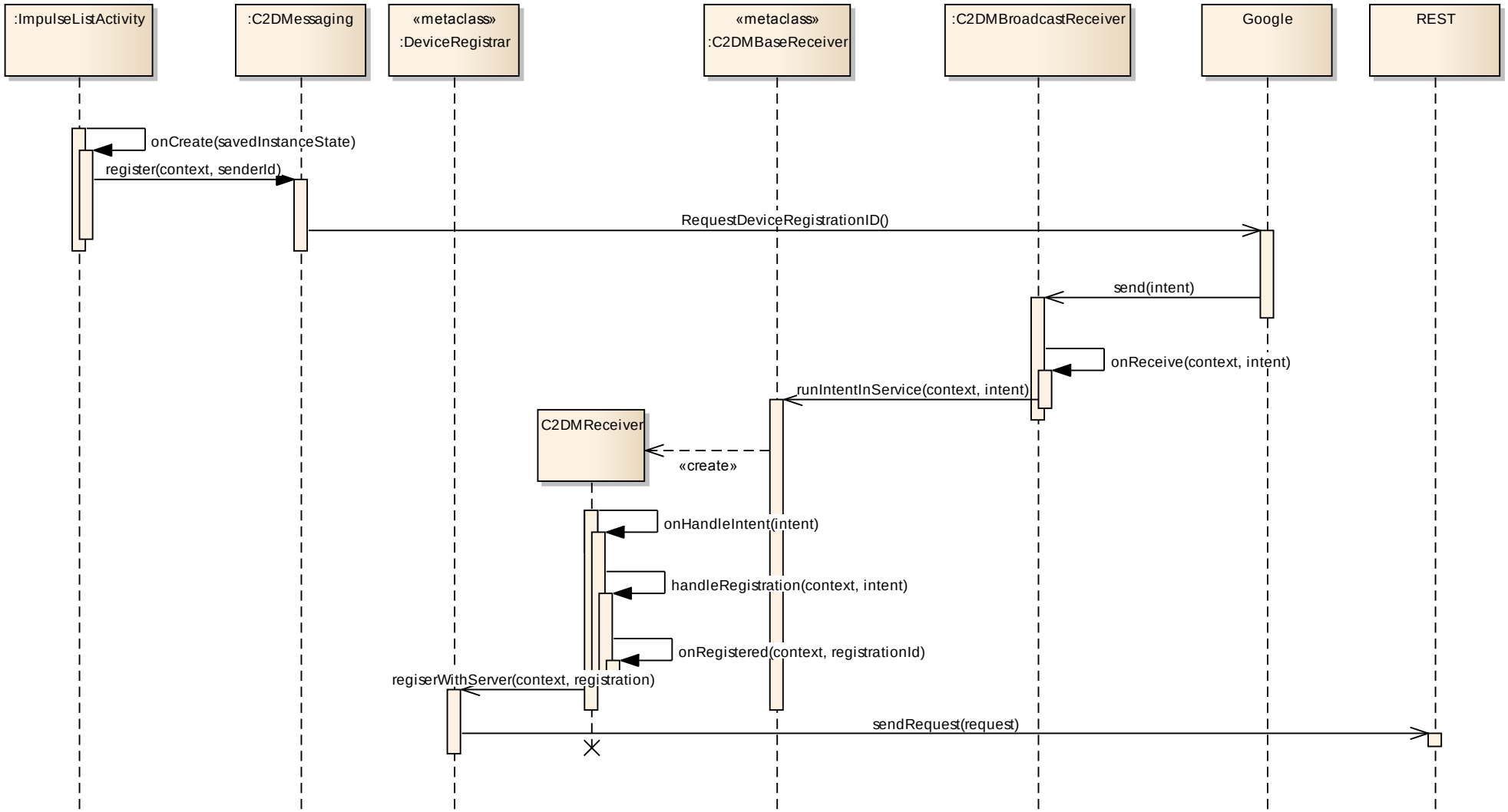

Abbildung 10: C2DM Registrierung

7.1.3 Senden einer Push-Notification

Sobald sich der Client beim Server registriert hat, kann dieser die Push-Notifications versenden. Dieses Szenario wird in der nächsten Skizze Abbildung 11: Senden einer Push-Notification veranschaulicht.

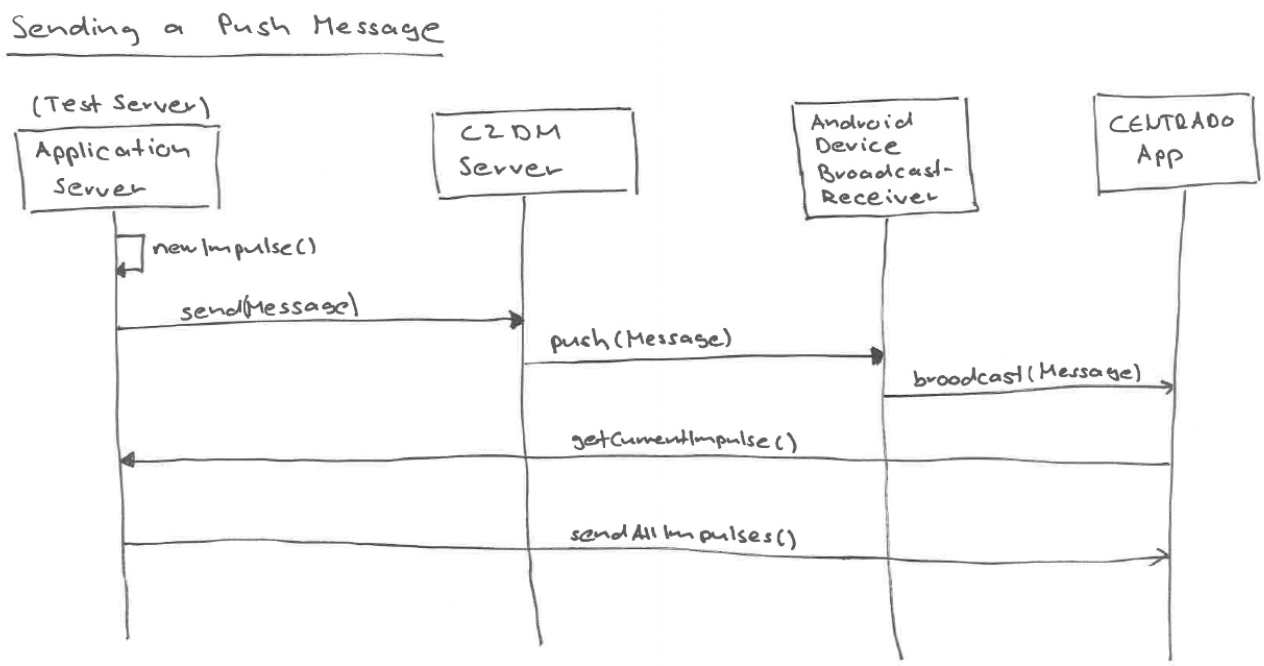

Abbildung 11: Senden einer Push-Notification

Dazu generiert der C2DM-Server einen HTTP-Post-Request an die Google-Schnittstelle, dieser Request beinhaltet das Authentication-Token, sowie die Device-Registration-ID. Die *C2DMSender* Klasse des Servers realisiert diesen Teil.
In vielen Anleitungen wird bei diesem POST-Request das Property-Feld „Content-Type“ gesetzt. Bei der Implementation des Prototyps sind wegen diesem Property-Feld Probleme aufgetreten. Aus diesem Grund wurde dieses Feld nicht mehr weiter verwendet.

```
public class C2DMSender {
	public static final String PARAM_REGISTRATION_ID = "registration_id";
	public static final String AUTH_TOKEN = "DQAAAL8AAAAx1MEeT…";
	public static final String DATAMESSAGING_SEND_ENDPOINT =
		"http://android.apis.google.com/c2dm/send";

	public static void sendTestNotification(String deviceRegistrationID, String message)
			throws Exception {
		URL url = new URL(DATAMESSAGING_SEND_ENDPOINT);
		HttpURLConnection request = (HttpURLConnection) url.openConnection();
		request.setDoOutput(true);
		request.setUseCaches(true);

		StringBuilder buf = new StringBuilder();
		buf.append(PARAM_REGISTRATION_ID).append("=")
				.append((URLEncoder.encode(deviceRegistrationID, "UTF-8")));
		buf.append("&collapse_key").append("=")
				.append((URLEncoder.encode("0", "UTF-8")));
		if (message != null) {
			buf.append("&data.MESSAGE=").append(message);
		}
		request.setRequestMethod("POST");
		request.setRequestProperty("Authorization", "GoogleLogin auth="
				+ AUTH_TOKEN);
		request.setRequestProperty("Content-Length",
				buf.toString().getBytes().length + "");

		OutputStreamWriter post = new OutputStreamWriter(
				request.getOutputStream());
		post.write(buf.toString());
		post.flush();
		System.out.println("Response-Code: " + request.getResponseCode());
	}
}
```

Code-Snippet 5: C2DMSender C2DM-Server

Attribut:	Beschreibung:
AUTH_TOKEN	Authentication-Token vom C2DM-Benutzer-Account wird ins Property „Authorization“ gschrieben.
deviceRegistrationID	Device-Registration-ID eines Clients wird dem Message-Body angehängt (Als Property „registration_id“).

Tabelle 9: C2DMSender Attribute

Wie im Kapitel 7.1.1 beschrieben, kann das Authentication-Token von Google geändert werden.
An dieser Stelle müsste die Response vom Google-Server überprüft werden, ob sich darin ein neues Authentication-Token befindet (Header-Feld „Update-Client-Auth“).

7.1.4 Empfangen der Push-Notification

Wie im Kapitel 7.1.2.1 Empfangen der Device-Registration-ID diskutiert, installiert die App einen Broadcast-Receiver, welcher auf die Google-Messages reagiert. Sobald eine Push-Notification eintrifft, instanziiert der *C2DMBroadcastReceiver* einen Service, welcher die Notification in der Status-Bar anzeigt.
Im Code-Snippet 6: C2DMBaseReceiver entscheidet der Service, ob es sich bei der Nachricht von Google um eine Push-Notification (C2DM_INTENT) oder um die Device-Registration-ID (REGISTRATION_CALLBACK_INTENT) handelt.

```
@Override
  public final void onHandleIntent(Intent intent) {
      try {
          Context context = getApplicationContext();
          if (intent.getAction().equals(REGISTRATION_CALLBACK_INTENT)) {
              handleRegistration(context, intent);
          } else if (intent.getAction().equals(C2DM_INTENT)) {
              onMessage(context, intent);
          } else if (intent.getAction().equals(C2DM_RETRY)) {
              C2DMessaging.register(context, senderId);
          }
      } finally {
          mWakeLock.release();
      }
  }
```

Code-Snippet 6: C2DMBaseReceiver

7.1.5 C2DM-Klassen

Das C2DM-Grundgerüst wird von Google zur Verfügung gestellt. Die Code-Beispiele stehen unter GNU General Public License und können von frei verwendet werden. Folgende Klassen entstammen aus den Google-Beispielen: *C2DMessaging*, *C2DMBroadcastReceiver* und *C2DMBaseReceiver*. Die beiden Klassen *DeviceRegistrar* und *C2DMReceiver* wurden, unseren Bedürfnissen entsprechend, implementiert und befinden sich darum nicht im Source-Ordner „C2DM“.

7.2 SQLite

Die CENTRADO Android App hat eine SQLite-Datenbank, um die Impulse dauerhaft speichern zu können. In der Datenbank werden alle Informationen zum jeweiligen Impuls abgelegt. Zusätzlich wird abgespeichert, ob der Impuls bereits gelesen wurde und ob der Impuls als Favorit markiert wurde. Jedes Mal wenn die *ImpulsListActivity* angezeigt wird, werden alle Impulse von Server geladen. Da die Serverschnittstelle während der Semesterarbeit nicht angepasst werden konnte, musste bei jedem Impuls wird überprüft, ob er bereits in der Datenbank gespeichert ist.

- Falls dies der Fall ist, wird der Inhalt des Impulses neu in die Datenbank gespeichert. Dies hat zum Vorteil, dass zum Beispiel Impulse, welche auf dem Server auf Rechtschreibung korrigiert wurden, auch beim Android Benutzer angepasst werden.
  Die Flags, welche Auskunft darüber geben ob der Impuls bereits gelesen wurde, oder ob er ein Favorit ist, werden dabei nicht überschrieben.
- Wenn der Impuls neu ist wird der Impuls in der Datenbank abgelegt. Die Flags Favorit/Gelesen werden dabei auf Default gestellt (Kein Favorit/Ungelesen).

7.2.1 Fetch All

Die Impulsliste hat immer einen aktuellen Cursor. Ein Cursor repräsentiert die Ergebnisse einer Datenbankabfrage. Wenn die Impulsliste neu geladen wird, wird über den *DBController* per fetchAll(), eine SQL-Query an die Datenbank abgesetzt. Die SQL-Query für den fetchAll() Aufruf ist im Hintergrund eine „SELECT * FROM ImpulseTable“ Abfrage. Von der Datenbank erhalten wir einen neuen Cursor, der somit alle Einträge aus der Datenbank beinhaltet.
Da in der Liste immer der neuste Impuls zuoberst sein sollte, wird bei allen Abfragen auf die Datenbank ein Sortierargument mitgegeben.

```
private static final String SQL_SORTING_ARGUMENT = ImpulseColumns.TIMESTAMP + " DESC";

public Cursor fetchAll() {
              return db.query(TABLE_NAME, null, null, null, null, null,
                            SQL_SORTING_ARGUMENT);
       }
```

Code-Snippet 7: SQLite fetchAll()

Die optionalen Anforderungen Suchen und Filtern arbeiten beide nach dem gleichen Schema wie der fetchAll() Aufruf. Der einzige Unterschied ist die query() Abfrage auf der Datenbank. Konkret bedeutet dies, dass die Suche eingeschränkt wird und der *Cursor* somit meistens weniger Einträge enthält.

7.2.2 Suche

Die Search Methode wird mit dem Suchwort, welches der User eingegeben hat, aufgerufen. Bei der Suche wird der Titel (TITLE), die kurze Beschreibung (DESCRIPTIONSHORT) sowie der Impulstext (DESCRIPTION) mithilfe der LIKE Anweisung durchsucht.

```
public Cursor search(String query) {
        String formattedQuery = "%" + query + "%";
        return db.query(
        TABLE_NAME,
        null,
        TITLE + " LIKE ? OR " + DESCRIPTION + " LIKE ? OR " + DESCRIPTIONSHORT + " LIKE ?",
        new String[] { formattedQuery, formattedQuery, formattedQuery },
        null,
        null,
        SQL_SORTING_ARGUMENT);
}
```

Code-Snippet 8: SQLite search()

In diesem Sequenzdiagramm wird der Ablauf einer Textsuche aufgezeigt.

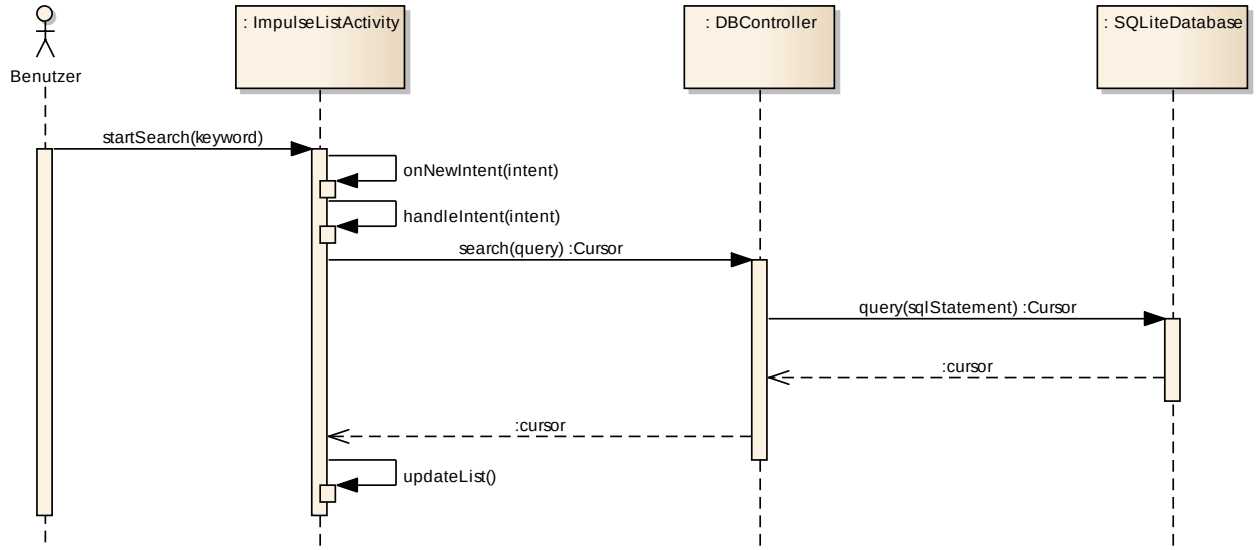

Abbildung 12: Suchablauf

Ablauf:

1. Nachdem der Benutzer seinen Suchtext eingegeben hat, startet er die Suche.
2. Durch das Starten der Suche bekommt die ImpulseListActivity einen Intent, welcher in der Methode onNewIntent(Intent intent) abgefangen wird.
3. In der handleIntent(Intent intent) Methode wird das Suchwort aus dem neuen Intent folgendermasse ausgepackt:
   ```
   String query = intent.getStringExtra(SearchManager.QUERY);
   ```
4. Der Suchstring wird anschliessend der search(String query) Methode übergeben.
5. Der *DBController* setzt nun eine SQL Abfrage ab, indem er nach dem String sucht.
6. Von der SQLiteDatabase wird ein *Cursor* zurückgegeben, der die entsprechenden Impulse beinhaltet.

7. Der *DBController* gibt den Cursor an die *ImulseListActivity* zurück.
8. In der updateList() Methode auf der *ImpulseListActivity* wird ein neuer *ImpulseCursorAdapter* instanziiert, welcher die Einträge des Cursors auf die Liste abbildet.

7.2.3 Filter

Der Filtermethode wird die zu durchsuchende Spalte (filterColumn), sowie das Filterkriterium (filterKeyword) mitgegeben. Wenn zum Beispiel nach allen Favoriten-Impulsen gefiltert werden soll, lautet die filterColumn: „favoriteFlag“ und das filterKeysword: „favorite“. Somit bekommt man einen Cursor, welcher nur die als Favoriten markierten Impulse beinhaltet.

```
public Cursor filter(String filterColumn, String filterKeyword) {
        return db.query(
        TABLE_NAME,
        null,
        filterColumn + " = ? ",
        new String[] { filterKeyword },
        null,
        null,
        SQL_SORTING_ARGUMENT);
}
```

Code-Snippet 9: SQLite filter()

7.3 Android Manfiest

7.3.1 Berechtigungen

Die App muss vom System bzw. Benutzer folgende Berechtigungen erlangen:

```
<!-- Only this application can receive the messages and registration result -->
<permission android:name="ch.centrado.app.permission.C2D_MESSAGE"
       android:protectionLevel="signature" />
<uses-permission android:name="ch.centrado.app.permission.C2D_MESSAGE" />
<!-- This app has permission to register and receive message -->
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />
<!-- Permissions for internet access and account access -->
<uses-permission android:name="android.permission.INTERNET" />
<!-- App must have this permission to use the library -->
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
<uses-permission android:name="android.permission.USE_CREDENTIALS" />
```

Code-Snippet 10: Manifest Permissions

Berechtigung:	Beschreibung:
ch.centrado.app.permission.C2D_MESSAGE	Signatur der Push-Nachrichten, damit nur die CENTRADO App diese verwenden kann.
com.google.android.c2dm.permission.RECEIVE	System-Berechtigung für das Empfangen von Push-Notifications
android.permission.INTERNET	System-Berechtigung für den Gebrauch der Internetverbindung (Impulse-Download)
android.permission.WAKE_LOCK	System-Berechtigung für das Sperren des Standby-Modus (Damit der C2DM-Service nicht terminiert wird)
android.permission.GET_ACCOUNTS	System-Berechtigung für den Zugriff auf den Account-Manager (C2DM-Registrierung)
android.permission.USE_CREDENTIALS	System-Berechtigung für das Benutzen des Google-Accounts (C2DM-Registrierung)

Tabelle 10: Manifest Permissions

7.3.2 Activities

Wie im Kapitel 7 Android Grundkonzepte beschrieben, sind im Android Manifest alle Komponenten der Applikation deklariert. Folgende Intent-Filter sind dabei speziell zu beachten:

Intent-Filter:	Beschreibung:
android.intent.action.MAIN	Definiert Haupteinstiegspunkt der Applikation
android.intent.category.LAUNCHER	Definiert welche Activity erscheint, wenn die Applikation aus dem Launcher gestartet wird.
android.intent.action.SEARCH	Wird für das Durchsuchen der Listen-Ansicht verwendet.

Tabelle 11: Intent Filter

7.3.2.1 Horizontale Ansicht

Die App unterstützt zurzeit nur die vertikale Ansicht (Portrait). Damit die App auch in horizontaler Ansicht verwendet werden kann, muss auf den gewünschten Activities das Flag (`android:screenOrientation="portrait"`) gelöscht werden.

```
<activity android:name=".gui.ImpulseListActivity"
	android:label="@string/app_name" android:theme="@style/ImpulseTheme"
	android:screenOrientation="portrait" android:launchMode="singleTask">
	<intent-filter>
		<action android:name="android.intent.action.SEARCH" />
	</intent-filter>
	<meta-data android:name="android.app.searchable"
		android:resource="@xml/searchable" />
</activity>
```

Code-Snippet 11: Manifest horizontale Ansicht

7.4 Login

Für den Login wird preemptive HTTP-Basic-Authentication verwendet. SSL wird nicht unterstützt. Die Abfrage auf den Server ist in einen Hintergrund-Thread ausgelagert (*LoginTask*). Falls eine *LoginException* (fehlerhafte Benutzerdaten) oder eine *IOException* (Verbindungsproblem) auftritt, wird eine Fehlermeldung angezeigt und der Benutzer zur erneuten Eingabe aufgefordert.

```
@Override
protected AsyncTaskResult<Object> doInBackground(String... params) {
	username = params[0];
	password = params[1];
	try {
		ConnectionUtil.login(username, password);
	} catch (LoginException e) {
		e.printStackTrace();
		return new AsyncTaskResult<Object>(e);
	} catch (IOException e) {
		e.printStackTrace();
		return new AsyncTaskResult<Object>(e);
	} catch (Exception e) {
		e.printStackTrace();
		return new AsyncTaskResult<Object>(e);
	}
	return new AsyncTaskResult<Object>("success");
}
```

Code-Snippet 12: LoginTask

```
public static void login(String username, String password)
	throws IOException, LoginException {
	setCredentials(username, password);
	HttpResponse response = makeRequest(LOGIN);
	if (getStatusCode(response) != SUCCESS_CODE) {
		throw new LoginException();
	}
}private static void setCredentials(String username, String password) {
	UPC = new UsernamePasswordCredentials(username, password);
	((AbstractHttpClient) HTTP_CLIENT).getCredentialsProvider()
			.setCredentials(AUTH_SCOPE, UPC);
}
```

Abbildung 13: ConnectionUtil

7.5 Shared Preferences

Android stellt mehrere Möglichkeiten zur Verfügung, um Daten persistent abzuspeichern. Shared Preferences ist eine Möglichkeit, um primitive Datentypen in Key-Value-Paaren abzuspeichern. Shared Preferences bieten sich daher sehr gut an, um Einstellungen des Users oder Benutzerinformation abzuspeichern. Aus diesem Grund wurden für die Login-Informationen des Benutzers Shared Preferences verwendet. [AndroidGrundProg]

Nachdem sich der Benutzer einmal erfolgreich eingeloggt hat, muss er sich nicht mehr erneut Anmelden, ausser wenn er sich explizit abmeldet. Um diesen Autologin anzubieten muss der Username und das Passwort des Benutzers abgespeichert werden.

7.6 GUI

Beim CENTRADO Android App wurde viel Wert auf ein ansprechendes GUI gelegt. Dies umfasst nicht nur ein ausgewogenes Layout, sondern auch Corporate Identitiy.

Folgende GUI Elemente wurden bei den GUI-Prototypen im Corporate Identity Stil designet:

- Hintergrund
- Titelleiste
- Farben
- Schriften

Android bietet gute Möglichkeiten, um die Gestaltung des GUI möglichst einheitlich zu machen. Nebst der zentralen Definition von Farben, können auch Styles und Themes erstellt werden. Styles und Themes werden im Ordner /values in der XML-Datei styles.xml gespeichert.

- Style: Formatvorlagen für View-Elemente
- Themes: Formatvorlagen für Layouts

7.6.1 Styles

Styles können als Formatvorlagen für View-Elemente genutzt werden. Ein View ist in diesem Fall ein GUI-Element, wie zum Beispiel eine *TextView*. Globale Eigenschaften einer *TextView* können somit im in der style.xml Datei definiert werden. Der Vorteil davon ist, dass diese Attribute nicht mehr in den Layout Files zugewiesen werden müssen, was das die Layout Dateien schlanker und übersichtlicher macht. [AndroidGrundProg]

Abbildung 14: Login-Screen

Da die beiden TextViews Benutzername und Passwort die gleiche Farbe, sowie gleichen Schrifttyp und Schriftgrösse haben, wird ihre Definition mithilfe von Code-Snippets erläutert.

In der colors.xml Datei, welche sich im Ordner /values befindet, wurden zwei Farben definiert.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
	<color name="centrado_blue_light">#39BDCD</color>
	<color name="centrado_blue">#008080</color>
</resources>
```

Code-Snippet 13: /values/colors.xml

In der Datei styles.xml wurde ein Style mit dem Namen „txt_normal“ erstellt. Dieser definiert die Schriftgrösse, sowie die Schriftfarbe.

```
<style name="txt_normal">
	<item name="android:textSize">15sp</item>
	<item name="android:textColor">@color/centrado_blue</item>
</style>
```

Code-Snippet 14: /values/styles.xml

Im Layout (login.xml) werden die View-Elemente definiert. Dabei wird für Benutzername und Passwort jeweils eine *TextView* für die Beschriftung der Eingabefelder verwendet, sowie zwei *EditText* Elemente, in denen die Eingaben getätigt werden können. Beiden TextViews wird mithilfe des Attributs „style“, der zuvor definierte „txt_normal“-Style zugewiesen. Somit ist die Einheitliche Gestaltung von View-Elementen möglich.

```
    <TextView android:layout_width="fill_parent"
	android:layout_height="wrap_content"
	android:text="@string/txt_login_username"
	android:paddingTop="10dp"
	style="@style/txt_normal"/>

    <EditText android:id="@+id/edit_login_username"
	android:layout_height="wrap_content"
	android:layout_width="match_parent"
	android:singleLine="true">
	<requestFocus></requestFocus>
    </EditText>

    <TextView android:layout_width="fill_parent"
	android:layout_height="wrap_content"
	android:text="@string/txt_login_password"
	style="@style/txt_normal"/>

    <EditText android:id="@+id/edit_login_password"
	android:layout_height="wrap_content"
	android:layout_width="match_parent"
	android:inputType="textPassword"
	android:singleLine="true">
    </EditText>
```

Code-Snippet 15: Styles /layout/login.xml

7.6.2 Themes

Im Gegensatz zu den Styles, welche das Format einzelner Views vorgeben, kann mit Themes das Erscheinungsbild einer ganzen Activity definiert werden. [AndroidGrundProg]

Da im CENTRADO Android App überall, ausser auf dem Splash-Screen, die Titelleiste mit dem CENTRADO Logo angezeigt wird, wurde dafür ein Theme definiert.

Themes werden ebenfalls in der styles.xml Datei definiert. Dabei wurde ein *MainTheme* definiert, in welchem die Titelleiste definiert wurde. Für jede Activity wurde anschliessend ein eigenes Theme mit dem richtigen Hintergrund definiert, welches vom *MainTheme* erbt. Somit ist die Titelleiste in jeder Activity gleich.

```
<style name="MainTheme" parent="android:Theme">
        <item name="android:windowTitleSize">40dp</item>
        <item name="android:windowTitleBackgroundStyle">@style/WindowTitleBackground</item>
</style>
<style name="WindowTitleBackground">
        <item name="android:background">@color/centrado_blue_light</item>
</style>
<style name="LoginTheme" parent="MainTheme">
        <item name="android:windowBackground">@drawable/bg_flowers</item>
</style>
```

Code-Snippet 16: /values/styles.xml

Nachdem das Theme definiert wurde, kann es über das Manifest einer konkreten Activity zugewiesen werden.

```
<activity android:name=".gui.LoginActivity" android:label="@string/app_name"
        android:theme="@style/LoginTheme" android:screenOrientation="portrait">
</activity>
```

Code-Snippet 17: Theme in AndroidManifest.xml

Die Platzierung des Logos innerhalb der Titelleiste wird wiederum über ein eigenes Layout gemacht um den Standort des Logos zu definieren (/layout/titlebar.xml).

7.7 Internationalisierung

Die Internationalisierung kann durch verschiede Sprachdateien realisiert werden. Alle Sprachfragmente, welche in verschiedenen Sprachen dargestellt werden sollen, müssen in den strings.xml Dateien abgelegt werden. Die Standard-Sprachdatei wird im Ordner „values“ abgelegt. Wenn man weitere Sprachen unterstützten möchte, müssen dafür neue Values-Ordner angelegt werden „values-xx“.

Beim laden des Apps wird jeweils die Systemsprache des Geräts mit den vorhandenen Sprachdateien abgeglichen. Wenn eine entsprechende Sprache angeboten wird, wird diese geladen. Ansonsten wird die Standardsprache verwendet.

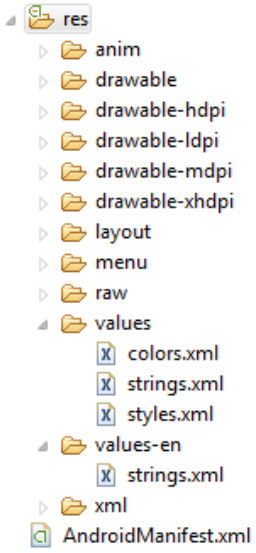

Abbildung 15: Internationalisierungs-Ordnerstruktur

In den Code-Snippets unterhalb sind jeweils zwei internationalisierte Strings dargestellt.

```
<string name="txt_login_username">Benutzername:</string>
<string name="txt_login_password">Passwort:</string>
```

Code-Snippet 18: /values/strings.xml

```
<string name="txt_login_username">Username:</string>
<string name="txt_login_password">Password:</string>
```

Code-Snippet 19: /values-en/strings.xml

Die internationalisierten Strings können anschliessend im Layout mittels dem Attribut „android:text“ referenziert werden. Das Code-Snippet unterhalb zeigt die TextViews Benutzername und Password aus der *LoginActivity*.

```
<TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="@string/txt_login_username"
        android:paddingTop="10dp"
                    style="@style/txt_normal"/>

<EditText android:id="@+id/edit_login_username"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:singleLine="true">
        <requestFocus></requestFocus>
</EditText>

<TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="@string/txt_login_password"
        style="@style/txt_normal"/>

<EditText android:id="@+id/edit_login_password"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:inputType="textPassword"
        android:singleLine="true">
</EditText>
```

Code-Snippet 20: Internationalisierung /layout/login.xml

Damit das Empfangsdatum der einzelnen Impulse in der Liste angezeigt wird, wurde das Datum folgendermassen gesetzt.

```
textDate.setText(DateFormat.getDateFormat(view.getContext()).format(new Date()));
```

Code-Snippet 21: Internationalisierung Datum

Somit wird das Datumsformat auch an die eingestellte Systemsprache angepasst.

7.8 Auflösungsoptimierung

Wenn die App auf allen verfügbaren Geräten ein einheitliches und ansprechendes GUI bieten sollte, müssen alle Grafiken in den vier definierten Auflösungen designet werden. Dies bedeutet, dass zum Beispiel für die CENTRADO Hintergrundgrafiken jeweils eine Version in ldpi, mdpi, hdpi und xhdpi angefertigt werden musste.

- **ldpi** 240x320 px
- **mdpi** 320x480 px
- **hdpi** 480x800 px
- **xhdpi** 720x1280 px

Sobald die Grafiken vorbereitet sind, bietet Android eine einfache Ordnerstruktur an, in welcher die Grafiken abgelegt werden können.

Ähnlich wie bei der Internationalisierung gibt es auch hier einen Standardordner, indem Grafiken abgelegt werden können (drawable). Wenn man die Grafiken jedoch in den unterschiedlichen Grössen hat, kann man diese in den passenden Ordnern ablegen. Beim Laden der App wird, wie bei der Internationalisierung, überprüft, ob es Grafiken gibt, welche zum Gerät passen. Falls dies nicht der Fall ist werden die Grafiken aus dem Standardordner geladen.

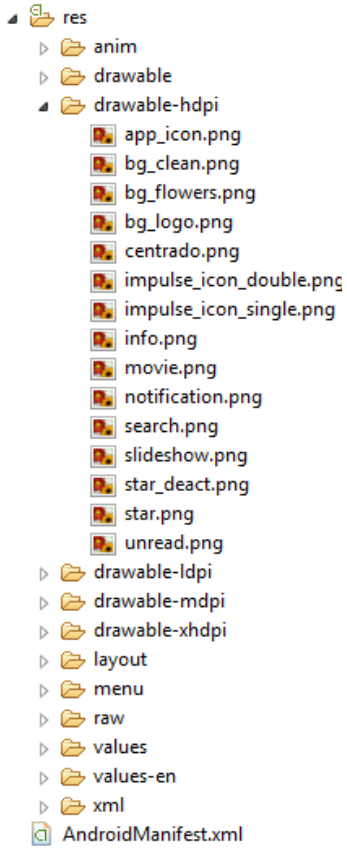

Abbildung 16: Auflösungsoptimierung Ordnerstruktur

7.9 Galerie

Wie im Kapitel 10 beschrieben, wurde eine eigene Galerie-Ansicht implementiert. Durch die Galerie kann mittles Swipe-Gesten navigiert werden.

7.9.1 Gesture Detector

Die *GalleryActivity* besitzt einen *GestureDetector*, welcher Swipe-Bewegungen auf dem Display erkennt. Folgendes Code-Snippet zeigt die Implementation:

```
public boolean onFling(MotionEvent e1, MotionEvent e2,
	float velocityX, float velocityY) {

		int distanceX = (int) (e2.getX() - e1.getX());
		if (isHorizontalSwipe(velocityX, velocityY, distanceX)) {
			if (isRightSwipe(velocityX)) {
				if (currentPosition > 0) {
					swipeRight();
				}
			} else {
				if (currentPosition < urlList.length() - 1) {
					swipeLeft();
				}
			}
			return true;
			} else {
				return false;
				}
		}
```

Code-Snippet 22: GestureDetector (GalleryActivity)

Die Methoden swipeRight() und swipeLeft() zeigen das nächste Bild an.

```
private void swipeLeft() {
	currentImage.startAnimation(slideLeftOut);
	currentPosition++;
	loadCurrentImage();
	currentImage.startAnimation(slideLeftIn);
}
```

Code-Snippet 23: Swipe

Die Methode loadCurrentImage() übergibt dem ImageCache die URL zum nächsten Bild.

```
private void loadCurrentImage() {
	currentImage.setImageDrawable(null);
	try {
		imageLoader.loadImage(urlList.getString(currentPosition),currentImage);
		loadInfoBar();
	} catch (JSONException e) {
		showError();
	}
}
```

Code-Snippet 24: loadCurrentImage()

7.9.2 Image Cache

Damit die Bilder der Impulse dargestellt werden können, müssen diese lokal gecached werden. Dafür wurde die Klasse *ImageCache* erstellt, welche mit AsyncTasks und einer LinkedHashMap einen einfachen Cache implementiert. Damit kein Memory-Leak entsteht, wurde die Grösse des Caches auf 18 Bilder beschränkt (Erfahrungswert), so wird gewährleistet, dass die App auch über lange Zeit nicht zu viel Memory in Anspruch nimmt, oder gar abstürzt.

Die Methode loadImage() schaut anhand der URL, ob das angeforderte Bild noch im Cache verfügbar ist (LinkedHashMap) und lädt anderenfalls mithilfe eines AsyncTasks das Bild vom Server herunter.

```
public void loadImage(String imageurl, ImageView imageview) {
	if (imagCacheMap.containsKey(imageurl)) {
		Drawable drawable = imagCacheMap.get(imageurl);
		imageview.setImageDrawable(drawable);
	} else {
		ImageDownloaderTask imagetask = new ImageDownloaderTask(imageview);
		imagetask.execute(imageurl);
	}
}
```

Code-Snippet 25: loadImage() ImageCache

```
@Override
protected AsyncTaskResult<Drawable> doInBackground(String... params) {
	url = params[0];
	Drawable drawable = null;
	try {
		drawable = loadImageFromUrl(url);
		imagCacheMap.put(url, drawable);
		return new AsyncTaskResult<Drawable>(drawable);
	} catch (MalformedURLException e) {
		return new AsyncTaskResult<Drawable>(e);
	} catch (IOException e) {
		return new AsyncTaskResult<Drawable>(e);
	} catch (Exception e) {
		return new AsyncTaskResult<Drawable>(e);
	}
}
```

Code-Snippet 26: ImageLoaderTask

7.10 Intents

Grundsätzlich werden alle Activities durch explizite Intents gestartet. Grund dafür ist, der statische Programmablauf (Splash→Login →Liste→Detail-Ansicht).
Die Suche nach Impulsen, das Abspielen von Videos und das Anzeigen von Webseiten werden mit Hilfe von impliziten Intents realisiert. Somit ist dem Benutzer freigestellt mit welchen 3rd Party-Apps er den Multimedia-Inhalt anschauen möchte.

8 Externes Design

8.1 Prototypen

Als Vorlage für das externe Design des Android Apps galt die iPhone App. Natürlich gibt es Android Eigenheiten, die speziell berücksichtigt werden mussten.

Mithilfe des Tools „Pencil“ von der Firma Pencil Project, wurden erste GUI Prototypen im Android Stil entworfen. Alle GUI Prototypen sind im Verzeichnis /GUI-Prototypes zu finden.

Abbildung 17: GUI-Prototypen

8.2 GUI-Map

Abbildung 18: GUI-Map zeigt alle Activities der CENTRADO Android App. Die Pfeile stellen den Programmablauf dar. Bei den Übergängen ist jeweils die auslösende Aktion beschrieben.

Abbildung 18: GUI-Map

9 Architektur und Design

Dieses Kapitel erläutert den Aufbau, sowie die Funktionsweise der App. Ausserdem wird aufgezeigt, wie die verschiedenen Komponenten miteinander interagieren.

9.1 Überblick

Das folgende Diagramm gibt eine Einsicht in alle involvierten Komponenten. Die CENTRADO Einheit wurde in dieser Arbeit mithilfe des Testservers simuliert.

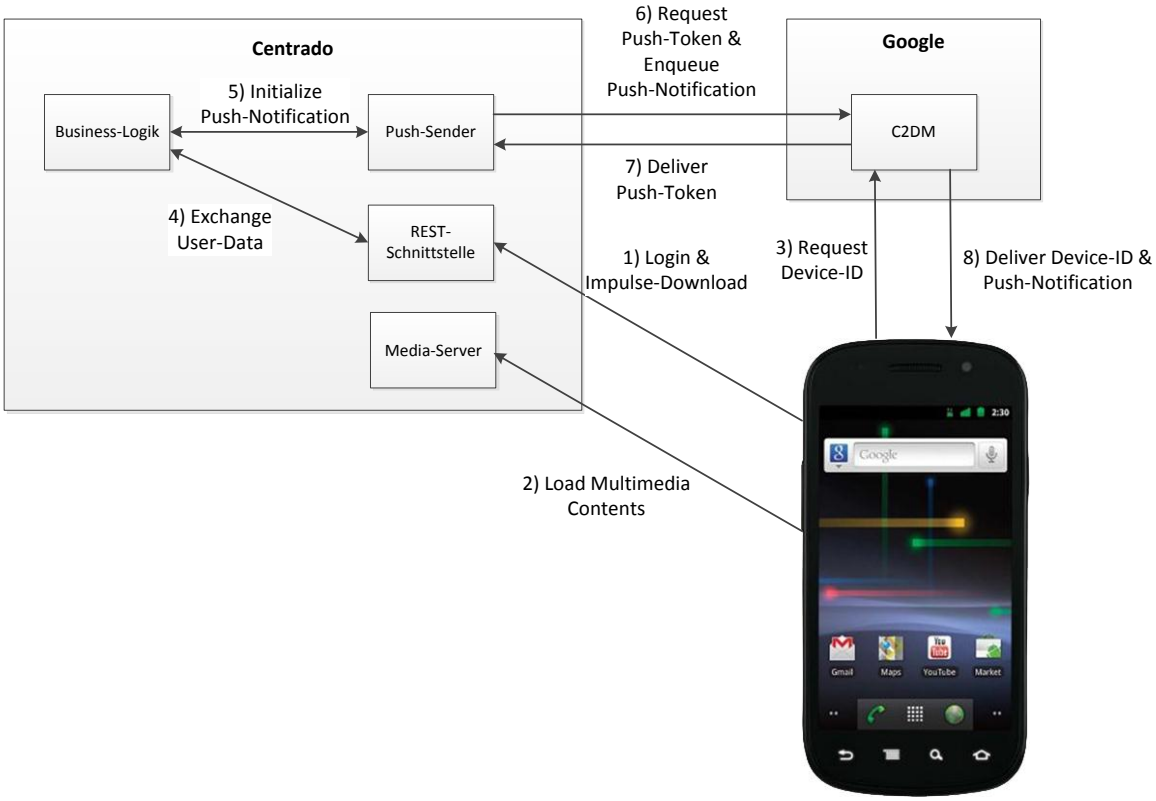

Abbildung 19: Architektur Übersicht

9.1.1 Komponenten

Einzelne Komponenten aus Abbildung 19: Architektur Übersicht werden in der Tabelle 12: Architektur Komponenten genauer erläutert.

Komponente	Beschreibung
Android App	Client unterstützt Polling und das Empfangen von Push-Notifications.
C2DM	Google Komponente zum Versenden von Push-Notifications
Push-Sender	Java Server Applikation, welche die Push-Notifications beim Google-Service initiiert.
REST-Schnittstelle	Schnittstelle, welche die Impulse zur Verfügung stellt. (Über HTTP-Basic-Authentication gesichert)
Media-Server	Beinhaltet die multimedialen Inhalte der Impulse. Videos werden extern auf YouTube gehostet.
Business-Logik	Verwaltet die Benutzer und Impulse (Datenbank).

Tabelle 12: Architektur Komponenten

9.1.2 Starten der Applikation

Die CENTRADO Android App startet mit einem „Splash-Screen“. Danach befindet sich der Benutzer auf der Login-Ansicht. Diese authentifiziert sich mit dem Server. Sind die Login-Daten fehlerhaft oder besteht keine Verbindung mit dem Internet, wird eine Fehlermeldung angezeigt und der Benutzer wird zur erneuten Authentifizierung aufgefordert.
Sobald die Authentifizierung erfolgt ist, wird der Client für den Push-Service (C2DM) registriert. Dazu wird ein Device-Token vom Google-Service angefordert und an den Server übermittelt. Das C2DM-Framework setzt voraus, dass auf dem Gerät ein Google-Account registriert ist. Ist dies nicht der Fall, wird eine Fehlermeldung angezeigt, die App kann jedoch trotzdem weiter verwendet werden.
Anschliessend werden die für den Benutzer vorbereiteten Impulse vom Server geladen und in die lokale Datenbank gespeichert. Dieser Ablauf ist in Abbildung 20: Ablauf beim Starten der App dargestellt.

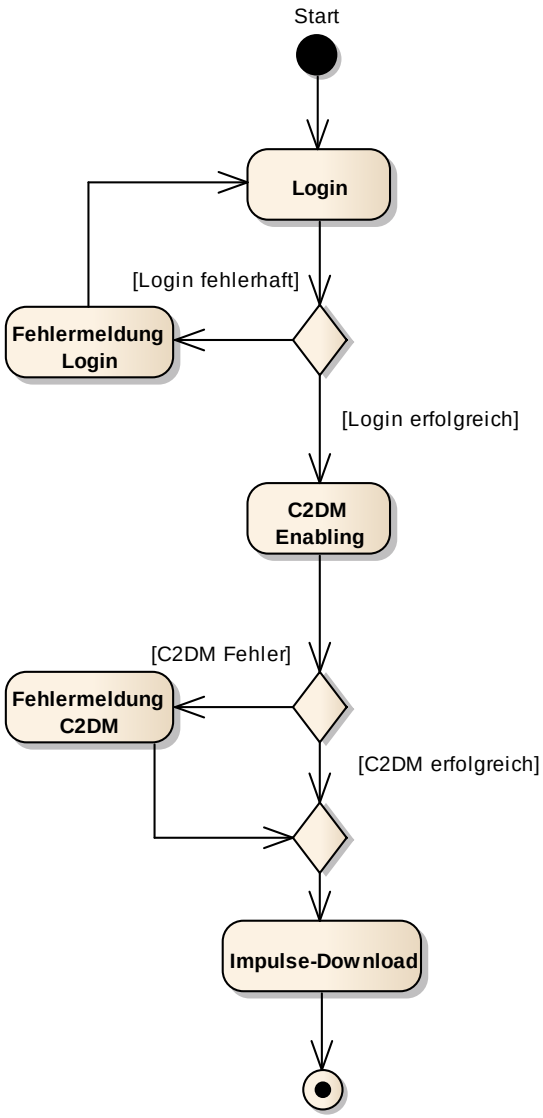

Abbildung 20: Ablauf beim Starten der App

9.2 Logical View

Die Architektur-Entscheide werden im Kapitel „Design-Entscheide“ genauer diskutiert. Die Performance ist bei der Implementation von mobilen Applikationen von grosser Bedeutung. Trotzdem ist die App in mehrere Schichten aufgeteilt.

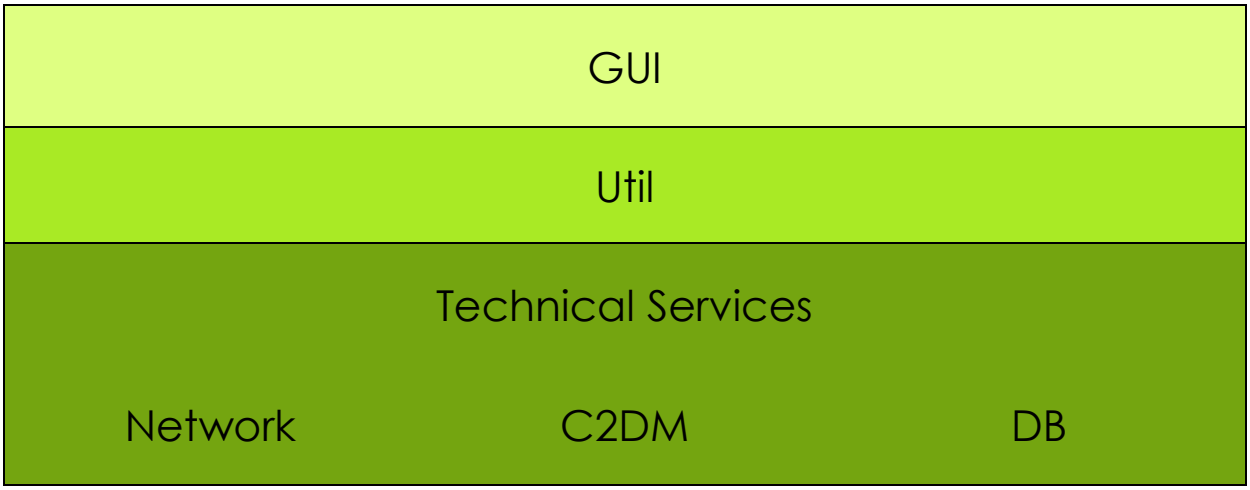

Abbildung 21: Schichten

Die oberste Schicht stellt der Presentation-Layer (GUI) dar, darunter liegt der Util-Layer, welcher diverse Hilfsklassen umfasst (z.B. JSON-Parser). Darunter ist die „Technical-Services“ Schicht, mit Datenbank und Netzwerk.

9.2.1 Package-Struktur

Die Abbildung 22: Package-Diagramm zeigt die Package-Struktur in der Schichtenanordnung.

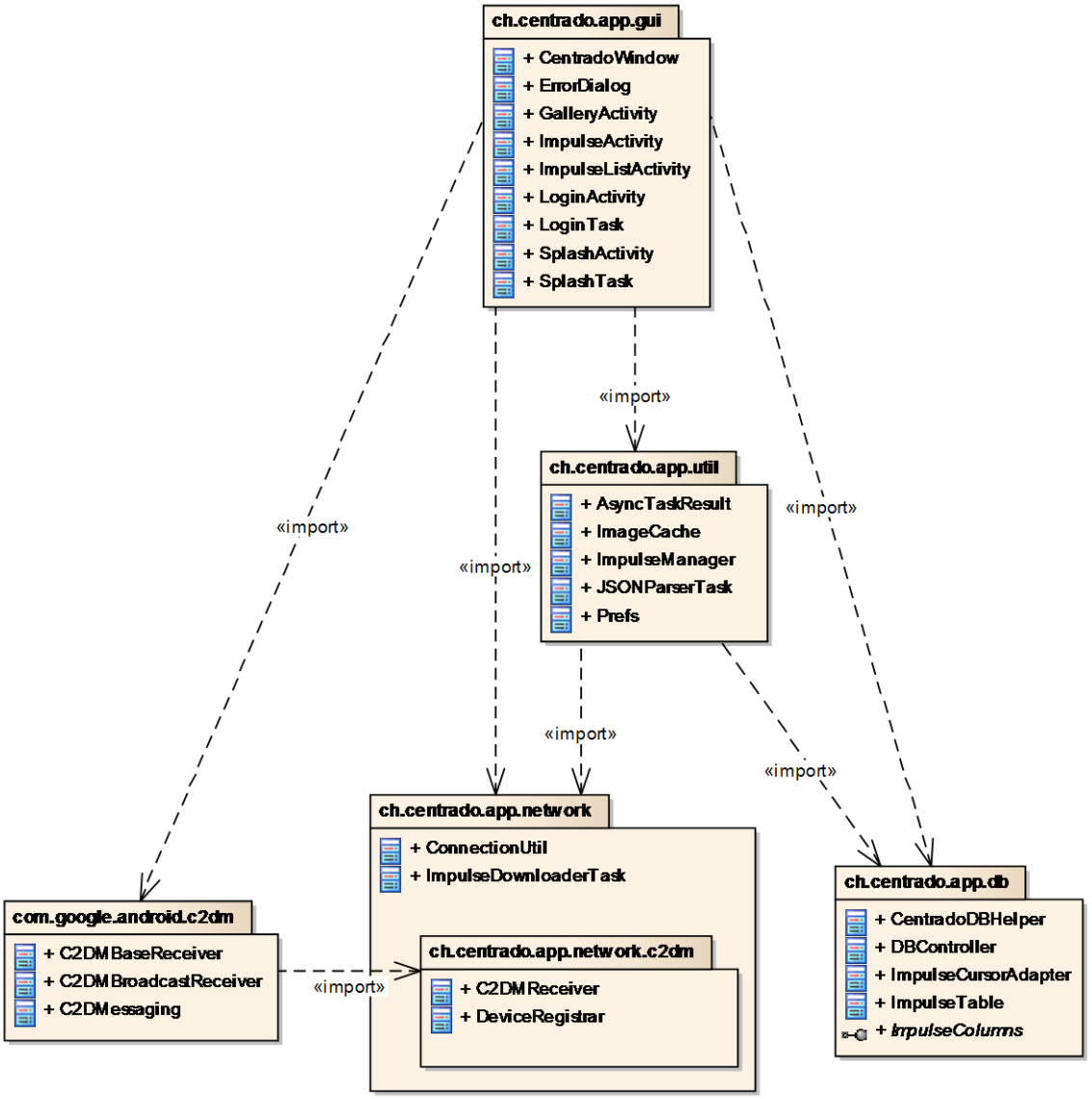

Abbildung 22: Package-Diagramm

9.2.2 GUI

Wie in den Android-Grundlagen bereits diskutiert, erleichtert Android die Kapselung der GUI-Elemente. Der eigentliche „Presentation-Layer“ sind die XML-Dateien, welche die Layouts der Benutzeroberflächen definieren.
Die Activities implementieren die Logik (Controller des MVC-Patterns) der Benutzeroberfläche. Die folgenden Klassen befinden sich im Package ch.centrado.app.GUI:

Klasse	Aufgabe
CentradoWindow	Das *CentradoWindow* definiert das „Standard-Layout“ der Activities (Centrado Titel), alle nachfolgenden Activities leiten von dieser Klasse ab.
LoginActivity	Die *LoginActivity* gibt dem Benutzer die Möglichkeit, sich am System anzumelden. Ausserdem wird eine Hilfe (Weblink)zur Verfügung gestellt.
ImpulseListActivity	Diese Activity zeigt die empfangenen Impulse in Form einer Liste an. Der *ImpulseCursorAdapter* stellt die Schnittstelle zur Datenbank dar.
ImpulseActivity	Die *ImpulseActivity* ist die Detail-Ansicht der Impulse. Je nach Art des Impulses verändert sich der Aufbau und das Layout der Activity (Video, Bild usw.)
GalleryActivity	Die *GalleryActivity* dient zur Darstellung der Bild-Impulse. Sie besitzt einen *ImageCache*, welcher die Bilder vom Media-Server herunterlädt und zwischenspeichert.
ErrorDialog	Wird von verschiedenen Tasks verwendet um den Benutzer über Fehler zu informieren.
OnImpulseClickListener	Dieser Event-Handler zeigt bei einem Klick auf einen Impuls die dazugehörige Detail-Ansicht (*ImpulseActvitiy*) an.

Tabelle 13: GUI-Klassen

Die Abbildung 23: GUI-Layer zeigt das Klassendiagramm des GUI-Layers.

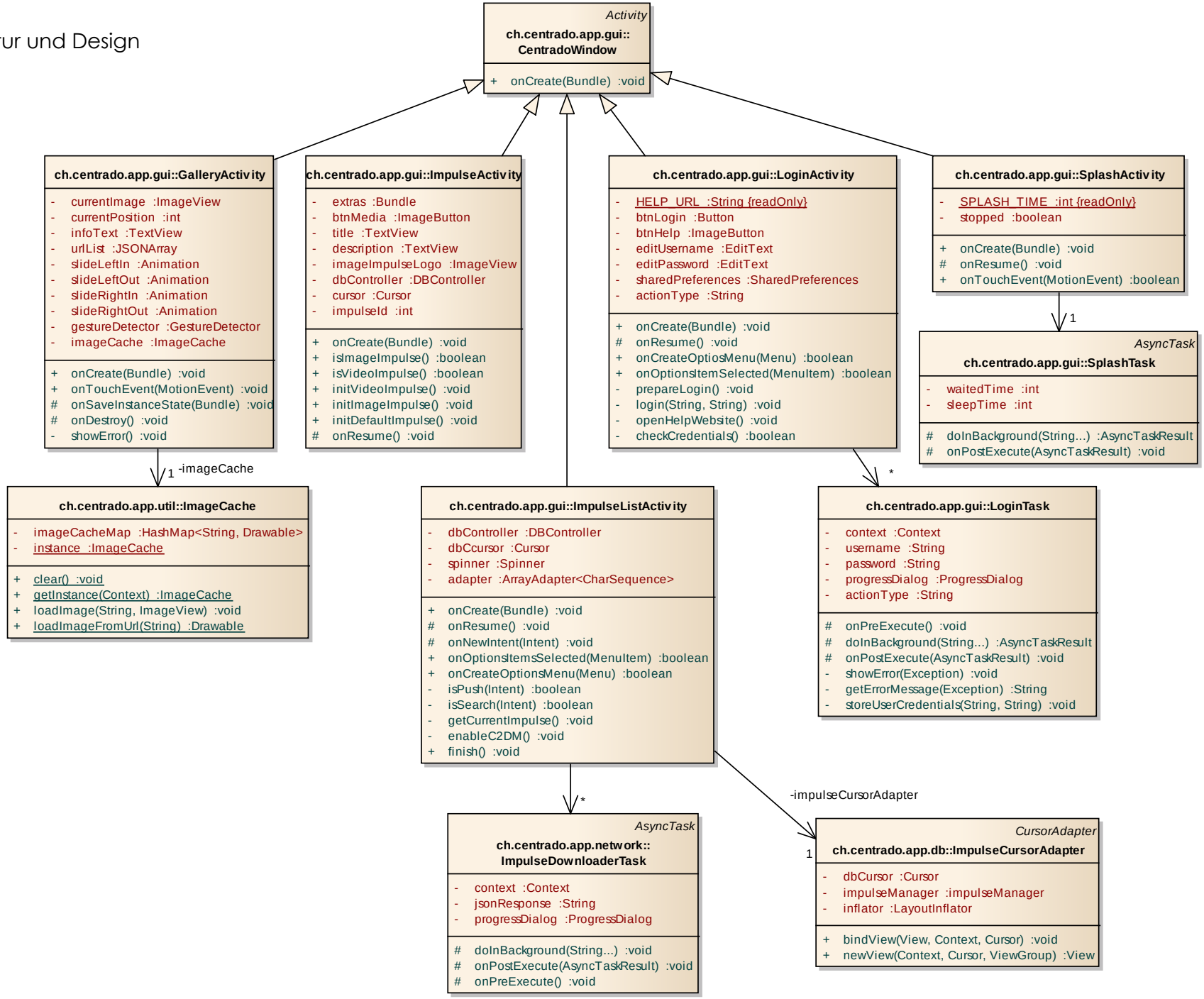

Abbildung 23: GUI-Layer

9.2.3 DB

Das Package ch.centrado.app.db besteht aus vier Klassen und einem Interface.

Klasse	Aufgabe
ImpulseColumns	Auf die Spaltennamen der Datenbank wird oft zugegriffen. Darum hat es sich laut Android Dokumentation bewährt, für jede Tabelle ein Interface zu erstellen. In diesem Interface werden die Spaltennamen als Konstanten definiert.
ImpulseTable	Diese Klasse kapselt die SQL Statements, welche auf die Datenbank abgesetzt werden.
DBController	Der *DBController* ist die Schnittstelle zwischen Activites und Datenbank. Hier werden die Zugriffe auf die Datenbank gekapselt.
CentradoDBHelper	Der *CentradoDBHelper* erbt von *SQLiteOpenHelper*. Diese Klasse regelt die Erstellung und den Zugriff auf die Datenbank.
ImpulseCursorAdapter	Diese Klasse mappt die Datenbankeinträge auf die Impulsliste. Anhand des Cursors (Referenz auf Impuls in DB) wird der Inhalt der List-Items aus der DB geholt.

Tabelle 14: DB-Klassen

Die Abbildung 24: DB-Layer zeigt das Klassendiagramm des DB-Layers.

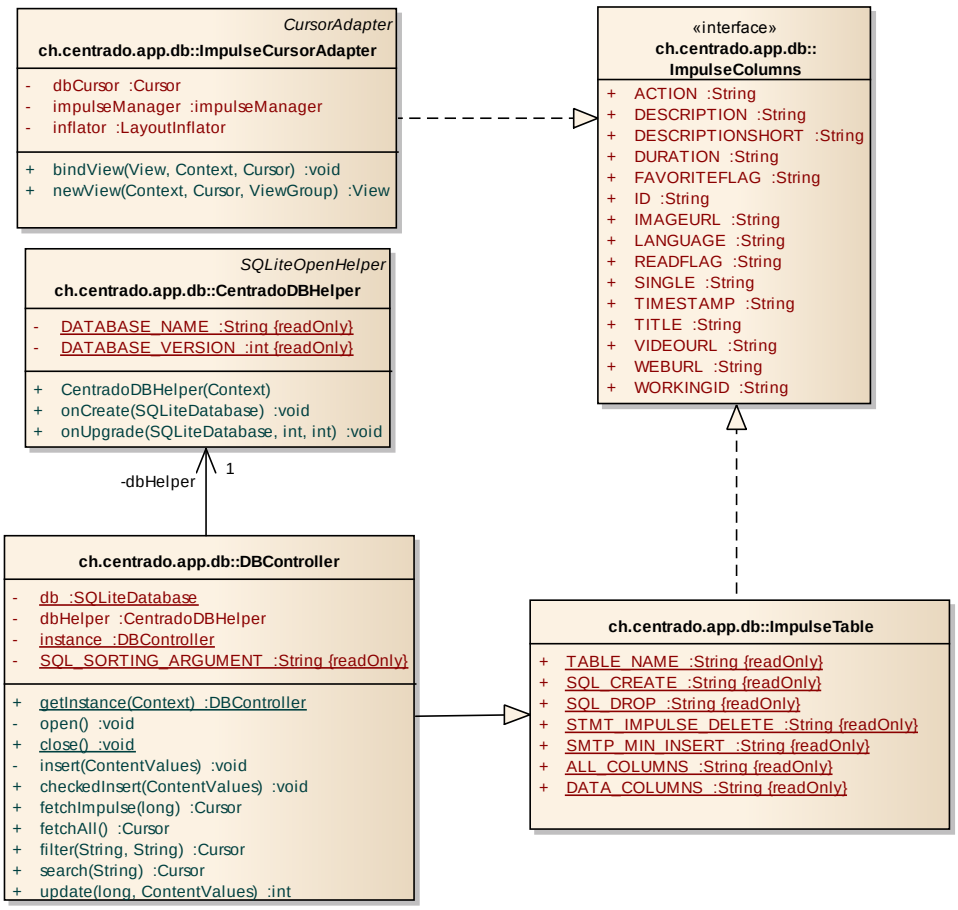

Abbildung 24: DB-Layer

9.2.4 Network

Der Network-Layer ist Teil der Technical Services und umfasst alle Klassen, die den Zugriff auf den CENTRADO Server oder die Google-Services ermöglichen.
Das Package ch.centrado.app.network.C2DM enthält die von uns implementierten C2DM-Klassen. Der Source-Ordner C2DM enthält, die von Google implementierten C2DM-Klassen, welche unter der GNU GPL stehen und frei verwendbar sind.

Klasse	Beschreibung
ConnecionUtil	Diese Klasse ist für den Auf- und Abbau der Verbindung (Login, Logout), sowie das Absetzen der HTTP-GET-Requests an die REST-Schnittstelle verantwortlich.
ImpulsDownloaderTask	Kapselt den Download der Impulse in einen eigenen Thread, damit der GUI-Thread nicht blockiert wird.
C2DMReceiver	Stellt eine Implementierung der *C2DMBaseReceiver* Klasse dar. Dieser Service ist verantwortlich auf den Empfang einer Push-Notification zu reagieren.
DeviceRegistrar	Sobald die DeviceRegistrationID, welche nach dem erfolgreichen Login angefordert wird, vom Google-Service erhalten wird, registriert sich der *DeviceRegistrar* bei der REST-Schnittstelle.

Tabelle 15: Network-Klassen

Die Abbildung 25: Network-Layer zeigt das Klassendiagramm des Network-Layers.

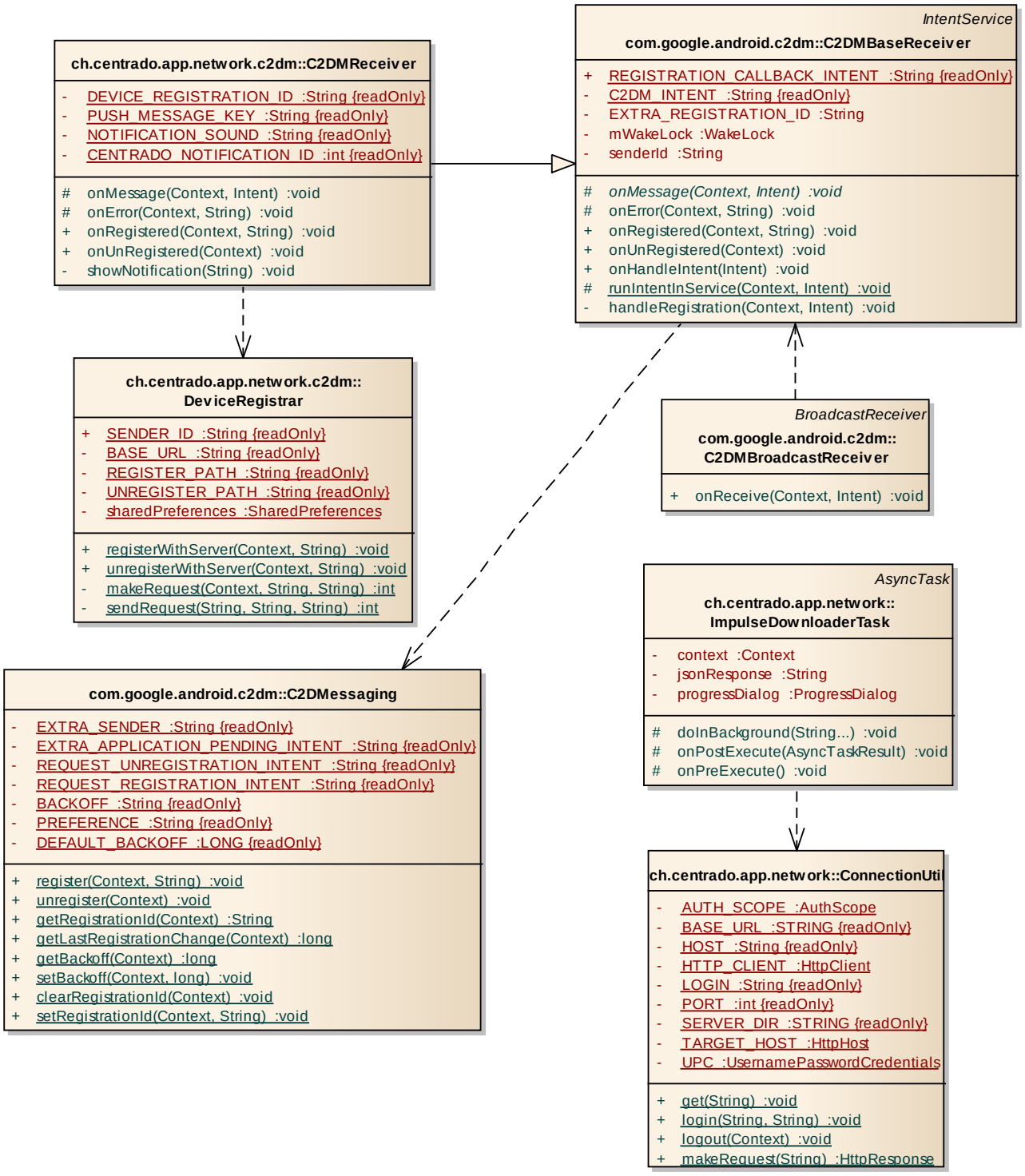

Abbildung 25: Network-Layer

Die Dependancy-Pfeile visualisieren statische Methodenaufrufe.

9.2.5 Util

Im Util-Layer befinden sich diverse Hilfs-Klassen. (*Prefs*, *JSONParserTask*, *ImageCache* usw.)

Klasse	Beschreibung
Prefs	Die Benutzerdaten werden nach einem erfolgreichen Login in den Shared Preferences gespeichert. Diese Klasse kapselt den Zugriff auf die Shared Preferences.
ImageCache	In der Klasse *ImageCache* werden die benötigten Bilder heruntergeladen und abgespeichert.
ImpulseManager	Die Impulsmanager Klasse kapselt die Abfrage- und Modifiziermethoden der Impulse. Es kann abgefragt werden ob ein Impuls als Favorit markiert wurde oder ob es sich um einen Single-Impulse handelt, was sich auf das Icon der *ImpulsActivity* auswirkt. Ebenfalls können Favoriten- und Read-Flags gesetzt werden.
JSONParserTask	Diese Klasse ist für das Parsen der JSON-Files zuständig. Die empfangenen JSON-Objekte werden in Impulse umgewandelt und in der Datenbank abgelegt.
AsyncTaskResult	Wird als Rückgabe-Typ von AsyncTasks verwendet. Sie ermöglicht es auf Exceptions aus den Hintergrund-Threads zu reagieren (Fehlermeldung).

Tabelle 16: Util-Klassen

Die Abbildung 26: Util-Layer zeigt das Klassendiagramm des Util-Layers.

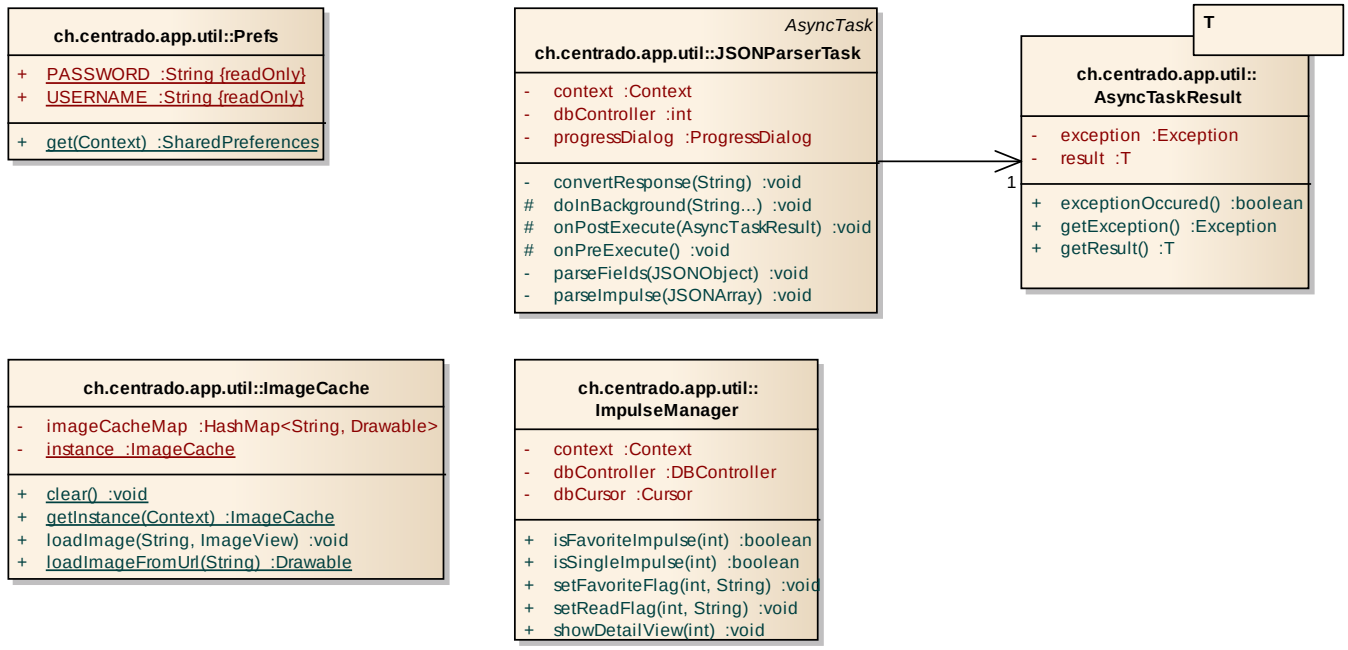

Abbildung 26: Util-Layer

9.3 Deployment View

Die Deployment View zeigt die drei Device-Nodes, welche für die Anwendung relevant sind. Die Kommunikation beruht auf dem HTTP Protokoll.

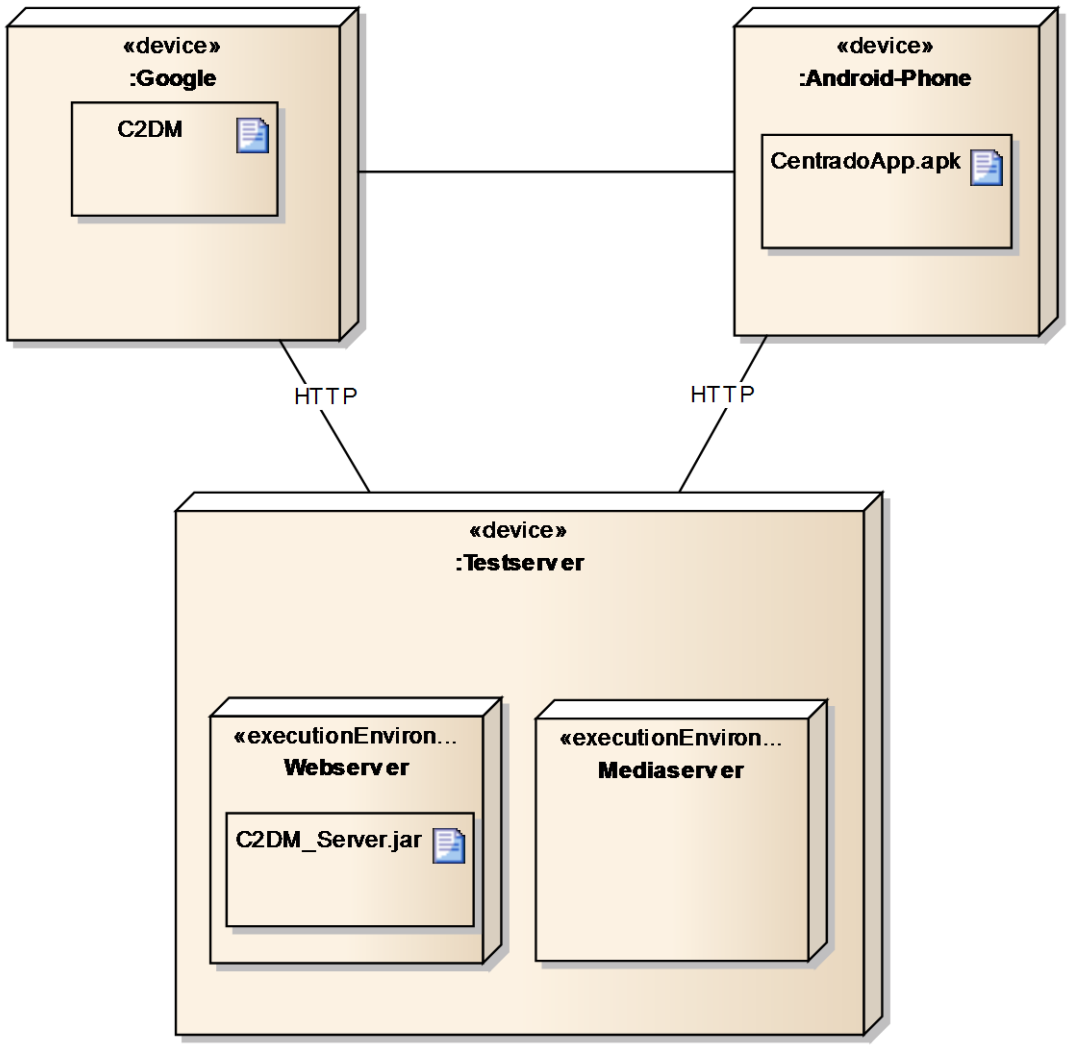

Abbildung 27: Deployment-Diagramm

9.4 Process View

Android instanziiert für jede Applikation einen eigenen Prozess. Dieser beinhaltet standardmässig einen GUI-Thread. Damit bei zeitintensiven Aufgaben nicht die App einfriert, arbeitet die Centrado-App mit zusätzlichen Threads. Die Android Library umfasst ein Konzept, welches es erlaubt elegant eine Multithread-Applikation zu realisieren.

9.4.1 AsyncTasks

Die abstrakte Klasse *AsyncTask<Params, Progress, Result>* bietet die Möglichkeit eine Aufgabe im Hintergrund auszuführen und sobald diese fertig ist, die Resultate auf dem GUI darzustellen. Das wäre sonst problematisch, weil nur der GUI-Thread Änderungen auf der Benutzeroberfläche vornehmen darf.
Für jede Instanz eines solchen Tasks wird ein Hintergrund-Thread erzeugt.

Die *AsyncTask<Params, Progress, Result>* Klasse umfasst folgende Methoden:

Methode	**Beschreibung**	**Ausführender Thread**
onPreExecute()	Diese Methode wird direkt nach dem Start des Tasks ausgeführt. In der Regel wird hier der Task vorbereitet und zum Beispiel ein Ladebalken auf der Benutzeroberfläche angezeigt.	GUI-Thread
doInBackground(Params…)	Sobald die Methode *onPreExecute()* durchgelaufen ist, startet der Hintergrund-Thread. Diese Methode beinhaltet die zeitintensiven Schritte.	Hintergrund-Thread
publishProgress(Progress…)	Diese Methode wird dazu verwendet, um Fortschritte oder Log-Daten aus der *doInBackground(Progress…)* Methode an die GUI zu melden.	Hintergrund-Thread
onProgressUpdate(Progress…)	Nachdem die *publishProgress(Progress…)* Methode aufgerufen wurde, wird die *onProgressUpdate(Progress…)* vom GUI-Thread ausgeführt.	GUI-Thread
onPostExecute(Result)	Diese Methode wird nach Beendigung des Tasks ausgeführt und erlaubt es die Resultate auf der Benutzeroberfläche anzuzeigen.	GUI-Thread

Tabelle 17: AsyncTask-Methoden

9.4.2 AsyncTaskResult

Die Klasse *AsyncTaskResult<T>* erlaubt es die Resultate sowie Exceptions (bzw. Fehlermeldungen) aus der *doInBackground(Params…)*Methode sauber auf der Benutzeroberfläche anzuzeigen. Es ist vorteilhaft als Rückgabe-Typ eines *AsnycTasks* immer die *AsyncTaskResult* Klasse zu verwenden.
Über die Methode *exceptionOccured()*kann abgefragt werden, ob im Hintergrund-Thread eine Exception aufgetreten ist und allenfalls eine Fehlermeldung auf der Benutzeroberfläche angezeigt werden.
Ist keine Exception aufgetreten kann das Resultat normal verarbeitet werden. Ohne dieses Konstrukt wäre es schwierig, Fehlermeldungen zu den Exceptions, die in einem Hintergrund-Task auftreten, auf der Benutzeroberfläche anzuzeigen.

Die Klasse *AsyncTaskResult<T>* beinhaltet folgende Methoden und Konstruktoren:

Methode	Beschreibung
AsyncTaskResult(T result)	Konstruktor zum Erstellen eines „Result-Objects“
AsyncTaskResult(Exception e)	Konstrukor zum Erstellen eines „Exception-Result-Object“
boolean exceptionOccured()	Diese Methode liefert true falls es sich um ein „Exception-Result-Object“ handelt.
T getResult()	Liefert das Resultat, falls vorhanden
Exception getException()	Liefert die Exception, falls vorhanden

Tabelle 18: AsyncTaskResult-Methoden

9.4.3 Beispiel LoginTask

Die Abbildung 28: LoginTask zeigt den Ablauf des LoginTasks.

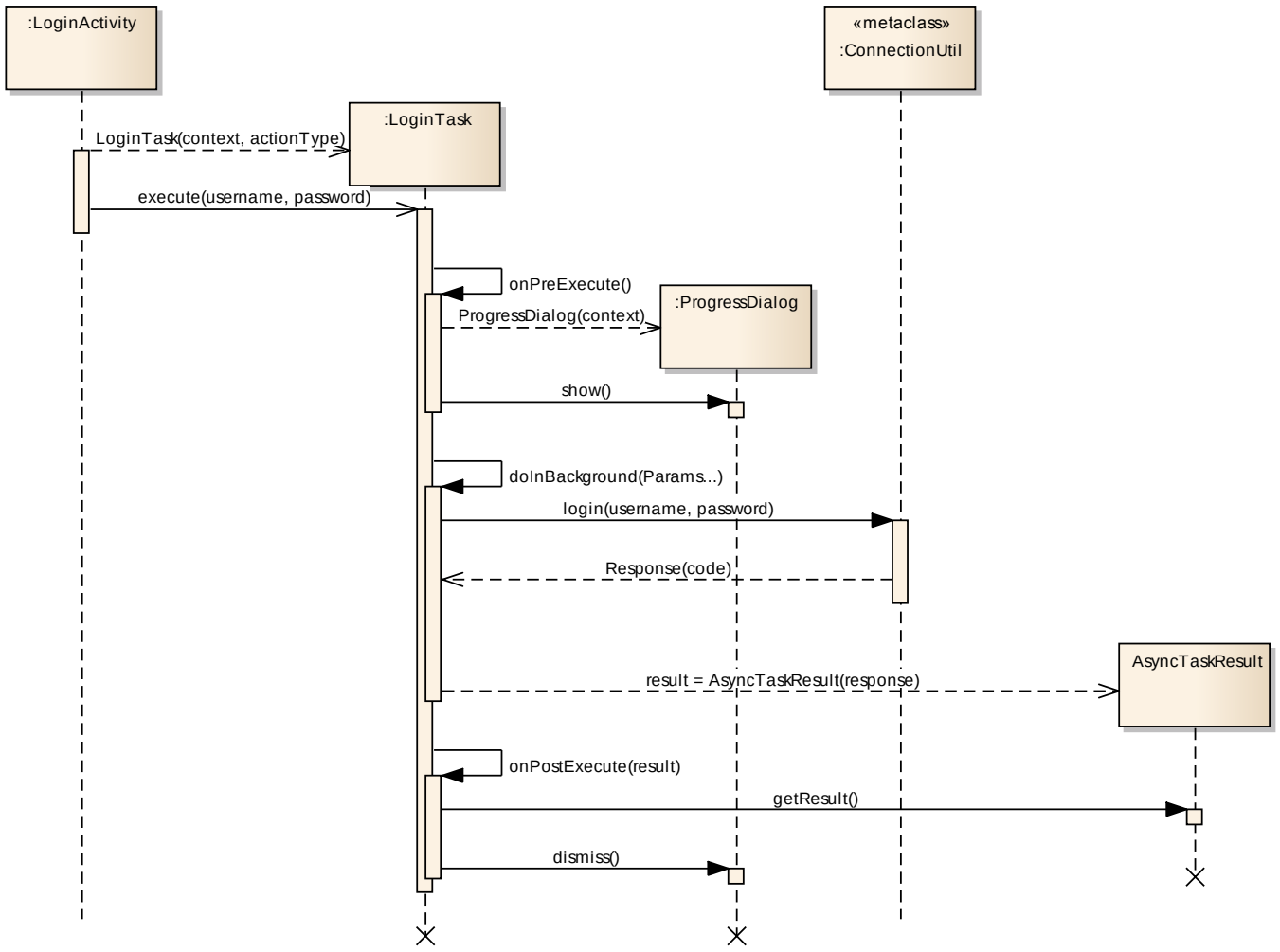

Abbildung 28: LoginTask

Ablauf:

1. *LoginActivity* instanziiert (Konstruktor) und startet den *LoginTask* (*execute()*)
2. *LoginTask* führt die Methode *onPreExecute()* aus und zeigt einen *ProgressDialog* an.
3. *LoginTask* führt *doInBackGround(Params…)* Methode aus und instanziiert ein AsyncTaskResult Objekt.
4. *LoginTask* führt *onPostExecute(result)* aus, dieser schliesst den ProgressDialog wieder.
   Hier wird zusätzlich mit *exceptionOccured()* auf dem *AsyncTaskResult* überprüft, ob im Hintergrund-Thread eine Exception aufgetreten ist. Falls dies der Fall ist, wird eine Fehlermeldung angezeigt, ansonsten wird die *ListActivity* angezeigt. (weggelassen)

Die *publishProgress(Progress…)* und *onProgressUpdate(Progress…)* werden nicht gebraucht, da ein „zustandsloser" *ProgressDialog*, anstelle einer *ProgressBar* verwendet wird.

9.4.4 Anwendung

Die CENTRADO App umfasst im Ganzen fünf solche AsyncTasks:

Task	Beschreibung
SplashTask	Anzeigen des Splash-Screens
LoginTask	Authentifiziert Benutzer beim Server
ImpulseDownloaderTask	Lädt die Impuls-Liste des Benutzers vom Server herunter.
JSONParserTask	Verarbeitet den JSON-String (Antwort des Servers) und erstellt die Datenbank-Einträge (Impulse).
ImageDownloaderTask	Lädt die Bilder in der Galerie-Ansicht herunter

Tabelle 19: CENTRADO-Tasks

10 Design-Entscheide

10.1 Einleitung

Während der Arbeit wurden innerhalb des Projektteams, sowie auch an den wöchentlichen Meetings einige Design-Entscheide getroffen. Diese Entscheide hatten Auswirkungen auf Technologien, Use Cases und das externe Design. Bei den Entscheiden ist jeweils in Klammern angegeben, welche nichtfunktionalen Anforderungen dabei beachtet und umgesetzt wurden.

10.2 Entscheide

10.2.1 Push vs. Poll

Eine Alternative zum Push-Mechanismus wäre eine Polling Lösung. Dabei könnte der Benutzer selbstständig überprüfen ob ein neuer Impuls für ihn vorhanden ist. Eine andere Alternative, jedoch auch mit Polling wäre, wenn die App im Hintergrund zu bestimmten Zeiten ein Polling ausführt.

10.2.1.1 Push

Der Push-Mechanismus spart den Akku des Smartphones. Die App wird erst benachrichtigt, sobald ein neuer Impuls vorliegt. Ein häufiges Polling würde den Akku belasten und zusätzlichen Traffic-Overhead produzieren.

Viele Android Applikationen von Google nutzten bereits Push um ihre Apps mit aktuellen Daten zu versorgen (Gmail, Kontakte, Calender). [AndroidDev]

10.2.1.2 Polling

Beim Polling ist es schwierig, eine gute Abfragefrequenz zu finden: Wenn man zu viel pollt, sind keine neuen Impulse vorhanden und es würde ein Traffic-Overhead generiert werden. Wenn die Wartezeit zwischen den Polls zu gross ist, hätte der Benutzer keine aktuellen Impulse.
Dieses Poll Problem könnte jedoch gelöst werden, indem die Impulse für alle Benutzer, jeden Tag zur gleichen Zeit online gestellt werden. Die Apps könnten anschliessend im Hintergrund pollen. Somit wäre gewährleistet, dass kein Overhead entsteht, die Impulse jedoch trotzdem aktuell sind.
Diese Lösung wäre jedoch, im Gegensatz zu Push, wieder änderungsanfälliger. Wenn der Zeitpunkt der Freigabe der Impulse ändert, müsste die App aktualisiert werden.

Dieser Designentscheid wurde vom Auftraggeber zu Beginn der Arbeit getroffen.

10.2.2 Datenbank

Android bietet den Programmierern die Möglichkeit, eine SQLite Datenbank zu verwenden. SQLite bietet für dieses Projekt folgende Vorteile (NF10):

- SQLite unterstützt eine umfangreiche SQL-Syntax und benötigt nur sehr wenig Speicherplatz
- SQLite muss nicht installiert werden

Aus diesen Gründen wurde für die Speicherung von Impulsen SQLite verwendet.

10.2.3 Textsuche in Impulsliste

Auf der Impulsliste sollte man die Möglichkeit haben, nach spezifischen Impulsen suchen zu können. In Android gibt es zwei Möglichkeiten, um eine Liste zu durchsuchen.

- Ab Android Version 3.0 (API Level 11 oder höher) ist ein **Search Widget** verfügbar, welches als Action View in der Action Bar eingepflegt werden kann. Das Widget kann aber auch beliebig in einem Layout platziert werden.
- Für Apps, die mit einer Android Version tiefer als 3.0 entwickelt wurden, steht ein **Search Dialog** zur Verfügung. Welcher in den meisten Apps verwendet wird, und daher vor Version 3.0 als Best Practise gilt. Es ist zu vermeiden eine eigene Textbox zu erstellen, welche die Suche koordiniert.

Da die CENTRADO Android App unter API Level 8 entwickelt wurde (Version 2.2) wurde der Search Dialog verwendet. Der Search Dialog kann mithilfe des Search Buttons, welcher auf den meisten Geräten verfügbar ist, angezeigt werden. Da nicht alle Geräte einen Search-Button besitzen, ist es wichtig, dass die Suche auch über einen Menü-Eintrag gestartet werden kann (NF01, NF02).

Abbildung 29: Search Dialog [AndroidDEV]

10.2.4 Galerie

Ursprünglich sollte die Bilder-Ansicht über implizite Intents mit der vorinstallierten Galerie-App realisiert werden. Dies hätte zur Folge gehabt, dass nach dem Schliessen der App die Bilder immer noch in der Galerie einsehbar gewesen wären. Aus diesem Grund wurde die Galerie-Ansicht mit einer eigenen Activity (*GalleryActivity*) realisiert. So kann sichergestellt werden, dass der Zugriff auf die Bilder nur von authentifizierten Benutzern erfolgt. (NF01, NF05)

10.2.5 Video

Da im Android-System mindestens ein Browser und/oder das YouTube-App vorinstalliert sind, wurden für die Video-Ansicht implizite Intents verwendet. Der Benutzer kann somit eine seiner vorinstallierten Apps verwenden um die CENTRADO-Videos zu schauen (NF02). Implizite Intents sind ein einfaches Konzept und können zeitaufwendige Implementationen ersparen.
Eine mögliche Alternative wäre die Implementation eines integrierten YouTube-Players gewesen, dies wäre aber umständlicher und aufwendiger gewesen.
Durch das Verwenden des vorinstallierten YouTube Apps werden jedoch auch andere YouTube-Videos sichtbar (Verlinkung „ähnliche Videos“).

10.2.6 Favoriten

Wie üblich wurden ein grauer und ein gelber Stern verwendet, welche den aktuellen Status des Impulses anzeigen. Damit sich der Benutzer schnell zu Recht findet, wurden für die Platzierung des Sterns andere Apps analysiert. Als Musterbeispiel gilt die Gmail App auf den Android Geräten. (NF01, NF02, NF03)

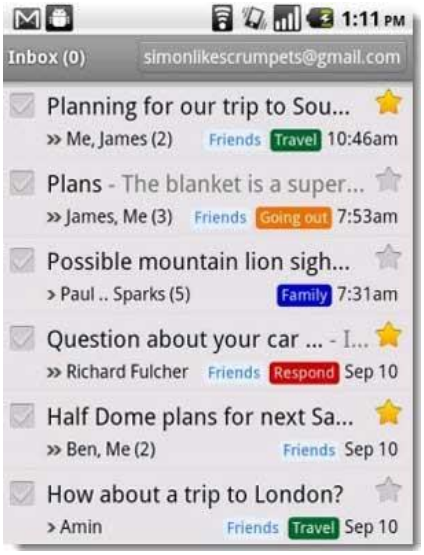

Abbildung 30: Gmail App [AndroidPIT]

10.2.7 SSL (NF06)

Laut NF06 (Kapitel 4.5.3 Sicherheit) muss die Übertragung zwischen Server und Android Phone immer verschlüsselt werden.
Diese nichtfunktionale Anforderung, wurde bereits in der iPhone Arbeit [BAiPhone] nicht umgesetzt. Daher gab es für dieses Projekt keine Referenz, wie die Schnittstellen-Kommunikation aussieht. Aus diesem Grund wurde auch in dieser Arbeit keine gesicherte Verbindung zwischen Android Gerät und Testserver implementiert.

10.2.8 Filterung

Da für die Filterung ein neues GUI-Element eingeführt wurde, mussten neue GUI-Prototyen erstellt werden, damit die Platzierung und Form diskutiert werden konnte.

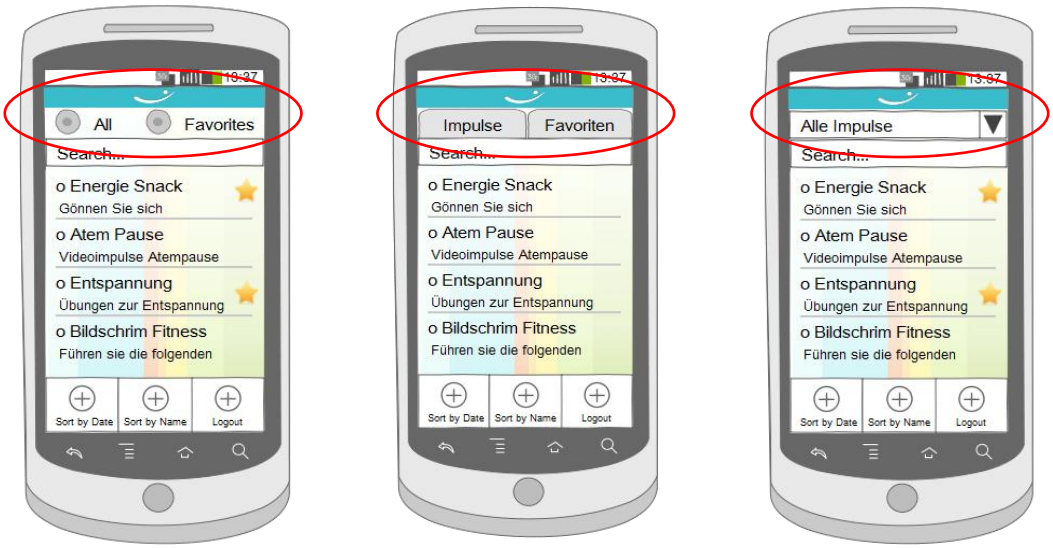

Abbildung 31: GUI Prototypen (Filterung)

Die erste Version mit den Radio Buttons hätte keinen Platz für weitere Filteroptionen geboten. Ebenso die zweite Version mit den Tabs.
Die dritte Version fand am meisten Anklang und stellte sich auch als praktischste heraus. Dabei wurde oberhalb der Liste ein Spinner eingefügt. Beim Antippen des Spinners werden die zur Verfügung stehenden Filter-Optionen angezeigt. Somit kann einfach und effiziert gefiltert werden (NF01).

11 Qualitätssicherung

11.1 Unit-Tests

Zu Beginn der Arbeit wurde diskutiert, welche Klassen per Unit-Tests getestet werden können. Da keine Business-Logik vorhanden ist, konnte diese auch nicht getestet werden. Bei der Implementation der verschiedenen Technologien wurden keine Vorkehrungen getroffen, die die Simulation von Technologien unterstützt hätte. Dies machte es am Schluss der Arbeit unmöglich, die vielen Technologien zu Mocken und anschliessend per Unit-Tests zu testen. Aus diesem Grund wurde grossen Wert auf umfängliche Systemtests gelegt.

11.2 Systemtest

Die Android Programmierung unterscheidet sich in einem Punkt sehr stark von der Entwicklung auf dem iPhone. Bei Android gibt es eine Vielfalt von verschiedenen Geräten, welche von unterschiedlichen Herstellern produziert wurden. Die Bildschirme dieser Geräte können in vier unterschiedliche Auflösungen eingeteilt werden (siehe Kapitel 6.3). Ebenfalls gibt es mittlerweile viele verschiedene Android Versionen, die sich teilweise sehr stark von der vorherigen Version unterscheiden (siehe Kapitel 6.2).

Das Benutzungsprinzip der CENTRADO Android App, die Android Eigenheiten, sowie die unterschiedlichen Technologien können am besten auf korrektes Verhalten geprüft werden, indem umfangreiche Systemtests durchgeführt werden.

11.2.1 Vorgehensweise

Um die Systemtests zu spezifizieren, musste zuerst festgelegt werden, was genau getestet werden soll:

- GUI
- Use Cases
- Optionale Anforderungen
- Sprachen

Ebenfalls musste festgelegt werden, unter welchen Umständen das produktive App benutzt werden kann. Daraus wurden Testbedingungen abgeleitet, unter denen die Use Cases durchgeführt wurden.

- Android Versionen
- Auflösungen
- Verbindung (WLAN, 2G, 3G)
- Spracheinstellung

Im Anhang ist das Systemtest-Protokoll aufgeführt. Dabei wird jeweils erwähnt, welches Ziel mit diesem Test verfolgt wird. Ebenfalls werden Angaben zu den Testbedingungen, Testgeräten, den Ergebnissen, sowie den Massnahmen gemacht.

11.3 Code Reviews

Um die Qualität des Codes sicherzustellen, wurden wöchentliche Code-Reviews durchgeführt, in denen der neue Code innerhalb des Projektteams besprochen und refactored wurde.
Der Code wurde ausserdem mit unterschiedlichen Tools analysiert (z.B. Find Bugs und State of Flow Metrics).

11.4 Codestatistik

Die Codestatistiken wurden mit Hilfe des Eclipse-Plugins State of Flow Metrics ermittelt (siehe Tabelle 20: Codestatistik).

Statistik	Wert
# Packages	6
# Classes	35
TLOC	1337

Tabelle 20: Codestatistik

Dabei ist zu beachten, dass die XML-Dateien (Layouts usw.), sowie der Server-Code nicht ausgewertet wurden.

11.5 GUI Review

Nachdem ein Grossteil der GUIs designet wurde, sind Screenshots von der ganzen App gemacht worden. Diese wurden anschliessend von der Grafik-Abteilung der Firma Crealogix analysiert und sind mit Verbesserungsvorschlägen versehen worden.

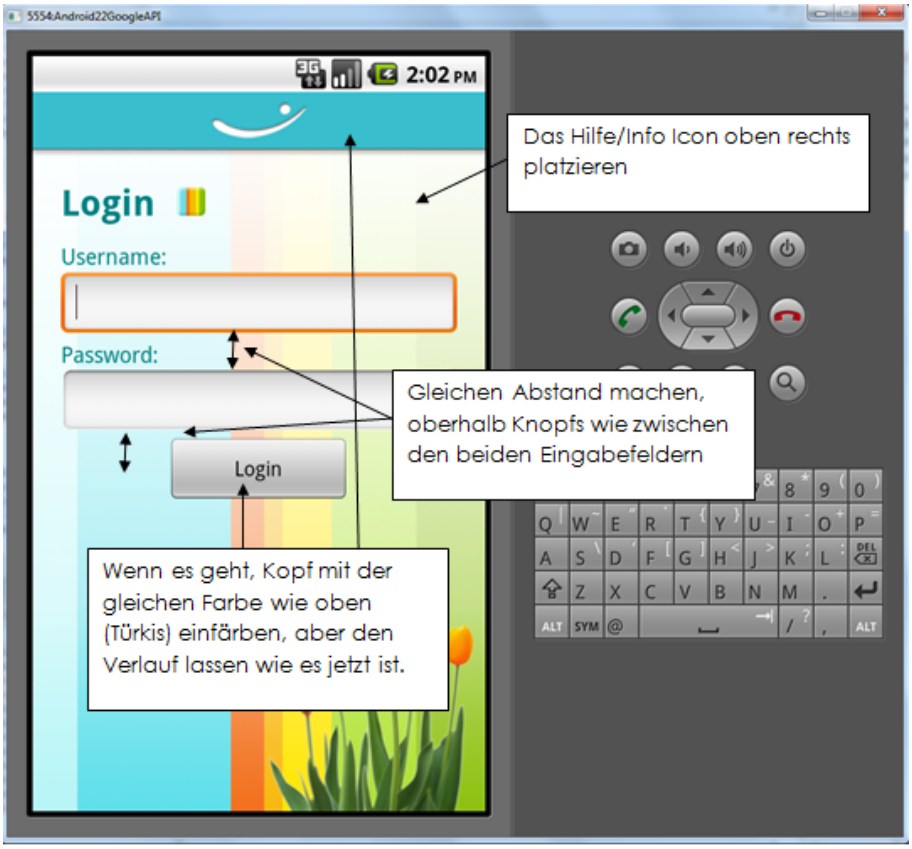

Abbildung 32: GUI-Verbesserungsvorschläge

11.6 GUI / Usability - Workshops

Zwei Mal während der Projektzeit nahm der Auftraggeber an den wöchentlichen Meetings Teil, um sich einen Überblick über die Fortschritte zu verschaffen.

Im ersten Meeting wurden die GUI-Prototypen vorgestellt. Dabei wurden die Android Design-Regeln erklärt, welche sich zum Teil sehr stark vom iPhone unterscheiden. Die dadurch entstandenen Änderungen im Design wurden vom Auftraggeber als ansprechend empfunden.

Beim zweiten Meeting waren nebst dem Projektteam und dem Betreuer zwei Mitarbeiter von Crealogix, sowie der Auftraggeber von CENTRADO dabei. Bei diesem Meeting hatten alle Teilnehmer die Möglichkeit, die App auf einem separaten Gerät zu testen. Dabei wurden die Teilnehmer auf alle Use Cases sowie optionalen Anforderungen hingewiesen, damit alles getestet werden konnte. Auch dieser Workshop verlief gut. Die wenigen Fehler konnten nach dem Meeting verbessert werden. Die App hinterliess einen guten Gesamteindruck.

12 Erreichte Ziele/ Offene Punkte

Alle Use Cases sowie optionale Anforderungen sind in der Anforderungsanalyse im Kapitel 4 dokumentiert. Dieses Kapitel beschreibt, welche Use Cases umgesetzt wurden und welche optionalen Anforderungen zusätzlich implementiert wurden. Ebenfalls wird erwähnt, welche Anforderungen nicht vollständig umgesetzt wurden.

- Bezüglich des Programmcodes wurden die nichtfunktionalen Anforderungen NF12 und NF13, NF21 erfüllt.
- Das externe Design der App entspricht folgenden nichtfunktionalen Anforderungen: NF01, NF02, NF03
- Die nichtfunktionale Anforderung NF11 (Qualität) wurde während des gesamten Projekts beachtet.
- Die nichtfunktionale Anforderung NF19 zur Mindestversion von Android wurde ebenfalls eingehalten.
- Die nichtfunktionale Anforderung NF22 (Internationalisierung) wurde umgesetzt.

12.1 Use Cases

12.1.1 Login

Wurde gemäss UC01 realisiert. Implementationsdetails sind im Kapitel 7.4 Login notiert.
Erfüllte nichtfunktionale Anforderungen: NF05,NF 09, NF18

12.1.2 Impulse anzeigen

Wurde gemäss UC02 realisiert.
Erfüllte nichtfunktionale Anforderungen: NF07, NF10, NF17

12.1.3 Push-Notification empfangen

Wurde gemäss UC03 realisiert. Der Push-Mechanismus wurde mithilfe von C2DM (Siehe Kapitel 7.1 C2DM) implementiert.

12.1.4 Logout

Wurde gemäss UC04 realisiert.
Erfüllte nichtfunktionale Anforderungen: NF08

12.2 Optionale Anforderungen (Priorität 1)

Bei allen optionalen Anforderungen wurde darauf geachtet, dass alle damit verbundenen Änderungen, den definierten nichtfunktionalen Anforderungen entsprechen.

12.2.1 Favoriten / Merkliste

Die Möglichkeit, Impulse als Favoriten zu markieren wurde umgesetzt.

12.2.1.1 Suche

Die Suchfunktion wurde implementiert. Implementationsdetails sind in Kapitel 7.2.2 Suche zu finden.

12.2.2 Gelesen / Ungelesen

Der Gelesen / Ungelesen Status wurde bereits beim iPhone umgesetzt. Aus diesem Grund war es wichtig, diese optionale Anforderung ebenfalls zu implementieren. Da die Symbole bereits beim iPhone App erstellt wurden, konnte diese verwendet werden. Der Standort des Gelesen / Ungelesen Symbols wurde, wie üblich, am linken Bildschirmrand platziert.

12.2.3 Liste sortieren

Dieser Vorschlag kam vom Projektteam. Während der Umsetzung der anderen optionalen Anforderungen wurde jedoch klar, dass die Benutzer bereits ausreichende Möglichkeiten haben, um den gewünschten Impuls zu finden. Eine zusätzliche Auswahl für die Sortierung wäre ein Overkill gewesen. Diese Anforderung wurde somit nur teilweise umgesetzt. Die Liste ist immer nach Datum sortiert, jedoch kann diese Sortierung nicht geändert werden.

12.2.4 Filterung

Die Filterung wurde implementiert. Nähere Informationen dazu sind im Kapitel 0 SQLite notiert.

12.2.5 Horizontale Ansicht der App

Die horizontale Ansicht der App wurde nur bei der Galerie umgesetzt. Dies bedeutet, dass bei einem Bild oder Bildfolge-Impuls das Gerät gekippt werden kann. Bei den anderen Activities ist die horizontale Ansicht gesperrt.
Es wurde jedoch ausprobiert, wie sich die App im horizontalen Modus verhalten würde. Dabei wurde festgestellt, dass nur kleine Anpassungen nötig wären, um alles korrekt in horizontaler Ansicht anzuzeigen. Aus Zeitgründen konnten jedoch keine Tests mehr durchgeführt werden, was das Projektteam dazu bewog, diese optionalen Anforderungen nicht vollständig zu implementieren. (NF11)

12.2.6 Jingle

Beim Empfang eines neuen Impulses wird der Centrado Jingle abgespielt.

12.2.7 Hilfe Icon (auf Login Screen)

Auf dem Login-Screen wurde ein Info Icon eingefügt, welches mit der Help Webseite für Android Geräte von CENTRADO verlinkt ist. Ebenfalls ist diese Hilfe, vom Menü aus, auf der LoginActivity auffindbar.

12.3 Optionale Anforderungen (Priorität 2 und 3)

Die optionalen Anforderungen mit Priorität 2 und 3 wurden nicht umgesetzt. Die Gründe dafür sind im Kapitel 4.4.2 zu finden.

13 Verbesserungsvorschläge

13.1 Einleitung

Auf Wunsch des Auftraggebers wurden Verbesserungsvorschläge zur bestehenden Infrastruktur notiert. Die Verbesserungsvorschläge sind dem Projektteam während der Arbeit aufgefallen und sind anschliessend in den Meetings angesprochen worden.

13.2 Vorschläge

13.2.1 REST-Schnittstelle

Die schmale REST-Schnittstelle, welche es nur erlaubt eine Liste aller Impulse herunterzuladen, birgt das grösste Verbesserungspotenzial. Da die Push-Nachrichten eine maximale Länge (Content-Length) haben, können auch keine Impulse per Push versendet werden. Die Konsequenz ist, dass der Client über den Push-Mechanismus benachrichtigt wird, dass neue Inhalte auf dem Server verfügbar sind. Der Client muss anschliessend die ganze Liste herunterladen, diese mit seinem lokalen Abbild in der Datenbank vergleichen und den aktuellsten Impuls anzeigen.

Besser wäre es, wenn per Push eine Impuls-ID verschickt würde und der Client diesen einzelnen Impuls über diese ID herunterladen könnte.

13.2.2 Alternatives Konzept (HTML5)

Im Verlauf der Arbeit wurde über ein alternatives Konzept mit HTML5 diskutiert. Dieser Ansatz würde eine Umstellung des Daten-Konzepts beinhalten. Die Impulse würden mittels HTML formatiert und dargestellt. Die Client-App würde sich auf eine Art Inbox und Browser reduzieren, da die multimedialen Inhalte (Bilder, Videos usw.) direkt im HTML Code eingebunden werden könnten.
Diese HTML-Impulse könnten von der CENTRADO-Server-Infrastruktur zur Verfügung gestellt werden, die Clients bräuchten lediglich noch URLs zu den entsprechenden Impuls-Seiten zu speichern, alternativ könnte der HTML Code auch heruntergeladen werden, damit die Inhalte auch offline verfügbar sind.

Dieser Ansatz bringt viele Vorteile und ermöglicht eine schönere und bessere Darstellung der Impulse.

14 Persönliche Berichte

14.1 Mathias Fasser

Die Möglichkeit eine Arbeit auf Android zu machen war für mich eine grosse Chance. Denn die Entwicklung auf mobilen Geräten hat mich schon seit längerem interessiert, speziell Android.

Die von CENTRADO entwickelte Work-Life-Balance fand ich spannend. Dass die App in naher Zukunft auf dem Android Markt verfügbar sein wird gab mir einen zusätzlichen Anstoss, die Arbeit möglichst gut zu machen.

Die Arbeit war sehr vielfältig und forderte in mehreren Bereichen grossen Aufwand in die Einarbeitung der verschiedenen Themengebiete.
Zuerst musste ich mich mit den grundlegenden Komponenten der Android Programmierung beschäftigen, da sich die Entwicklung auf Smartphones doch sehr von der Desktop-Entwicklung unterscheidet.
Anschliessend war es für diese Arbeit essentiell, den Push-Mechanismus von Android zu verstehen und implementieren zu können.
Da die App eine grafisch hochwertige Benutzeroberfläche erforderte, verbrachte ich ebenfalls viel Zeit mit der Gestaltung des GUIs.

Da ich mit Marco Pfiffner bereits einige Projekte gemacht hatte, war die Zusammenarbeit sehr angenehm. Dies führte dazu, dass wir schnell zu Erfolgen kamen und grössere Hürden effizient lösen konnten.
Die Zusammenarbeit mit unserem Betreuer war mir sehr wichtig. Auch hier herrschte ein gutes Verhältnis und ich bin dankbar für diese Unterstützung.

Die Zusammenarbeit mit dem Industriepartner hat die Semesterarbeit von bisherigen Projekten abgehoben. Das wöchentliche Meeting mit Betreuer und Mitarbeitern von Crealogix, sowie teilweise dem Auftraggeber von CENTRADO war jeweils sehr spannend.
Während den Meetings wurde eine Liste mit optionalen Anforderungen erstellt. Die Implementation dieser zusätzlichen Aufgaben bot die Möglichkeit, mich intensiver mit der Materie zu beschäftigen.

Während dieser Arbeit habe ich viel über die Android Programmierung gelernt. Beim wöchentlichen Meeting konnte ich meine Präsentationsfähigkeiten verbessern und vieles über die Zusammenarbeit mit Industriepartnern lernen. Dieses Wissen kann ich in meinen nächsten Projekten gut einsetzen. Ebenso das Spezifizieren und durchführen von Systemtests.

Da wir während der Arbeit wenig dokumentiert haben, musste dies am Schluss aufgeholt werden. In diesem Punkt muss ich mich verbessern, da die Vorteile davon am Ende der Arbeit sehr gross sind.
Ein Software-Projekt mit viel Technologien aber wenig Business-Logik ist kompliziert zu testen, da die Technologien „gemockt“ werden müssen. Während der Implementation wurde dies nicht beachtet. In Zukunft werde ich mir zu Beginn der

Arbeit überlegen, wie der technologie-abhängige Code aussehen muss, damit auch dieser getestet werden kann.

14.2 Marco Pfiffner

Schon im Vorfeld unserer Studienarbeit war klar, dass wir eine Arbeit im Bereich der Mobile-Apps machen wollen. Die Herausforderung eine „marktreife" App auf Android für einen externen Partner zu machen, gab mir noch den letzten Anstoss.

Ich war sehr motiviert das, mir noch relativ unbekannte, Android-Framework genauer kennen zu lernen.
Die Aufbereitung der verschiedenen Android-Konzepte und Technologien war zwar zeitintensiv, jedoch sehr spannend und lehrreich. Die zu Beginn aufgewendete Zeit hat sich jedoch ausgezahlt, so konnten wir alle grösseren Risiken mit unseren Prototypen beseitigen und schon bald erste Resultate an unseren Meetings präsentieren.

Der gute und enge Kontakt zu Projektpartner, Kunden und Dozenten hat diese Arbeit in vielerlei Hinsicht geprägt. In den zahlreichen Meetings wurde viel diskutiert und gemeinsam Entscheide getroffen. Unklarheiten unserer Seite konnten stets mit den diversen Ansprechpartnern geklärt werden, sodass die Arbeit nie wirklich ins Stocken geriet.
Auch die Zusammenarbeit mit meinem Team-Kollegen verlief stets problemlos. Mathias Fasser und ich kennen uns schon seit der Berufsschule und sind ein gut eingespieltes Team. Nicht zuletzt dank unserer gemeinsamen Erfahrung konnten auch die etwas grösseren Probleme gemeinsam gelöst werden.

Ich schaue auf 14 lehrreiche Wochen und eine sehr spannende, wenn auch ein wenig stressige Zeit zurück. Mit dem Resultat unserer Arbeit bin ich sehr zufrieden. Zusammen mit Crealogix konnten wir, meines Erachtens, eine ansehnliche App „erschaffen". Neben der geplanten Grundfunktionalität konnten wir noch einige Features implementieren, die das Arbeiten mit der App vereinfachen bzw. verbessern.

Natürlich verlief nicht immer alles perfekt. Nun gilt es die begangenen Fehler zu analysieren und im Hinblick auf die BA aus ihnen zu lernen. Die Projektplanung, sowie das Erfassen der Arbeitspakete und der aufgewendeten Zeit will ich in der BA bestimmt anders lösen. Dadurch, dass die Arbeitspakete in Unfuddle und die aufgewendete Zeit in Excel (da auf Unfuddle nicht möglich) erfasst wurde, entstand eine Art „Kluft". Dadurch konnte die Zeitauswertung nicht wirklich aussagekräftig erstellt werden.
Ausserdem haben wir es zu Beginn der Arbeit etwas verpasst unsere Technologien „testbar" zu designen. Am Schluss war es fast unmöglich die Kollaboration der verschiedenen Komponenten zu mocken und so die einzelnen Technologien mit Unit-Tests zu überprüfen. Allenfalls könnte beim nächsten Projekt sogar ein Test-Driven-Ansatz verwendet werden.

15 Glossar

Begriff:	Definition:
Android-Manifest	Zentrale Verwaltungs-Datei von Android-Projekten
Application Framework	
Authentication-Token	Authentifiziert den C2DM-Server beim Google-Service
C2DM	Cloud To Device Messaging Framework, Push-Mechanismus von Android
DVM	Dalvik Virtual Machine Virtuelle Umgebung für Android Applikationen
Externes Design	Benutzeroberfläche (GUI) eines Software-Programmes
Garbage Collector	Zuständig für automatische Speicherbereinigung
HTML5	Textbasierte Auszeichnungssprache zur Strukturierung von Webseiten
JSON	JavaScript-Object-Notation, textbasierte Objekt-Austausch-Notation
Kernel	Zentraler Bestandteil eines Betriebssystems (Speicher- und Prozess-Management)
Library	Sammlung von Programmfunktionen
NF	Abkürzung: Nichtfunktionale Anforderung
Open Handset Alliance	Konsortium zur Schaffung offener Standards für Mobilgeräte
REST	Eine in sich abgeschlossene HTTP-Anfrage
Swipe-Gesten	Streichbewegungen auf dem Display
SSL	Secure Sockets Layer Protokoll zur sicheren Kommunikation über das Internet
UC	Abkürzung für Use Case

16 Literaturverzeichnis

Referenz	Quelle
[AndroidDev]	Android Developers http://developer.android.com/index.html
[AndroidGalileo]	Thomas Künneth: Android 3, Apps entwickeln mit dem Android SDK (ISBN: 978-3-8362-1697-5)
[AndroidGrundProg]	Arno Becker, Marcus Pant: Android 2, Grundlagen und Programmierung (ISBN: 978-3-89864-677-2)
[AndroidPIT]	Android-Pit.com http://www.androidpit.com/en/android/market/
[BAiPhone]	Christian Kölla, Andres Eugster: Bachelorarbeit, CENTRADO iPhone App
[GoogleC2DM]	Google C2DM http://code.google.com/intl/de-DE/android/c2dm/
[WikiAndroidDE]	Wikipedia.org http://de.wikipedia.org/wiki/Android_%28Betriebssystem%29
[WikiAndroidEN]	Wikipedia.org http://en.wikipedia.org/wiki/Android_%28operating_system%29

17 Verzeichnisse

17.1 Abbildungen

17.2 Code-Snippets

17.3 Tabellen