HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Centrado iPhone App

Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühjahrssemester 2011

Autoren:	Andres Eugster, Christian Kölla
Betreuer:	Prof. Hans Rudin
Projektpartner:	Centrado GmbH, Crealogix AG
Experte:	Daniel Hiltebrand, Crealogix AG
Gegenleser:	Prof. Beat Stettler

Inhalt

1 Abstract

1.1 Ausgangslage

Die Firma Centrado erstellt zusammen mit Crealogix eine Work Life Balance Plattform. Das Ziel dieser Plattform ist, dass der Benutzer - durch eine ganzheitliche Betrachtung und Förderung - eine ausgeglichene Work Life Balance erreicht. Eine Funktion dieses Systems besteht darin, dem Benutzer regelmässig *Impulse* zuzustellen. Diese bestehen aus Texten, Bildern oder Videos, die Anregungen oder Übungen enthalten, die der Benutzer ausführen kann.

Die Impulse können über verschiedene Kanäle empfangen werden. Das Ziel dieser Bachelorarbeit ist das Erstellen einer iPhone App und einer Server-Schnittstelle, um Impulse auch auf dem iPhone empfangen zu können.

1.2 Vorgehen / Technologie

Zu Beginn der Arbeit wurde eine umfassende technische Analyse in den Bereichen iPhone-App-Programmierung, Push Notifications, sicheres Ablegen von Zugangsdaten, sicherer Login am Server und SSL-Verbindungen durchgeführt. Eine grosse Herausforderung dabei war die Einarbeitung in Objective-C und XCode als Entwicklungsumgebung.

Während der gesamten Arbeit wurde intensiv mit dem Auftraggeber zusammengearbeitet und verschiedene Lösungsvorschläge diskutiert. Bei den wöchentlichen Sitzungen wurde Produktqualität und Usability regelmässig besprochen und falls nötig auf die nächste Sitzung angepasst.

1.3 Ergebnisse

Im Rahmen der Bachelorarbeit wurden die iPhone App und ein Server zum Testen der Schnittstelle implementiert.

Sobald der Server einen neuen Impuls für einen Benutzer erstellt hat, wird der Benutzer mit einer Notification benachrichtigt. Nachdem der Benutzer die Notification bestätigt hat, wird die App geöffnet und der neue Impuls geladen. Der Benutzer kann nun den Text lesen und die Bilder oder das Video betrachten. Anschliessend führt er die vorgeschlagene Übung aus.

Im Weiteren kann die App Impulse und Bilder speichern, sodass diese auch ohne Internetverbindung angezeigt werden können. Die gespeicherten Impulse können mit einer intuitiven Benutzeroberfläche betrachtet und verwaltet werden. Die Bedienung der Centrado iPhone App ist an die Bedienung der Standard iPhone Apps von Apple angelehnt.

Für den Zugriff auf den Server muss der Benutzer seine Anmeldedaten eingeben. Diese werden in der App gespeichert, können aber auf Wunsch des Benutzers auch wieder gelöscht werden. Die Übertragung von benutzerspezifischen Daten zwischen iPhone App und Server ist verschlüsselt.

Hat der Benutzer einmal keine Zeit, einen neuen Impuls zu lesen, so kann er die Benachrichtigung einfach schliessen und den Impuls zu einem späteren Zeitpunkt ausführen.

2 Management Summary

2.1 Einführung

Die Centrado GmbH entwickelt zusammen mit der Crealogix AG eine Work Life Balance Plattform. Diese Plattform soll die Work Life Balance ihrer Benutzer verbessern. Für eine gute Work Life Balance ist es wichtig, dass alle Bereiche im Leben eines Menschen in Ordnung sind, sei es nun die Familie, Arbeit oder .

Centrado betrachtet für die Work Life Balance 12 Bereiche des Lebens. Damit der Benutzer seine schwächeren Bereiche verbessern kann, hat Centrado mehrere Einzelimpulse und auch ganze Serien von Impulsen ausgearbeitet.

Impulse geben Hinweise, wie ein Bereich der Work Life Balance verbessert werden kann. Die Impulse werden nicht zufällig verschickt, sondern basieren auf Werten, die für jeden Benutzer in einem Screening ermittelt wurden. Im Screening beantwortet der Benutzer verschiedene Fragen zu den verschiedenen Lebensbereichen. Die Daten werden ausgewertet und in einer Grafik dargestellt (Abbildung 1). Aufgrund dieser Daten weiss das System nun, dass vor allem in den Bereichen Beweglichkeit, Kreativität und Spiritualität etwas verbessert werden kann.

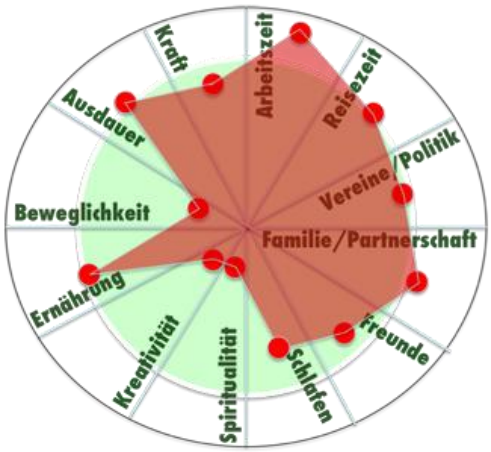

Abbildung 1: Resultat eines Screening (Quelle: centrado.ch)

Die Impulse können dem Benutzer über verschiedene Kanäle zugestellt werden: Web-Seite, E-Mail, MMS oder auch SMS. Der eleganteste Weg, einen Impuls zu erhalten ist auf dem iPhone in einer eigens dafür programmierten Centrado iPhone App.

Sobald der Benutzer nun einen Impuls erhalten hat, kann er ihn lesen und die mitgeschickten Bilder oder das Video betrachten.

2.2 Vorgehen

Damit das iPhone in seiner Funktionalität ausgereizt werden konnte, wurde eine umfangreiche technische Analyse durchgeführt. Diese umfasst unter anderem *Push Notification*, sicheres Ablegen von Zugangsdaten, sicherer Login am Server und SSL-Verbindungen. Auch die Eigenheiten des Programmierens einer iPhone App benötigte einige Einarbeitungszeit. Eine grosse Herausforderung dabei war die Verwendung von *Objective-C* und *XCode* als Entwicklungsumgebung.

Während der gesamte Bachelorarbeit fanden wöchentlich Sitzungen statt. Durch die regelmässigen Sitzungen war die Zusammenarbeit mit dem Auftraggeber sehr intensiv. Die geplanten Arbeiten wurden jeweils mit dem Auftraggeber besprochen und priorisiert. Die ausgearbeiteten Lösungen konnten gleich nach der Umsetzung diskutiert werden und bei Bedarf korrigiert oder ergänzt werden.

2.3 Ergebnisse

Die erstellte App kann sich sehen lassen. Sie stellt dem Benutzer einen grossen Funktionsumfang für das Empfangen, Betrachten und Verwalten von Impulsen zur Verfügung. Die Impulse sind übersichtlich in einer Inbox dargestellt und können angezeigt oder auch gelöscht werden.

Die Benutzeroberfläche ist der des E-Mail Clients auf dem iPhone ähnlich. Somit findet sich der Benutzer auf Anhieb damit zurecht. Crealogix stellte Logos, Designs und Farbkonzept für die App zur Verfügung, sodass die App schon im Rahmen der Bachelorarbeit ihr definitives Erscheinungsbild erhalten hat.

Abbildung 2: Impulse Inbox und Darstellung eines einzelnen Impulses

Zum Testen der App wurde ein Server implementiert. Mit diesem Server konnte die App unabhängig von der Infrastruktur betrieben werden, die von Crealogix parallel zur Bachelorarbeit erstellt wurde. Die App ist in der Lage, sämtliche Impulse von diesem Server zu beziehen und sich zu registrieren. Diese Registrierung ist nötig, damit der App eine *Push Notification* zugestellt werden kann.

Wenn nun der Server einen neuen Impuls hat, so wird der Benutzer mit einer Push Notification benachrichtigt. Diese Notification wird dem Benutzer mit einer Meldung angezeigt. Nach einer Bestätigung wird die App geöffnet und der neue Impuls geladen. Der Benutzer kann nun den Text lesen und die Bilder oder das Video betrachten. Anschliessend führt er die vorgeschlagene Übung aus.

Hat der Benutzer einmal keine Zeit, einen Impuls zu betrachten oder auszuführen, so kann er dies auch zu einem späteren Zeitpunkt nachholen. Die Impulse (Titel und Text) werden sofort persistent abgelegt. Dass die Bilder ebenfalls vorhanden sind, muss der Benutzer sie kurz anzeigen, danach sind die Bilder in einem Cache gespeichert und können auch offline betrachtet werden.

Zum Einloggen an der App muss der Benutzer seine Anmeldedaten eingeben. Diese werden in der App gespeichert, können aber auf Wunsch des Benutzers wieder gelöscht werden.

Die Übertragung von benutzerspezifischen Daten zwischen iPhone App und dem Server ist verschlüsselt und durch ein Zertifikat geschützt.

2.4 Ausblick

Die Videos können beim jetzigen Stand der App noch nicht im Cache abgelegt werden. Das Handling ist um einiges komplexer als bei den Bildern, da der Benutzer erwartet, dass auch erst teilweise geladene Videos abgespielt werden können.

Die Kommunikation mit dem Server funktioniert bestens. Es ist jedoch darauf zu achten, dass der Server z.B. nur die letzten 10 Impulse schickt. Werden mehr Impulse geschickt, so entstehen bei einer schlechten Internetverbindung ziemlich lange Ladezeiten.

Da der Benutzer eventuell einmal keine Zeit hat, einen Impuls zu betrachten, könnte die App um eine Erinnerungs-Funktion ergänzt werden. Der Benutzer könnte dann in der App eine Zeit einstellen, zu der er nochmals an den Impuls erinnert werden möchte.

3 Einführung

Das Ziel der Bachelorarbeit ist die Entwicklung einer iPhone Applikation für die Centrado Work Life Balance Plattform. Da manche Personen mit dem Begriff Work Life Balance noch nicht vertraut sind, wird das Thema erklärt und zusammen mit Centrado noch kurz vorgestellt.

3.1 Einführung zu Centrado

Der folgende Text stammt aus der Broschüre „CENTRADO – Work Life Balance Plattform“ (Centrado.ch)

> *Viele Menschen schöpfen – verursacht durch eine nicht ausgeglichene Work Life Balance - ihr volles Leistungspotential nicht aus.*
> *Nicht erst wenn Krankheitssymptome gegen aussen sichtbar werden, beginnt die Beeinträchtigung: Kreativität, Intuition, Energie, Belastbarkeit und Konfliktfähigkeit nehmen ab, und damit wird die Leistungsfähigkeit zunehmend eingeschränkt.*
> *Mit CENTRADO wird es erstmals möglich, die Work Life Balance durch regelmässige, individuelle Impulse nachhaltig zu fördern und zu festigen.*
> *Zusätzlich unterstützt die Mandantenfähigkeit in CENTRADO die Pflege und Lebendigkeit der eigenen Firmenkultur. Im Zuge der demografischen Entwicklung werden die „Soft Factors“ für die Gewinnung neuer resp. die Bindung qualifizierter Arbeitskräfte entscheidend sein.*

Verwendung

Durch ein feingliederiges **Screening** wird zu Beginn ein Bild der aktuellen Work Life Balance erstellt. Dazu muss der Benutzer verschiedene Fragen beantworten. Auf Grund der Ergebnisse aus dem Screening werden **Impulse** zusammengestellt, die dem Benutzer über einen längeren Zeitraum zugestellt werden.

Im persönlichen **Cockpit** kann der Benutzer unter anderem Einstellungen zu seinem Account vornehmen oder die Ergebnisse seiner Screenings und die Veränderungen in seiner Work Life Balance betrachten.

3.2 Motivation und Vision

Das Ziel der Bachelorarbeit ist die Entwicklung einer iOS Applikation (App), die von den Endbenutzern als Client für die Centrado-Plattform benutzt werden kann. Zudem soll eine REST-Schnittstelle entwickelt werden, über die der Server angesprochen werden kann.

Die App soll graphisch hochwertig sein und die speziellen Eigenschaften und Einschränkungen des iPhones berücksichtigen. Crealogix liefert Designvorschläge und Icons als Grundlage für das Design.

Es gilt dabei Qualität vor Quantität. Es sollen die Funktionen bis zur Produktionsreife entwickelt werden, bevor weitere Funktionen hinzugefügt werden. Es wird erwartet, dass das Anzeigen von Impulsen inklusive lokale Speicherung im Rahmen der Bachelorarbeit realisiert werden kann.

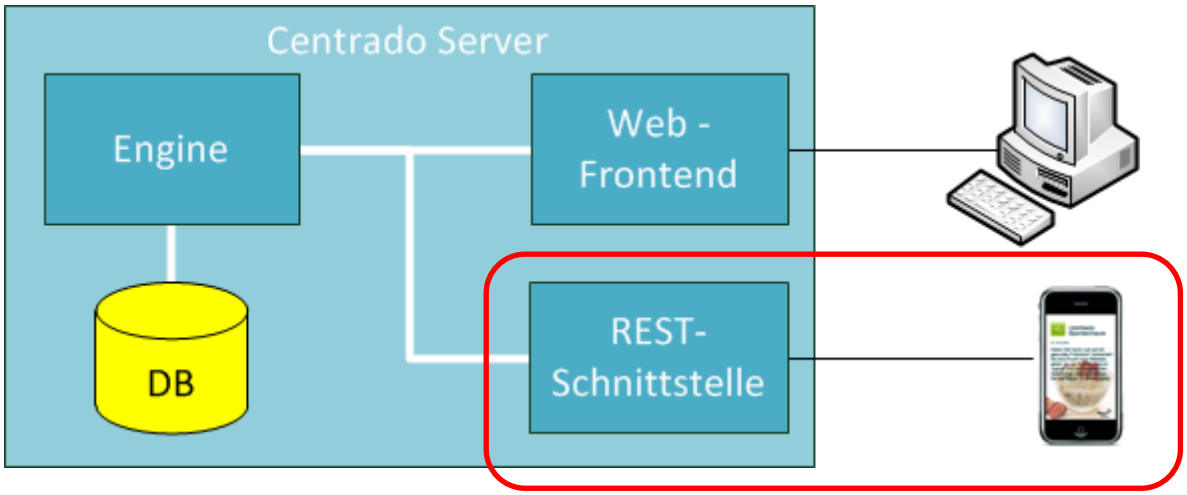

Abbildung 3: Fokus der Bachelorarbeit

Komponente	Beschreibung
Engine	Die Engine hat Zugriff auf die DB. Auf Grund der erfassten Benutzerdaten entscheidet die Engine, welcher Benutzer wann, welche Impulse erhält.
Web-Frontend	Über das Web-Frontend ist der Benutzer in der Lage, seine Angaben, Impulse und Einstellungen einzusehen und zu verändern.
REST-Schnittstelle	Damit die App mit dem Server kommunizieren kann, verbindet sie sich mit der *REST-Schnittstelle*.
DB	Die Datenbank enthält alle Benutzerangaben und Impulse, die für den Betrieb der Centrado-Plattform benötigt werden.

3.3 Vorgehen und Funktionalität

Die Arbeit ist in Iterationen aufgeteilt, die jeweils mit einem Meilenstein abgeschlossen werden. Da sämtliche mögliche Features für eine Bachelorarbeit zu viel geworden wären, wurde eine Aufteilung in *Funktionssätze* vorgenommen:

Die App soll folgende *Funktionssätze* mit der untenstehenden Priorisierung implementieren.

- FU01: Anzeigen von Impulsen inklusive deren lokale Speicherung
- FU02: Cockpit
- FU03: Screening
- FU04: B2C Abo über In-App-Purchase

3.4 Abgrenzung

Die App soll auf Geräten mit iOS 4.0 oder höher lauffähig sein. Als minimale Hardwareanforderung gilt das iPhone 3GS. iPads werden nicht speziell unterstützt, das heisst, es wird keine iPad-optimierte Benutzeroberfläche entwickelt.

Die Engine und das Web-Frontend (siehe Abbildung 3: Fokus der Bachelorarbeit) werden parallel zur Bachelorarbeit durch Crealogix realisiert. Die App aus der Bachelorarbeit und die Infrastruktur von Crealogix werden während der Bachelorarbeit nicht integriert.

Das Deployment der App über den App Store, sowie die Weiterentwicklung und Pflege werden durch Crealogix vorgenommen.

Die REST-Schnittstelle wird vor allem zum Testen der App implementiert. Es soll aber darauf geachtet werden, dass der Code auch produktiv verwendet werden kann.

Die genaue Aufgabenstellung der Arbeit kann dem Dokument [Aufgabenstellung] entnommen werden. Die wichtigsten Punkte sind jedoch hier aufgelistet.

Umfang

- Erstellung einer iPhone App mit grafisch hochwertigem Design
- Erstellung eines geeigneten Testkonzepts (Server inklusive Schnittstelle)
- Erstellung eines Projektplans – das Vorgehen ist iterativ
- Vorschläge für die nächste Iteration werden von den Studierenden ausgearbeitet.

Organisation

- Wöchentliche Sitzung mit Betreuern der HSR und einem Vertreter des Auftraggebers
- Die Dokumentation ist vollständig auf CD/DVD in 4 Exemplaren abzugeben. Auf Wunsch ist für den Auftraggeber eine gedruckte Version zu erstellen.

Referenzen

- Centrado.ch
- Broschüre „CENTRADO - Work Life Balance Plattform“ (centrado.ch)
- Sitzungsprotokolle

4 Anforderungsanalyse

4.1 Personas & Szenarios

Bei dieser Arbeit wurden Designvorschläge der GUIs von Crealogix zur Verfügung gestellt. Daher konnte auf Wunsch des Auftraggebers [ProtoW2] auf eine detaillierte Analyse der Benutzer (Personas) und den Gebrauch der App (Szenarios) verzichtet werden.

Es wurden zwei Interviews mit zukünftigen Benutzern der App durchgeführt. Folgende Punkte haben sich dabei herausgestellt:

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.

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.

4.2 Use Cases

Die folgenden Use Cases beziehen sich auf den Funktionssatz „FU01: Anzeigen von Impulsen inklusive deren lokale Speicherung“.

4.2.1 Übersicht

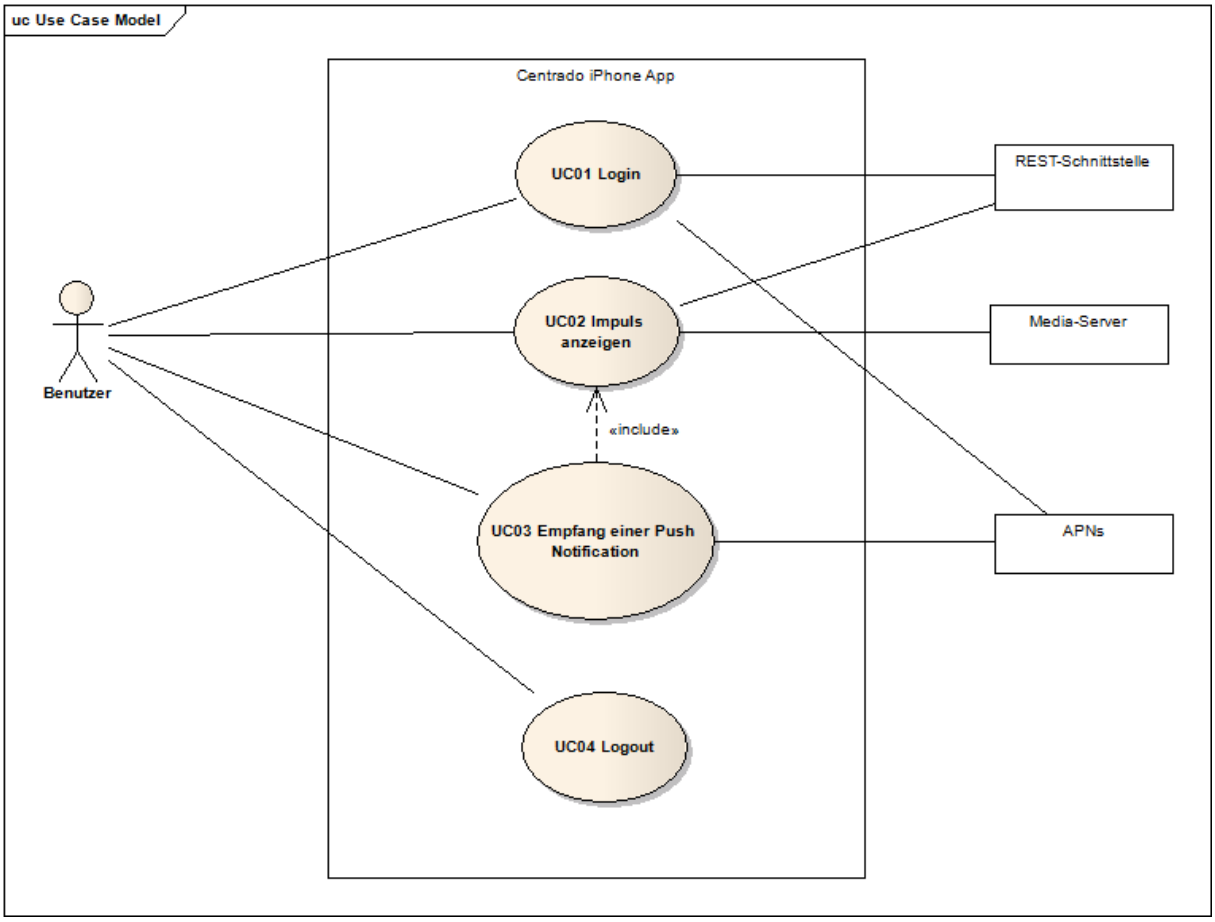

Abbildung 4: UseCase-Diagramm zu Funktionssatz „Anzeigen von Impulsen inklusive deren lokale Speicherung“

4.2.2 Aktoren

Aktor	Beschreibung
Benutzer	Benutzer der Centrado iPhone App
REST-Schnittstelle	Centrado Serverkomponente, die mit der iPhone App kommuniziert
Media-Server	Server mit Medieninhalten wie Bilder und Videos für die Impulse
APNs	Apples Infrastruktur für die Apple Push Notification Services

4.2.3 Use Case Beschreibungen

Use Case ID	UC01
Use Case Name	Login
Umfang	iPhone App, REST-Schnittstelle
Ebene	Benutzer
Primäraktor	Benutzer
Überblick	Der Benutzer startet die App und loggt sich mit seinen Login-Daten ein.
Stakeholder und Interessen	Der Benutzer will Zugriff auf seinen Account.

Vorbedingungen	Benutzer hat einen Account mit Login-Daten. Die Login-Daten sind in der App noch nicht erfasst.
Nachbedingungen	Die Login-Daten sind gültig und in der App verfügbar.
Auslöser	Start der App
Standardablauf	1. App fordert beim APNs einen Device Token an. 2. App überprüft, ob Login-Daten gespeichert sind. 3. App fordert Benutzer zur Eingabe von Login-Daten auf. 4. Benutzer gibt Login-Daten ein. 5. App überprüft ob Login-Daten gültig sind. 6. App teilt den Device Token der REST-Schnittstelle mit. 7. App zeigt die Impuls-Inbox an.
Bemerkungen	• Die Situation, dass ein Benutzer noch keine Login-Daten hat, wird nicht berücksichtigt.
Erweiterungen	

1a	**Apple nicht erreichbar**
Beschreibung	Die App kann sich nicht beim APNs registrieren, da der Server nicht erreichbar ist.
Bemerkung	Nach einem erfolgreichen Login, teilt das Gerät den Device Token nicht mit. Beim nächsten Öffnen der App wird wieder ein Device Token angefordert und verschickt.
Fortsetzung	Hauptablauf weiter bei Schritt 2

5a	**Login-Daten falsch**
Beschreibung	Der Benutzer hat den Benutzernamen oder das Passwort falsch eingegeben.
Ablauf	1. App zeigt eine Fehlermeldung an.
Fortsetzung	Hauptablauf zurück zu Schritt 3

5b	**Login-Daten ungültig**
Beschreibung	Die Login-Daten sind korrekt, aber nicht mehr gültig.
Ablauf	1. App zeigt Fehlermeldung an, mit Hinweis, warum die Daten ungültig sind und wie das Problem behoben werden kann.
Fortsetzung	Hauptablauf zurück zu Schritt 3

Use Case ID	**UC02**
Use Case Name	Impuls anzeigen
Umfang	iPhone App, REST-Schnittstelle
Ebene	Technisch
Primäraktor	Benutzer
Überblick	App stellt einen Impuls dar
Stakeholder und Interessen	Der Benutzer möchte den aktuellen Impuls betrachten.
Vorbedingungen	App hat gültige Login-Daten des Benutzers.
Nachbedingungen	Benutzer kann den Impuls betrachten.
Auslöser	Tap auf Impuls oder Eingang einer Push Notification.

Standardablauf	1. App stellt Verbindung zur REST-Schnittstelle her. 2. App fordert aktuellen Impuls vom Server an(Benutzerdaten werden mitgeliefert). 3. REST-Schnittstelle liefert Impuls-Daten. 4. App lädt benötigte Medien vom Medien-Server und zeigt Impuls an.
Spezielle Anforderungen	• Die Verbindung ist verschlüsselt.
Bemerkungen	Der Impuls gilt im Centrado System als konsumiert, wenn die Impuls-Daten bei der REST-Schnittstelle abgeholt wurden.
Erweiterungen	

2a	**REST-Schnittstelle nicht erreichbar**
Beschreibung	Die App kann keine Verbindung zur REST-Schnittstelle herstellen.
Ablauf	1. App zeigt Fehlermeldung an.
Fortsetzung	Abbruch

4a	**Medien-Server nicht erreichbar**
Beschreibung	Die App kann keine Verbindung zum Medien-Server herstellen.
Ablauf	1. App zeigt eine Fehlermeldung an. 2. App zeigt statt der Medien einen Platzhalter und zeigt nur die Beschreibung aus den Impuls-Daten an.
Fortsetzung	Abbruch

Use Case ID	**UC03**
Use Case Name	Empfang einer Push Notification
Umfang	iPhone App
Ebene	Benutzer
Primäraktor	Benutzer
Überblick	Die App empfängt eine Push Notification über APNs. Wenn der Benutzer dies wünscht, wird diese gleich dargestellt.
Stakeholder und Interessen	Der Benutzer möchte wissen, wenn ein neuer Impuls verfügbar ist.
Vorbedingungen	App hat Login-Daten, Benutzer benutzt die App in dem Moment nicht. (iPhone gelockt oder andere App aktiv).
Nachbedingungen	Benutzer ist darüber informiert, dass ein neuer Impuls verfügbar ist.
Auslöser	Eingang der Notification im iPhone
Standardablauf	1. iPhone zeigt Notification an. 2. Benutzer wählt den View-Button zur Darstellung des Impulses aus. 3. Aktuellen Impuls darstellen UC02
Erweiterungen	

2a	**Benutzer wählt Close**
Beschreibung	Der Benutzer nimmt die Notification zur Kenntnis, will den Impuls aber in dem Moment nicht betrachten.
Ablauf	1. Benutzer klickt auf Close.
Spezielles	Die App erfährt bei diesem Ablauf nie, dass eine Notification eingegangen ist.
Fortsetzung	Abbruch

Use Case ID	UC04
Use Case Name	Logout
Umfang	iPhone App
Ebene	Benutzer
Primäraktor	Benutzer
Überblick	Der Benutzer entfernt die Login-Daten aus der App.
Stakeholder und Interessen	Der Benutzer möchte, dass keine unberechtigte Person auf die Daten der App zugreifen kann.
Vorbedingungen	Der Benutzer ist eingeloggt.
Nachbedingungen	Kein Zugriff auf persönliche Daten des Benutzers mehr möglich.
Auslöser	Benutzer wählt Logout-Funktion.
Standardablauf	1. Benutzer wählt Logout-Funktion. 2. App entfernt Login-Daten.
Bemerkungen	Die Logout-Funktion kann in der App selbst oder auch in den Settings realisiert werden.

4.3 Nichtfunktionale Anforderungen

Benutzbarkeit

- NF01: Die iPhone App soll in der Bedienung intuitiv sein. Es darf keine Schulung erforderlich sein. Wenn nötig, werden Hinweise und Kontexthilfe in der App eingebaut.
- NF02: Die App soll die Richtlinien in Apples iOS Human Interface Guidelines beachten.

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.

Sicherheit

- NF05: Alle Zugriffe auf persönliche Daten des Benutzers sollen nur mit Login möglich sein.
- NF06: Übertragungen zwischen Server und iPhone 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 iPhone verloren oder gestohlen wird. Impulse sind nicht speziell gesichert, es sind keine „persönlichen Daten“.

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.

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.

Konformität

- NF15: Die App muss die App Store Review Guidelines einhalten. (https://developer.apple.com/appstore/resources/approval/guidelines.html)

Portabilität

- NF16: Die App soll für das iPhone erstellt werden. Eine iPad-Version ist nicht gefordert.

- NF17: Für den Austausch zwischen iPhone und Serverkomponenten soll JSON verwendet werden.
- NF18: Für die Anmeldung am Server soll http Basic Authentication[1] (mit SSL) verwendet werden.

Umgebung (Vorgaben)

- NF19: Die App soll auf iOS-Geräten mit iOS 4.0 oder höher laufen.
- NF20: Als minimale Geräteanforderung gilt das iPhone 3GS
- NF21: Alle Serverkomponenten sollen in Java 1.6 geschrieben werden.

[1] Siehe 6.1.11 Basic Authentication

5 Domainanalyse

In diesem Kapitel wird das Domain Modell für den Funktionssatz FU01 mit seinen Konzepten vorgestellt. Im Weiteren werden die einzelnen Konzepte mit ihren Beziehungen und Attributen beschrieben.

5.1 Domain Modell (Konzeptionelles Modell)

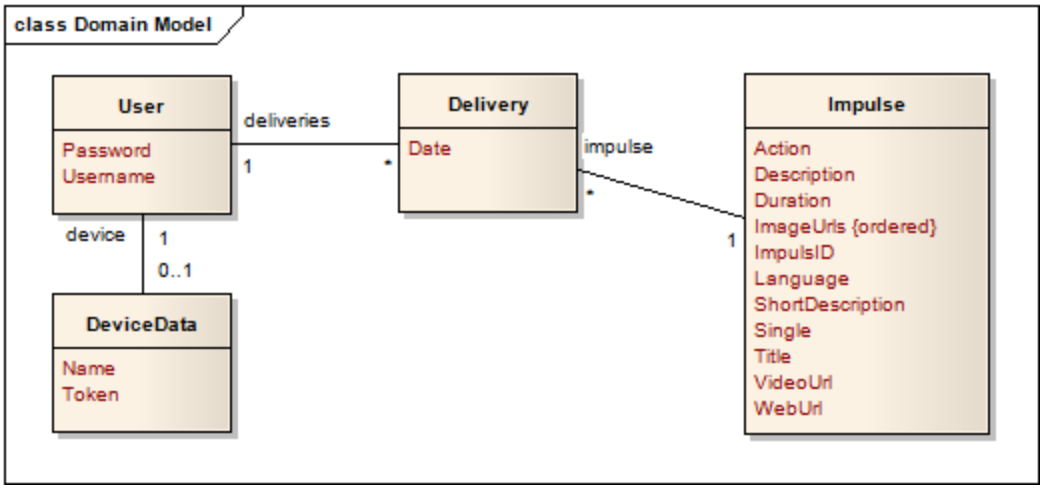

Abbildung 5: Domain Modell

5.2 Beschreibung der Konzepte

5.2.1 User

Beschreibung	Ein User repräsentiert den iPhone- bzw. Centrado-Benutzer	
Attribute	**Password**	Passwort des Benutzers für den Login bei Centrado
	Username	Benutzername für den Login bei Centrado
Beziehungen	Einem User können keine oder beliebig viele Deliveries zugeordnet sein. Einem User kann ein oder kein DeviceData zugeordnet sein.	

5.2.2 Delivery

Beschreibung	Eine Delivery beschreibt, welcher Impuls dem User wann zugestellt wurde.	
Attribute	**Date**	Versendezeitpunkt des Impuls‘
Beziehungen	Eine Delivery ist immer genau einem User zugeordnet Eine Delivery bezieht sich immer auf genau einen Impuls.	

5.2.3 Impulse

Beschreibung	Ein Impulse ist die Definition eines Einzel- oder Programmimpulses.	
Attribute	**ImpulseID**	Merkmal, mit dem der Impuls eindeutig identifiziert werden kann.
	Language	Bezeichnet die Sprache des Textes der Description und ShortDescription.
	Title	Titel des Impulses
	Duration	Für den Impuls benötigte Ausführungszeit.
	Action	Gibt über die Art des Impuls‘ Auskunft: lesen oder ausführen.
	Single	Sagt aus, ob der Impuls ein Einzelimpuls ist oder Teil einer Serie.
	ShortDescription	Kurzbeschreibung des Impulses. Ca. 1 Satz.
	Description	Ausführliche Anweisung an den Benutzer, was bei diesem Impuls zu tun ist.
	VideoUrl	Kann eine URL zu einer Video-Datei enthalten, die zum Impuls abgespielt werden kann.
	WebUrl	Eine URL zu einer Webseite, die weiterführende Informationen zum Impuls enthält.
	ImageUrls	Eine Menge von URLs zu Bildern, die zu diesem Impuls angezeigt werden können.
Beziehungen	Ein Impulse ist keiner oder beliebig vielen Deliveries zugeordnet.	

5.2.4 DeviceData

Beschreibung	*DeviceData* beschreibt ein iPhone, zu welchem Push Notifications gesendet werden sollen.	
Attribute	**Token**	*Device Token* des iPhones.
	Name	Gerätename des iPhones.
Beziehungen	Ein *DeviceData* ist genau einem Benutzer zugeordnet.	

6 Technischer Bericht

Dieses Kapitel gibt Auskunft über die verwendeten Technologien in der Centrado App. Im Abschnitt 6.1 Technologien und Anwendungen wird jeweils untersucht, ob eine Technologie für die App verwendet werden kann. Im Kapitel 6.2 Lösungsansätze und Ergebnisse wurden die zuvor untersuchten Technologien in einem Proof of Concept in ihrer Funktion geprüft.

6.1 Technologien und Anwendungen

6.1.1 iPhone Applikationen entwickeln

Für die Entwicklung von iPhone Applikationen werden zwar die gleichen Werkzeuge, die gleiche Sprache und mehrheitlich dieselben API verwendet, wie bei Desktop Applikationen. Es gibt jedoch einige wichtige Punkte, die bei der Entwicklung beachtet werden müssen.

Multitouch

Die Multitouch-Oberfläche wird von Entwicklern fälschlicherweise oft mit der Bedienung einer Eintasten-Maus gleichgesetzt. Multitouch bietet aber die Möglichkeit, neben dem Antippen von Elementen auf der Anzeige auch Gesten mit mehreren Fingern gleichzeitig zu machen. Eine der berühmtesten Gesten ist zum Beispiel, zwei Finger auf der Anzeige zusammen oder auseinander zu bewegen. Dies wird meistens dazu verwendet, um in das angezeigte Element hinein oder heraus zu zoomen.

Neben dem Vorteil, dass die Touch-Eingabe für den Benutzer sehr intuitiv ist, hat diese Form der Eingabe gegenüber der Maus einige Einschränkungen. Mit einer Maus sieht der Benutzer immer einen Cursor, den er sehr leicht pixelgenau platzieren kann. Ein Bedienungselement kann darauf reagieren, dass sich der Mauscursor über ihm befindet, in seinen Bereich kommt oder diesen verlässt. Bei der Touch-Eingabe gibt es nur die Zustände berühren oder nicht. Die Genauigkeit einer Berührung liegt bei ca. 0.6cm. Die Genauigkeit ist nicht von der Pixeldichte der Anzeige abhängig, sondern von der Grösse des Fingers. Bei der Touch-Eingabe muss ebenfalls beachtet werden, dass während der Eingabe mit den Fingern ein Teil der Anzeige verdeckt wird.

Auch wenn in der Entwicklungsumgebung ein Simulator zur Verfügung steht, um ein iPhone auf dem Mac zu simulieren, ist das regelmässige Testen der Applikation auf einem physischen Gerät ein essentieller Teil der Entwicklung.

Anzeige Grösse

Das iPhone 4 hat mit 3.5 Inch (8.89cm) Bilddiagonale und 960 x 640 Pixel (Verhältnis 3:2) - verglichen mit früheren Smartphones - eine relativ grosse Anzeige. Verglichen mit einem PC ist diese jedoch klein. Auf einem PC können gleichzeitig mehrere unterschiedliche Informationen in verschiedenen Fenstern dargestellt werden. Auf dem iPhone kann nur immer eine Information dargestellt werden. Der Entwickler muss somit entscheiden, welche Informationen wichtig sind und wie diese möglichst effizient dargestellt werden können.

Portabel und sofort einsatzbereit

Im Gegensatz zu einem PC trägt der Benutzer das iPhone fast immer mit sich herum. Auch verglichen mit einem Laptop oder Net-Book ist das iPhone viel näher beim Benutzer. Auch ist das iPhone komfortabel im Stehen benutzbar, ohne dass eine Ablagefläche benutzt werden muss. Diese Eigenschaften haben einen sehr grossen Einfluss auf das Benutzungsverhalten und die Erwartungen, die der Benutzer an das Gerät stellt. Die typische Benutzungsdauer liegt zwischen 5 und 60 Sekunden ([iAPG] Seite 13). Innerhalb dieser Zeit will der Benutzer seine Aufgabe erledigt haben. Dies setzt voraus,

dass die Applikation keine oder nur kurze Wartezeiten hat. Der Benutzer erwartet auch, dass er die gewünschte Aktion ausführen kann, ohne lange durch die Applikation navigieren zu müssen. Im Gegensatz dazu werden typische Desktop Applikationen mehrere Minuten bis mehrere Stunden an einem festen Arbeitsplatz benutzt.

Eingabemöglichkeiten
Das iPhone besitzt keine physische Tastatur. Die Texteingabe erfolgt über eine Bildschirmtastatur, die bei Bedarf eingeblendet wird. Auch wenn diese Bildschirmtastatur verschiedene Hilfen bietet, um Text komfortabler einzugeben, ist die Eingabe viel umständlicher als mit einer PC-Tastatur. Entwickler sollten wenn immer möglich auf zwingende Texteingaben verzichten und stattdessen kurze Auswahllisten mit Vorschlägen anbieten.

Neben der Eingabe über die Bildschirmtastatur können auch weitere Eingabemöglichkeiten genutzt werden. Das iPhone besitzt ein Mikrophon, Sensoren um Lage und Beschleunigung zu messen, verschiedene Mechanismen um die geographische Position des Gerätes zu bestimmen, einen Kompass und eine Kamera. Mit all diesen Eingabemöglichkeiten stehen dem Entwickler eine Vielzahl von Möglichkeiten zur Verfügung, um dem Benutzer vom Eintippen von Informationen zu befreien.

Netzwerkverbindungen
Mittlerweile ist es für PC-Benutzer selbstverständlich, dass jederzeit eine Internetverbindung zur Verfügung steht, über die auch das Streamen von Videos möglich ist. Dies trifft für das iPhone jedoch nur beschränkt zu. Das iPhone besitzt zwar ein WLAN Modul für 802.11b/g/n-Verbindungen, ein Bluetooth-Modul sowie die Möglichkeit, Datenverbindungen über 3G- und GSM-Netze herzustellen. Da das iPhone häufig unterwegs genutzt wird, stehen aber nicht immer die besten, manchmal auch gar keine Datenverbindungen zur Verfügung.

Netz-Typ	Eigenschaften
WLAN	Mit WLAN-Verbindungen sind Internetverbindungen möglich, wie sie der Benutzer vom PC kennt. Diese Verbindungen haben normalerweise Ping-Zeiten im Bereich von 2 bis 50ms und einen Datendurchsatz von 1MBit/s bis 100MBit/s. WLANs haben eine Reichweite von wenigen 10 Metern. Abgesehen von Public Hot Spots hat der Benutzer normalerweise nur Zuhause und eventuell am Arbeitsplatz WLAN-Empfang.
3G	Mit 3G-Verbindungen können unterwegs schnelle Datenverbindungen hergestellt werden. Die Qualität der Verbindungen kann jedoch, abhängig vom aktuellen Aufenthaltsort, sehr stark schwanken. Ping-Zeiten von 90 bis 500ms sind üblich. Der Datendurchsatz liegt im Bereich von ca. 100kBit/s bis 5Mbit/s. In Städten kann allgemein mit guten 3G-Verbindungen gerechnet werden. Auf dem Land gibt es viele Regionen, in denen keine 3G-Verbindungen hergestellt werden können.
GSM	Mit GSM können meistens noch Verbindungen hergestellt werden, wenn kein 3G-Netz zur Verfügung steht. Die Verbindungen über GSM sind verglichen mit den andern Verbindungen sehr langsam. Es muss mit Ping-Zeiten von bis zu 2000ms gerechnet werden und der Datendurchsatz ist selten grösser als 40kBit/s.

Die Funktionsfähigkeit einer iPhone Applikation sollte nicht davon abhängig sein, dass eine Internetverbindung hergestellt werden kann. Der Entwickler muss davon ausgehen, dass eine Verbindung jederzeit unterbrochen werden kann.

Entwickeln für und mit iPhone
Smartphones stellen eine eigene Kategorie von Computern dar. Es ist deshalb wichtig, dass iPhone-Entwickler selbst ein iPhone benutzen, um ein gutes Verständnis der Plattform und der Benutzererwartungen zu entwickeln.

6.1.2 Objective-C

Für die Programmierung von iPhone Apps wird Objective-C verwendet. Die Sprache ist eine objektorientierte Erweiterung von C und ist in der Grundfunktionalität mit C identisch. Sobald jedoch objektorientiert programmiert wird, unterscheidet sich die Syntax recht stark von gängigen C-verwandten Sprachen wie Java oder C++.

Die Methodensignaturen und das Aufrufen von Methoden unterscheidet sich ebenfalls sehr stark von bekannten Sprachen wie C oder Java. Ebenso das Kennzeichnen von statischen oder Instanzmethoden.

Methodensignatur:
`+ (void) switchToUser:(NSString*)username withPassword:(NSString*)password{ … }`
Statische (+) Methode `switchToUser:withPassword`. Die Methode hat zwei Parameter mit den Namen `username` und `password` vom Typ `NSString` und keinen Rückgabewert `(void)`.

`- (NSString*) username{ … }`
Instanzmethode (-) `username`. Die Methode hat keine Parameter und gibt ein `NSString` Objekt zurück.

Aufrufen einer Methode:
```
NSString *deviceTokenString = [Utilities dataToHexString: deviceToken];
```
Aufruf der statischen Methode `dataToHexString:` auf der Klasse `Utilities` mit dem Parameter `deviceToken`.

```
[[RESTClient sharedInstance ] saveDeviceToken: deviceTokenString];
```
Verschachtelter Aufruf zweier Methoden. Zuerst wird die Methode `sharedInstance` auf der Kasse `RESTClient` aufgerufen, die ein `RESTClient` Objekt zurückliefert. Auf diesem wird dann die Methode `saveDeviceToken` aufgerufen.

Strings:
```
NSLog(@"The variable is %@", myVar);
```
Ausgabe auf der Konsole (`NSLog`) des Strings „`The variable is` “ gefolgt von der Beschreibung des Objektes in `myVar`.

Klassen und Vererbung
In der Header-Datei wird die Klasse definiert.

`@interface RESTClient : NSObject {…`
Die Klasse `RESTClient` erbt von `NSObject`.

`@interface Impulse : NSManagedObject {…`
Die Klasse `Impulse` erbt von `NSManagedObject`.

Visibility
Die Sichtbarkeit der Felder kann wie in C++ definiert werden. Die Sichtbarkeit von Methoden ist immer Public. Es können jedoch auch Methoden ausserhalb der Headerdatei definiert werden, welche dann nicht in der öffentlichen Schnittstelle ersichtlich sind.

```
...
@private
   NSString * _aPrivateField;
@protected
   NSString * _aProtectedField;
   BOOL _anotherProtectedField;
@public
   NSInteger _aPublicField;
}
```

Properties
Properties ermöglichen den Zugriff auf private Felder ähnlich wie bei Java über Setter und Getter. Speziell daran ist, dass ein Property wie ein Feld verwendet werden kann. Mit `retain`, `assign`, `copy`, `nonatomic` und `readonly` können zudem Eigenschaften der Implementation beschrieben werden.

```
@property(retain, nonatomic) NSString * deviceToken;
```
Deklaration des Properties `deviceToken`

```
client.deviceToken = @"01AF46";
[client setDeviceToken:@"01AF46"];
```
Zuweisung eines Strings zu `deviceToken` auf dem Objekt `client`. Oben in der Property Syntax, unten in der Methoden Syntax.

Memory-Management
Objective-C ist nicht garbage-collected, deshalb ist beim Umgang mit Objekten Vorsicht geboten. Der verwendete Mechanismus heisst Reference-Counting. Allgemein gilt der Grundsatz: Wer's erstellt hat, muss es auch wieder abräumen.
Objekte werden von einer oder mehreren Stellen gehalten (retained). Das heisst, dass der Reference Count für jedes Halten um eins erhöht wird. Will ein Objekt-Nutzer sicher gehen, dass das verwendete Objekt nicht plötzlich weg ist, so muss er es für sich ebenfalls sichern (retain). Sobald der letzte retain entfernt wurde, wird das Objekt gelöscht.
Hat man ein Objekt erstellt oder gesichert (retain), so muss es unbedingt mit release wieder freigegeben werden. Ansonsten entstehen Memory-Leaks. Dies führt dazu, dass eine App immer mehr Speicher belegt. Damit so etwas nicht passiert, müssen sämtliche Objekte wieder freigegeben werden.

Ein Objekt kann auf zwei Arten erstellt werden:

- Man erstellt sich das Objekt selber und ist dann für seine Freigabe verantwortlich (`release`). Die Methoden für die Objekterstellung sind: `new`, `alloc` und `copy` (+ Varianten davon).
- Erstellt man das Objekt nicht selber, sondern lässt es sich erstellen, so muss es anschliessend nicht aufgeräumt werden. Die Methoden dazu heissen `create…` oder haben andere Namen die nicht mit `new`, `alloc` oder `copy` beginnen.
  Diese Methoden verwenden den autorelease Mechanismus. Dieser weist ein Objekt einem Autorelease Pool zu, der das Objekt temporär hält. Der Aufrufer der Methode kann dann entscheiden, ob er das Objekt auch halten will. Wenn er das Objekt nicht hält, wird es beim Leeren des Pools freigegeben.

```
+ (NSString *) dataToHexString:(NSData *) data{
   NSMutableString *hexString = [[NSMutableString alloc] initWithCapacity:
data.length * 2];
```

```
    Byte *bytes = (Byte *)[data bytes];
    for(int i = 0; i < data.length; ++i){
        [hexString appendFormat:@"%02.2hhx", bytes[i]];
    }
    return [hexString autorelease];
}
```

Protokolle
In Objective-C gibt es ein Konstrukt, das den Interfaces in Java sehr ähnlich ist. Der Unterschied besteht darin, dass die Methoden darin als @optional deklariert werden können und somit von der implementierenden Klasse nicht implementiert werden müssen.

```
@protocol MyProtocol
- (void)requiredMethod;

@optional
- (void)anOptionalMethod;
- (void)anotherOptionalMethod;

@required
- (void)anotherRequiredMethod;

@end
```

Delegates
Ein Delegate ist eine Art „Service-Klasse“, die sämtliche Callbacks oder Actions ihres Nutzers entgegennimmt. So haben z.B. viele GUI-Klassen ihr Delegate (Controller). Beim Erstellen von umfangreicheren Objekten, z.B. `NSURLConnection`, kann ein Delegate angegeben werden, das die Callbacks der Connection abarbeitet.

Diverses:

- Das Binding der Methoden ist dynamisch.
- Ein Callback benötigt ein Objekt (id) und einen Selector (SEL). Nachdem der Callback eingerichtet wurde, kann er ausgeführt werden:
  [_object performSelector:_selector withObject: param];
- Die Implementations-Dateien heissen *.m und nicht *.c .

6.1.3 App ID

Die App ID ist ein wesentlicher Teil vom iOS Development- und Provisioning-Prozess, die es der App erst erlaubt, verteilt und verwendet zu werden. Zusätzlich kann eine App ID auch dazu genutzt werden, um Key Chain-Daten innerhalb einer Suite von Apps gemeinsam zu verwenden.

Die App ID ist eine Kombination aus der Bundle Seed ID (zehn Zeichen, unique) und einer CF Bundle ID (Bundle Identifier). Eine gleiche Bundle Seed ID erlaubt die gemeinsame Verwendung der Key Chain zwischen mehreren Apps (alle mit gleicher App ID). Die Registrierung der App ID inklusive Bundle Identifier ist nötig, um den *APNs* benutzen zu können.

Der Bundle Identifier-Teil kann durch wild-cards (*) ersetzt werden, sodass eine einzelne App ID zum Erstellen und Installieren mehrerer Apps genutzt werden kann.

Falls keine wild-cards benutzt werden, muss der Bundle-Identifier-Teil in XCode eingetragen werden. Der Bundle Seed ID-Teil muss nicht in XCode eingetragen werden. Apps mit wild-card App ID können den APNs nicht verwenden.

Aufbau einer App ID

Feld	Beschreibung
Description	Name oder Beschreibung der App
Bundle Seed ID (App ID Prefix)	Wird jeweils generiert (oder besteht bereits als Teil einer App ID)
Bundle Identifier (App ID Suffix)	Eindeutige Bezeichnung für die App. Bevorzugtes Format: com.domainname.appname

Für die Centrado App wird die folgende App-ID verwendet:

Centrado Client
LP49796WL2.ch.centrado.CClient

6.1.4 Provisioning und Erstellen der App

Provisioning

Damit die App auf ein Gerät verteilt werden kann, wird ein *Provision Profile* benötigt.

Im iOS Development Center (http://developer.apple.com/devcenter/ios/index.action) kann dazu auf der rechten Seite das iOS Provisioning Portal geöffnet werden. Im Abschnitt Provisioning kann dann die Datei erstellt werden. Dazu kann man einen normalen Developer-Account (kein Admin nötig) verwenden.

Die Profiles bestehen aus einem Profilnamen, den Zertifikaten der Entwickler, der App ID und den Geräten, auf die dann deployt werden darf.

Nachdem das Profile erstellt ist, kann man es im XCode Organizer herunterladen und verwenden. Das verbundene Gerät muss mit der Datei geladen werden. Der passende Button heisst „Use for development".

Erstellen der App

Sobald die zu programmierende App über ein paar erste Funktionen verfügt, kann sie auf ein Gerät deployt werden.

Handelt es sich um eine App ohne Apple Push Notification Service, so reicht es, das Provisioning Profile auf dem Gerät zu installieren.
Benötigt man aber auch noch den APNs, so muss der hintere Teil der App-ID bei XCode in den Projekteigenschaften im Feld Identifier eingetragen werden. Dabei muss der angegebene Wert mit dem Bundle Identifier (App ID Suffix), der online erstellten App ID, übereinstimmen.

6.1.5 MVC

Das GUI von iPhone Apps wird nach dem MVC-Pattern programmiert. Da es unterschiedliche Beschreibungen zum MVC-Patterns gibt, wird an dieser Stelle erläutert, wie dieses Pattern in Cocoa verwendet wird. Die hier beschriebene Variante weicht in mehreren Punkten von der Beschreibung in [POSA96] ab. Eine detaillierte Beschreibung ist in [CocoaMVC] zu finden.

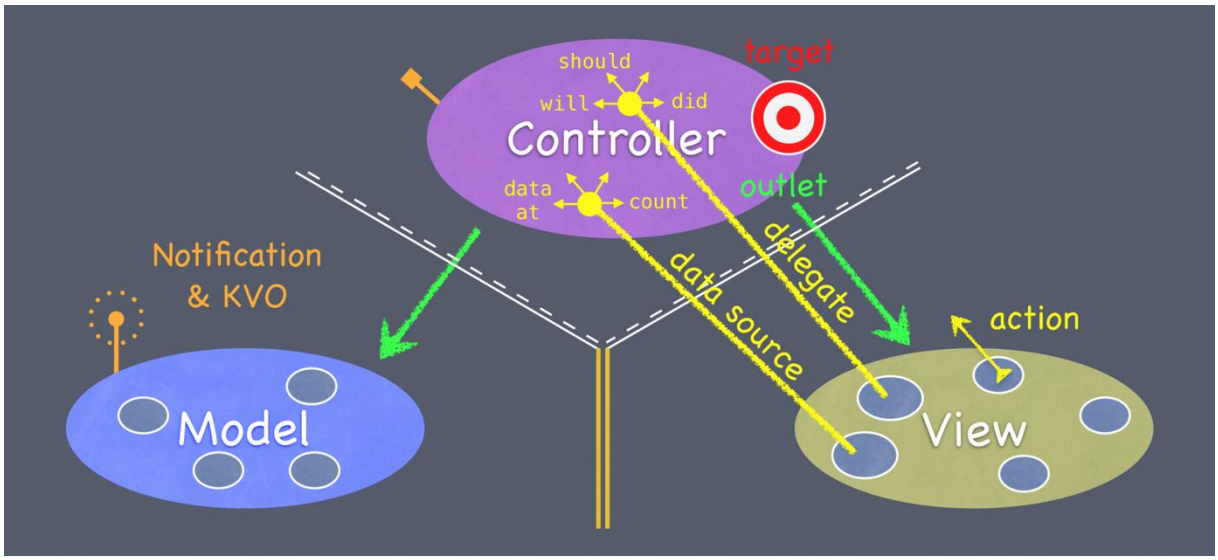

Abbildung 6: Aufbau MVC in Cocoa (Based on a work at cs193p.stanford.edu)

Der Controller kann direkt mit dem Model und der View kommunizieren. Er liest Werte aus dem Model und setzt diese in der View. Die View und das Model kennen den Controller nicht und können nur über Callback-Mechanismen mit diesem kommunizieren. Das Model und die View dürfen nicht miteinander kommunizieren. In der Abbildung 6 ist dies durch die doppelte Sicherheitslinie dargestellt. Sämtliche Kommunikation wird durch den Controller erledigt.

Komponente	Beschreibung
View	Eine View besteht aus grafischen Elementen. Dies können vordefinierte Klassen wie z.B. Bilder (`UIImageView`), Textfelder (`UITextField`) oder Buttons (`UIButton`) sein oder auch eigene Klassen. Views sind für das Darstellen von Informationen und das Erkennen von Benutzerinteraktionen verantwortlich. Für das Bearbeiten der Views existiert ein grafischer Editor.
Controller	Der Controller steuert die View und das Model. Er kann auf Ereignisse der View oder des Models reagieren und die entsprechenden Aktionen durchführen. Properties des Controllers, die View-Elemente referenzieren, werden auch Outlets genannt.
Model	Das Model enthält Daten und Methoden, die unabhängig von der Darstellung existieren.

Callback-Mechanismen

Da View und Model den Controller nicht direkt kennen, gibt es verschiedene Callback-Mechanismen, mit denen die beiden den Controller benutzen und Ereignisse mitteilen können.

Mechanismus	Beschreibung
Target Action	Target Action wird verwendet, um einzelne Ereignisse auf View-Elementen an den Controller weiterzuleiten. Dies kann z.B. die Mitteilung sein, dass ein Button gedrückt wurde. Der Controller registriert sich dafür auf dem View-Element. Er gibt sich selbst als Empfänger (Target) der Benachrichtigung und einen Selektor auf eine seiner Methoden (Action) an.

Delegate	Delegates sind Objekte, die ein bestimmtes Protokoll implementieren. Sie werden verwendet, um das Verhalten der View-Elemente zu steuern, Ereignisse zu behandeln oder das View-Element mit Daten zu versorgen. Die Methoden der Delegate Protokolle beginnen oft mit `should`, `did` oder `will`. Die Namen der Delegate Protokolle enden meist auf `Delegate` oder `DataSource`.
KVO	KVO steht für Key Value Observer. Mit diesem Mechanismus kann sich der Controller auf Veränderungen von bestimmten Properties auf dem Model registrieren.
Notification	Notifications werden durch ein `NotificationCenter` Objekt verteilt. Der Controller kann sich für den Empfang von Notifications bei einem `NotificationCenter` registrieren. Wenn ein Model dann eine Notification an das `NotificationCenter` sendet, wird der Controller von diesem benachrichtigt.

6.1.6 Local Notifications and Push Notifications

Abbildung 7: Local Notification

Lokale und Push Notifications ermöglichen es, eine App - die nicht läuft - in den Vordergrund zu holen und etwas anzuzeigen. Dies können eine Meldung, ein bevorstehender Termin oder neue Daten auf einem Server sein.

Wenn die beiden Notifications (lokal und push) dem User gezeigt werden, sehen sie genau gleich aus. Wahlweise kann eine Meldung gezeigt und, sofern gewünscht, eine App mit einem *Badge* markiert werden. Damit der User auf die Meldung aufmerksam wird, kann sie mit einem Signalton ergänzt werden.

Push Notifications wurden mit iOS 3.0 eingeführt, die lokalen Notifications mit iOS 4.0.

Da der User nun weiss, dass etwas ansteht, kann er die App starten und auf das neue Element zugreifen. Er hat ebenfalls die Möglichkeit, die Notifikation zu ignorieren: In diesem Fall wird die App nicht gestartet.

Sinn und Zweck

Es kann jeweils nur eine App auf einem iPhone oder iPad aktiv im Vordergrund sein. Viele Apps arbeiten zeitbasiert oder sind mit der Umgebung verbunden, die Events liefert, an denen der User interessiert sein könnte, obwohl die App nicht im Vordergrund ist.

Lokale und Push Notifications erlauben den Apps, ihre User zu benachrichtigen, wenn solche Events auftreten.

Apple Push Notification Service
Für den APNs stehen zwei Umgebungen zur Verfügung: Development und Produktion. Beide Dienste sind unter separaten Adressen verfügbar und benötigen ein unterschiedliches "Push SSL Certificate". Für die Bachelorarbeit wird nur die Development-Umgebung verwendet.

Referenzen

- About Local Notifications and Push Notifications
  https://developer.apple.com/library/ios/#documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Introduction/Introduction.html (Stand 11.06.11)
  (This document previously was titled Apple Push Notification Service Programming Guide.)

6.1.7 Verschicken von Push Notifications

Zum Testen der Push Notifications wird ein virtueller Server der HSR verwendet. Auf diesem befindet sich ein Java-Client, der mit einem beigefügten Zertifikat in der Lage ist, den Apple Push Notification Service (APNs) zu nutzen.

6.1.7.1 APNs-Bibliothek

Java-apns ist ein Java Client zum Versenden von Push Notifications über den Apple Push Notification Service. Die Bibliothek liefert eine gut skalierbare Schnittstelle zum APNs und bleibt dabei einfach und modular. Die Schnittstelle ist ebenfalls einfach gehalten und deckt die meisten Funktionen dennoch ab. Die Bibliothek ist unter einer BSD-3-Lizenz (Siehe Anhang 13.2) verfügbar und kann somit für die Arbeit verwendet werden.

Referenzen

- java-apns auf GitHub https://github.com/notnoop/java-apns (Stand 11.06.11)
- java-apns JARs https://github.com/notnoop/java-apns/downloads (Stand 11.06.11)

6.1.7.2 APNs mit PHP

Neben Java kann das Senden von Push Notifications auch mit PHP programmiert werden.
Mit PHP ist es sehr einfach die Payload herzustellen. Es wird ein Array erstellt, der dann anschliessend zu JSON konvertiert wird. Dieses Beispiel funktioniert auch mit eigener Payload:

```
$payload['aps'] = array('alert' => 'This is the alert text', 'badge' => 1, 'sound'
=> 'default');
$payload['server'] = array('serverId' => $serverId, 'name' => $name);
$output = json_encode($payload);
```

Die JSON-Meldung sieht dann so aus:

```
{
     "aps" : { "alert" : "This is the alert text", "badge" : 1, "sound" :
"default" },
     "server" : { "serverId" : 1, "name" : "Server name"}
}
```

Anschliessend wird eine Verbindung zum APNs hergestellt. Dazu wird das Zertifikat benötigt. Danach wird die Meldung mit dem Token und ein paar weiteren Bytes ergänzt und verschickt.

Fazit:
Der Autor des Artikels [APNsPHP], aus dem auch das PHP Code Stück stammt, äussert sich besorgt, ob PHP für einen kontinuierlichen Serverprozess geeignet ist.

Das Beispiel wurde von uns nicht getestet, da wir zu diesem Zeitpunkt schon eine funktionsfähige Lösung in Java hatten.

6.1.8 Verschicken von Push Notifications (Drittanbieter)

Anfangs der Arbeit wurde das Versenden von Push Notifications als komplex eingestuft. Deshalb wurde auch mit dem Gedanken gespielt, das Versenden als Service bei einem Drittanbieter zu beziehen.

Da die Notifications aber recht einfach verschickt werden können, wird dafür nicht auf ein Drittanbieter zurückgegriffen. Drittanbieter könnten sich in späteren Versionen der App (nicht Teil der Bachelorarbeit) z.B. für „in app purchase“ noch als nützlich erweisen.

6.1.8.1 iLime:Push

Die Analyse wurde am 1.3.2011 erstellt. Der Anbieter hat den Service per 7.3.2011 jedoch eingestellt.

iLime:Push ist eine Lösung, um die Integration mit dem Apple Push Notification Service zu vereinfachen. Mit einer Reihe von REST APIs und einer webbasierten Administrationskonsole kann der Service benutzt und konfiguriert werden.

Mit iLime:Push ergeben sich die folgenden Vorteile:

- geringe Hosting- und Entwicklungskosten
- gute Skalierung des Services falls mehr Notifications verschickt werden sollen
- Schnittstellen zum APNs über zuverlässige RESTful APIs

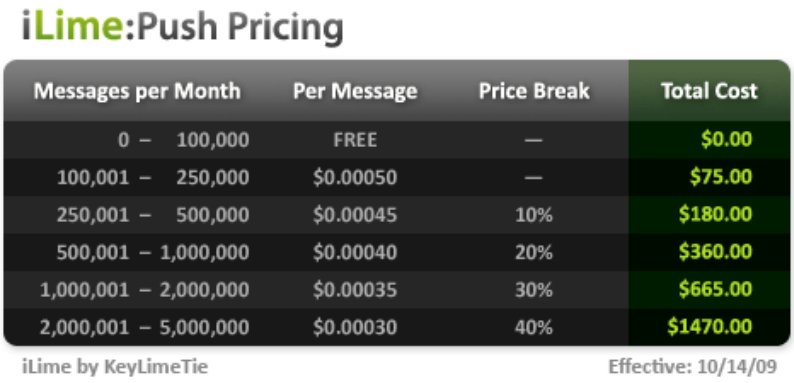

Abbildung 8: iLime Push Preise

Referenzen

- iLime Homepage https://www.ilime.com (Stand 1.3.11)

6.1.8.2 iLime:Purchase

Die Analyse wurde am 1.3.2011 erstellt. Der Anbieter hat den Service per 7.3.2011 jedoch eingestellt.

Dieser Service ermöglicht das Verkaufen von App-Inhalten via In App Purchase and Store Kit.

Mit iLime:Purchase ergeben sich folgende Vorteile:

- Hosten von Content, den App-User dann direkt von ihrem iPhone kaufen können.
- Das Programmieren einer eigenen Applikation, die den User mit Content versorgt, entfällt.

Abbildung 9: iLime Purchase Preise

6.1.8.3 Urban Airship

Push Notifications:

Uraban Airship (kurz UA) bietet einen "Push Composer" an, mit dem man Push Notifications ähnlich wie SMS' online verschicken kann. Beim Verschicken kann zwischen Broadcast, einigen Geräten oder Einzelgerät unterschieden werden.

Eigenschaften

- Der Kunde kann einstellen, wann er die Meldungen empfangen will.
- Einfache Integration: Aufgebaut auf open-source Bibliotheken und simplen RESTful APIs, welche einfach in eine mobile App integriert werden können.
- Report: Zeichnet auf, ob der User die empfangenen Inhalte liest. Die Betrachtungszeit und wie viele Male das Dokument geöffnet wurde, wird ebenfalls aufgezeichnet.

In-App Purchase:

In-App Purchase erlaubt dem Kunden Upgrades, neuen Content und auch ergänzende Features direkt in der App zu kaufen, ohne den App Store besuchen zu müssen. Somit wird der Kauf sehr vereinfacht und auch verkürzt.

Rich Push:

Rich Push ergänzt das konventionelle Push System um weitere Kapazitäten. Damit kann man nun HTML, Video, Audio, Karten und anderen Rich-Content der Meldung anhängen.

Nicht wie beim einfachen Push ist die Message nach dem Lesen (durch den User) einfach weg. Der Service hilft beim Betrieb einer Inbox, wichtigen Content zu speichern. Das ist vor allem für Daten geeignet, die der Kunde nicht sofort sehen möchte, sondern später in Ruhe anschauen will.

Eigenschaften

- Speichert alle Meldungen in einer Inbox auf dem iPhone, sodass der Kunde bei Gelegenheit darauf zugreifen kann.

Subscriptions

Dieser Service ermöglicht das Abonnieren von Inhalten mit dem iPhone. Damit ist eine feste Kundenbindung möglich. Es wird dem Kunden ermöglicht, Publikationen innerhalb der App zu abonnieren.

Eigenschaften:

- Der App-Vetreiber legt den Preis und die Dauer des Abonnements fest, der Rest wird von UrbanAirship erledigt.
- Der App-Vertreiber kann die gekauften Abonnemente jederzeit administrieren und überwachen.

Dieser Anbieter ermöglicht ebenfalls das Aufzeichnen von Gebrauchsstatistiken. Durch den Rich Push Service hebt sich UA im Vergleich zu iLime doch leicht hervor. Deshalb wird im nächsten Abschnitt etwas näher darauf eingegangen.

UA's Rich Push Library
Es heisst zwar Rich „Push", hat jedoch mit dem APNs nicht mehr viel gemeinsam. Vielmehr versteckt sich hinter diesem Namen ein umfangreiches System unter anderem mit Benutzerverwaltung.

Rich Push Client Library - iOS

- UA Rich Push ist ein drop-in Interface, das es erlaubt, Push Notifications zu speichern und Rich-Content in einer Inbox persistent abzulegen.
- Die Rich Push Inbox ist als statische Bibliothek verfügbar.

Download
Unter http://urbanairship.com/docs/richpush_client.html#overview kann sowohl die Bibliothek als auch eine Sample App heruntergeladen werden.

Wie schnell man sich mit diesem Framework zurechtfindet ist nicht bekannt. Im Rahmen der Suche nach "Push-Drittanbietern" wird hier aber nicht mehr näher darauf eingegangen.

Preise:
Basic Package - Nur Push Notifications (gratis). Man kann bis zu 1 Million Gratis-Meldungen pro Monat schicken. Jede weitere Meldung kostet $ 0.0010 .

Rich Push Messages - Basis Preise starten bei $0.0025 / message

In-App Purchase and Subscription

Volumen	Preis
Less than 25MB	$0.05 per download*
Between 25MB and 150MB	$0.10 per download*
More than 150MB	Contact us for pricing
Subscriptions	$0.15/user/month (incld. downloads)

Referenzen

- Einstieg: http://urbanairship.com/ (Stand 11.6.11)
- Preise: http://urbanairship.com/pricing/ (Stand 11.6.11)
- Rich Push http://urbanairship.com/docs/richpush_index.html (Stand 11.6.11)

6.1.8.4 Mac OS X Server v10.6: Push Notification Server

Der Push Notification Service in Mac OS X Server v10.6 unterstützt das Senden von Push Notifications auf Mac OS X v10.6 Clients mit iCal und Mail Applikationen.
Zusätzlich kann der Service Notifications an third-party Anwendungen schicken, die das ServerNotification Framework verwenden.

Betroffene Produkte:
iPhone, iPad, Mac OS X Server 10.6, iPod touch

Weitere Informationen:
iOS Mail und Kalender Apps sind zurzeit nicht unterstützt (Stand: 21 Dezember, 2010).

Referenzen

- http://support.apple.com/kb/ht3947 (Stand 11.6.11)

6.1.8.5 Push Server Zertifikat

Für jede App-ID, die APNs verwenden soll, muss ein eigenes Zertifikat erstellt werden. Das Zertifikat erlaubt z.B. dem firmeninternen Push Server, sich mit dem APNs zu verbinden.

6.1.9 Internationalisierung

Mit Internationalisierung kann eine App mehrsprachig verwendet werden.

Verwendung

NSLocalizedString	Die Funktion `NSLocalizedString` gibt einen lokalisierten String zurück. Der erste Parameter dient dabei als Key, der Zweite ist für den Übersetzer gedacht (Kontext und ev. eine kurze Erklärung). Verwendung: `text = NSLocalizedString(@"loading...", @"message that is displayed while loading the impulse");`
NSString stringWithFormat:	Diese Methode erlaubt das Zusammensetzen von Text. `[NSString stringWithFormat:@“Posteingang: %i ungelesene Impulse“, unreadMessages];`
NSBundle	Die Source-Code-Dateien gelten für alle Sprachen. Unterschieden wird erst bei resource-Dateien. Die Ressourcen sind in einem Ordner abgelegt, der mit dem standardisierten Name der Sprache angeschrieben ist.

Für die Internationalisierung verwendete Datei: localizable.strings

Die Datei ist in Key-Value-Pairs aufgebaut. Jedes Paar ist mit der Übersetzungshilfe angeschrieben.

```
/* Title of a button that cancels a action */
"CancelButton" = "Cancel";

/* Label for Close Button */
"CloseButton" = "Close";
```

Für jede benötigte Sprache muss eine Datei erstellt werden, bei der die Values in die entsprechende Sprache übersetzt wurden. Localizable.strings werden auch String-Tables genannt. Für Singular- und Pluralformen sollte immer ein separates Key-Value-Pair angelegt werden, auch wenn sich die Texte in der eigenen Sprache nicht unterscheiden.

Wichtige Begriffe

Begriff	Beschreibung
Internationalization	Programm vorbereiten, damit es lokalisiert werden kann.
Localization	Alle Elemente, mit denen der Benutzer in Kontakt kommt, an eine Sprache und Kultur anpassen.
Fallback-Strategie	Wird eine geforderte Sprache nicht gefunden, wählt die App die Sprache, in der die App entwickelt wurde (meistens Englisch).

Tools

Tool	Beschreibung
genstrings	genstrings ist eine Konsolenapplikation, welche Source-Code-Dateien nach `NSLocalizedString` durchsucht und alle Keys in eine Datei mit dem Namen localizable.strings schreibt. Die Übersetzungshilfe (zweiter Parameter von `NSLocalizedString`) wird ebenfalls hineingeschrieben. genstrings wird zusammen mit XCode installiert.
LocalizableStringMerge	LocalizableStringMerge ist ein komfortableres Tool mit einer grafischen Oberfläche. Es bietet unter anderem zwei Funktionen zum Sortieren und Zusammenfügen (merge) von .strings-Dateien. LocalizableStringMerge ist ein kostenpflichtiges Programm. Es kann über den Mac App Store bezogen werden.

Referenzen

- Dokumentation von Apple: Resource Programming Guide > String Resources
- Dokumentation von Apple : Introduction to Internationalization Programming Topics https://developer.apple.com/library/ios/#documentation/MacOSX/Conceptual/BPInternational/BPInternational.html (Stand 11.6.11)

6.1.10 Core Data

Core Data ist ein Persistenz-Framework für Mac OS X (10.4+) und iOS (3.0+). Als Store können unter anderem die Optionen SQLite, XML, Binary oder in-Memory verwendet werden.

Aufbau und Klassen:

Klasse	Beschreibung
NSManagedObject	Ein Managed Object repräsentiert einen Datensatz bzw. eine Zeile.
NSEntityDescription	Eine Entity Description enthält Meta-Informationen über eine Entität.
NSFetchRequest	Ein Fetch Request beschreibt, welche Managed Objects geladen werden sollen. Auf NSFetchRequest kann auch festgelegt werden, wie viele Objekte auf einmal geladen werden sollen.
NSPredicate	NSFetchRequest benutzt NSPredicate, um zu definieren, welche Objekte geladen werden sollen.
NSSortorder	Mit NSSortorder kann angegeben werden, wie das Resultat von NSFetchRequest sortiert werden soll.
NSManagedObjectContext	Der Managed Object Context verwaltet die Managed Objects. Über ihn werden auch Abfragen ausgeführt.
NSPersistentStoreCoordinator	Ein Persistent Store Coordinator verwaltet die Zugriffe auf die persistenten Daten. Er hat eine Referenz auf das ManagedObjectModel, welches die Entities in einem Store beschreibt. Der Coordinator ist das zentrale Objekt in einem Core Data Stack. In den meisten Apps hat es nur einen Store, der StoreCoordinator wäre aber in der Lage, verschiedene Stores zu einem Store zusammen zu fassen. Er dient dabei als Facade, damit seine Contexte auf die Stores zugreifen können.

NSManagedObjectModel	Das Managed Object Model beschreibt die Entitäten und ihre Beziehungen. Es kann in XCode in einer Datei mit der Endung .xcdatamodel erstellt und als Ressource abgelegt werden. Es kann aber auch zur Laufzeit geladen oder erzeugt werden.

Attribute von Managed Objects
Attribute können als Type eine Auswahl von 10 Typen haben oder einen Spezial-Typ wie Custom und Transform.
Custom Properties müssen von Hand implementiert werden. Um die Werte der Custom Properties persistent zu machen, müssen die benötigten Werte in „normalen" persistenten Properties abgelegt werden. Transform Properties können von einem beliebigen Typ sein. Sie werden serialisiert und als BLOB abgelegt.

Multi Threading
Core Data ist nicht thread-safe. Sowohl der Managed Object Context, wie auch die Managed Objects gehören zu einem Thread und dürfen nicht von einem andern Thread benutzt werden.
Wenn z.B. ein Managed Object in einem Hintergrund-Thread bearbeitet wird und dann dargestellt werden soll, so wird nicht das Objekt selbst, sondern nur die `objectID` an den andern Thread übergeben. In diesem Thread kann dann über den Managed Object Context mit der Methode `objectWithID:` oder `existingObjectWithID:error:` das Managed Object für diesen Thread geholt werden.

6.1.11 Basic Authentication

Um eine Basic Authentication durchzuführen, wird am besten eine eigene Klasse erstellt. Im Konstruktor der Klasse wird eine `NSURLConnection` erstellt und abgeschickt.

Da der Vorgang asynchron ist, muss die Klasse bei der `NSURLConnection` als `delegate` angegeben und verschiedene Methoden implementiert werden. Die wichtigste ist `didReceiveAuthenticationChallenge` - dort werden die Credentials der eingetroffenen Challenge übergeben.

Während die Nutzdaten übertragen werden, wird die Methoden `didReceiveData` wiederholt für die neu eingetroffenen Daten aufgerufen. Wenn die Übertragung abgeschlossen ist, wird `connectionDidFinishLoading` aufgerufen.

Referenzen

- http://iphonedevelopertips.com/networking/handling-url-authentication-challenges-accessing-password-protected-servers.html (Stand 11.6.11)

6.1.12 Versenden von Parametern

Die Analyse zum Übermitteln von Parametern (Benutzername und Passwort) besteht aus zwei Komponenten; der Serverseite (Java Servlet) und der Clientseite (iPhone App).

Anmerkung: Der Login der App funktioniert jetzt mit http Basic Authentication.

Serverseite:
Auf der Serverseite wird ein Servlet verwendet, welches von einem Apache Server gehostet wird. Das Servlet kann Benutzername und Passwort entgegennehmen. Aufgerufen wird es mit folgendem Link:

http://152.96.56.36/LoginServletV1.0withSource/LoginServAlias?u=hmuster&p=secret

Es findet eine Überprüfung von Benutzername und Passwort statt. Im Erfolgsfall wird der Text „Hallo HM“ dargestellt.

Da die Get-Parameter noch codiert werden müssen, existiert ein weiteres Account mit dem Benutzernamen: sz und dem Passwort: % & . Somit kann überprüft werden, ob die App auch Sonderzeichen richtig codiert. Der entsprechende Link lautet:

http://152.96.56.36/LoginServletV1.0withSource/LoginServAlias?u=sz&p=%25+%26

Zu beachten:

- Port 80 muss an der Windows-Firewall geöffnet werden.
- Der Apache-Server ist normalerweise auf 8080 (Umstellen: Ein Neustart des Servers reicht nicht, Eclipse muss ebenfalls neu gestartet werden).

iPhone App:
Für das Absenden der Parameter wurde eine entsprechende App geschrieben. Dabei stellte sich heraus, dass gewisse Zeichen (z. B & und /) nicht korrekt codiert wurden. Deshalb musste das Codieren noch um eine Funktion aus der Foundation erweitert werden. Der Code ist nun auch in der Lage, Zeichen wie % & zu verarbeiten.

Referenzen

- http://cybersam.com/programming/proper-url-percent-encoding-in-ios (Stand 11.6.11)

6.1.13 Sichere Verbindung

In diesem Abschnitt wurde das Herunterladen von Inhalten über eine sichere Verbindung getestet. Dazu wurde die Startseite von Google SSL Beta heruntergeladen. Der passende Link dazu lautet: https://encrypted.google.com/ (Stand 11.6.11)

Der HTML-Quelltext wurde anschliessend in einem Textfeld dargestellt. Wir gehen davon aus, dass es mit einer JSON- oder PList-Datei ebenfalls funktionieren wird.

6.1.14 Background Tasks

Wenn der Benutzer eine Push Notification erhält, so möchte er den Impuls häufig gleich sehen. Er muss deshalb kurz warten, bis der Content vollständig geladen ist.

Das Laden im Hintergrund kommt dann zum Zug, wenn der Benutzer die Notification akzeptiert, die App dann aus Zeitgründen aber wieder schliesst[2].

Die beste Lösung ist, der App nach dem Beenden noch zusätzliche Zeit einzuräumen. In dieser können grössere Dateien geladen werden. Der Vorgang ist im nächsten Abschnitt detailliert beschrieben.

> ***„ Executing Code in the Background***
> *Most applications that enter the background state are moved to the suspended state shortly thereafter. While in this state, the application does not execute any code and may be removed from memory at any time. Applications that provide specific services to the user can request background execution time in order to provide those services.*
> ***Not all devices support multitasking***
> *The ability to execute code in the background is not supported on all iOS-based devices. Even devices running iOS 4 or later may not have the hardware to support multi-*

[2] Das Pollen nach Impulsen kommt unserer Meinung nach nicht in Frage. Einerseits macht es von der Architektur her keinen Sinn, andererseits muss das Gerät jeweils eine Datenverbindung herstellen.

tasking. In those cases, the system reverts to the previously defined behaviour for handling applications. Specifically, when an application quits, it is terminated and purged from memory.

Completing a Finite-Length Task in the Background

Any time before it is suspended, an application can call the beginBackgroundTaskWithExpirationHandler: method to ask the system for extra time to complete some long-running task in the background. If the request is granted, and if the application goes into the background while the task is in progress, the system lets the application run for an additional amount of time instead of suspending it. (The backgroundTimeRemaining property of the UIApplication object contains the amount of time the application has to run.)

You can use task completion to ensure that important but potentially long-running operations do not end abruptly when the user leaves the application. For example, you might use this technique to save user data to disk or finish downloading an important file from a network server. There are a couple of design patterns you can use to initiate such tasks:

- Wrap any long-running critical tasks with beginBackgroundTaskWithExpirationHandler: and endBackgroundTask: calls. This protects those tasks in situations where your application is suddenly moved to the background.

- Wait for your application delegate's applicationDidEnterBackground: method to be called and start one or more tasks then."

Quelle: [BackExec]

6.2 Lösungsansätze und Ergebnisse

6.2.1 Datenformat: JSON und PList

Für das Übertragen von Daten zwischen dem Server und dem iPhone gibt es verschiedene Formate, die verwendet werden können. Im Folgenden werden mögliche Formate beschrieben und begründet, warum die Entscheidung auf JSON fiel.

Formate:

Bei der Auswahl wurden zwei Formate betrachtet. JSON und PList. Die Verwendung eines allgemeinen XML-Formats wurde nicht weiter untersucht.

6.2.1.1 JSON

Die Syntax von Java Script Object Notation (JSON) basiert auf Java Script und kann sowohl von Mensch als auch von Maschinen leicht gelesen werden.

Beispiel:

```
{
    "id":12345,
    "language": "DE",
    "title": "Dehnung der Nackenmuskeln",
    "videoURL": null,
    "imageURLs": [
        "http://domain.com/img-k3-0001-1.png",
        "http://domain.com/img-k3-0001-2.png"]
}
```

JSON definiert Strukturen, die aus Objekten, Arrays und Werten bestehen. Ein Objekt ist eine Menge von Key-Value-Pairs. Der Key ist ein String, der Value ein Wert.

Der Bereich eines Werts kann eines der Schlüsselwörter true, false oder null, ein String, eine Zahl, ein Array oder ein Objekt sein.
Mit dieser Struktur lassen sich ganze Bäume bilden. JSON wird durch den RFC 4627 beschrieben. Die Seite json.org gibt einen sehr guten Überblick zu JSON.

Unterstützte Datentypen

Type	Werte
Bool	`false`, `true`
Collections	Object, Array
Zahlen	Number
Zeichen	String
Kein Wert	`null`

6.2.1.2 PList

Property List ist ein Format, das ursprünglich in NeXTStep verwendet wurde und von Apple mit Mac OS X sehr verbreitet eingesetzt wird. iOS, das auf Mac OS X basiert, verwendet dieses Format ebenfalls an vielen Stellen.
Wie auch bei JSON besteht eine Property List aus einfachen Strukturen, die zu Bäumen kombiniert werden können. Die Struktur, die in JSON als Objekte bezeichnet wird, wird bei PLists Dictionary genannt. Array werden ebenfalls unterstützt. Bei den primitiven Datentypen werden Zahlen in integer und real unterschieden. Im Gegensatz zu JSON sind auch Repräsentationen für Datum und Binärdaten definiert.
PList kennt 3 Repräsentationen.

- Das alte NeXTStep-Format, dass sich nur sehr gering von JSON unterscheidet.
- Ein XML-Format
- Binärformat (siehe http://opensource.apple.com/source/CF/CF-550/CFBinaryPList.c)

Beispiel im XML Format:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
	<key>id</key>
	<integer>1234</integer>
	<key>language</key>
	<string>DE</string>
</dict>
</plist>
```

PList Datentypen

Type	Werte
Bool	True, false
Collections	Array, dict
Zahlen	real, integer
Zeichen	string
Kein Wert	Null
Zeitangaben	date
Binärdaten	data

6.2.1.3 JSON-Libraries für iOS

Für JSON gibt es keine API im iOS SDK. Es gibt aber mehrere Open Source Projekte, die JSON-Parser und Generatoren implementieren.

TouchJSON /TouchCocoa

Quelle	https://github.com/TouchCode/TouchJSON
Lizenz	MIT License
Letzte Aktivität	05.03.2011
Letzter Release	-
Autor	Jonathan Wight et al.

Bemerkungen:
-Unit Tests vorhanden

Verwendung:
Die Bibliothek besitzt je eine Klasse zum Serialisieren und Deserialisieren: CJSONSerializer und CJSONDeserializer.

```
-(id)initWithJsonData:(NSData *)data{
  [super init];
  NSError *parseError;
  NSDictionary *values = [[CJSONDeserializer deserializer]
                          deserializeAsDictionary:data error:&parseError];
  self.language = [values valueForKey:@"Language"];
  self.subject = [values valueForKey:@"Title"];

  return self;
}
```

```
-(NSData*)writeToJson{
  NSMutableDictionary *dictionary = [[NSMutableDictionary alloc] init];
  [dictionary setObject:self.language forKey:@"Language"];
  [dictionary setObject:self.subject forKey:@"Title"];

  NSError *error;
  NSData * data = [[CJSONSerializer serializer]
                   serializeDictionary:dictionary error:&error];

  [dictionary release];

  return data;
}
```

MTJSON

Quelle	https://github.com/mysterioustrousers/MTJSON
Lizenz	MIT License
Letzte Aktivität	08.02.2011
Letzter Release	-
Autor	Adam Kirk

Bemerkungen
- nur eine Klasse
- sehr neu
- sieht nach generiertem Parser aus
- letzter Kommentar: “fixing memory leaks”
- arbeitet mit `NSString` als Input und Output
- gibt beim Parsen immer `NSMutableArray` zurück

JSONKit

Quelle	https://github.com/johnezang/JSONKit
Lizenz	BSD License
Letzte Aktivität	05.03.2011
Letzter Release	05.02.2011, (Version 1.3)
Autor	John Engelhart

Bemerkungen
- laut Autor sehr schnell

JSON-Framework

Quelle	http://code.google.com/p/json-framework/ http://stig.github.com/json-framework/
Lizenz	BSD
Letzte Aktivität	18.02.2011
Letzter Release	23.09.2010, (Version 2.3.2)
Autor	Stig Brautaset

Bemerkungen
- Tests

Fazit
Wir haben uns für TouchJSON entschieden, da die Library einen soliden Eindruck macht:

- Unit Tests für Serialisieren und Deserialisieren
- Einfache Anwendung
- „Touch JSON tends to be the best in terms of speed and unit test coverage. It's also the most widely adopted and actively developed.“ (http://stackoverflow.com/questions/286087/best-json-library-to-use-when-developing-an-iphone-application (Stand 11.6.11))

6.2.1.4 PList-Libraries in Java

Javaplistreader

Quelle	http://sourceforge.net/projects/javaplistreader/
Lizenz	GPL
Letzte Aktivität	17.07.2009
Letzter Release	02.28.2007
Author	Gie Spaepen

Bemerkungen
- Reader und Writer
- Nur XML. Kein Binärformat.

JPList

Quelle	http://jplist.sourceforge.net/
Lizenz	L-GPL
Letzte Aktivität	17.07.2009
Letzter Rel.	26.05.2005
Autor	Sujit Pal

Bemerkungen
- ASCII Format

Apache Configuration

Quelle	http://commons.apache.org/configuration/apidocs/org/apache/commons/configuration/plist/XMLPropertyListConfiguration.html

Bemerkungen
- Kein Binary, nur XML

6.2.1.5 Bewertung und Fazit

PList

Pro	Contra
Support im Framework	XML-Repräsentation (Standard) hat mehr Overhead (Tags)
Definierte Elemente für Data und Date	Java Libraries, die das Binärformat unterstützen, sind schwer zu finden/nicht vorhanden.

JSON

Pro	Contra
Als RFC Dokumentiert	Kein nativer Support in iOS
Wird auch für Push Notifications und im *In App Purchase Programming Guide* verwendet.	
Kleinere Dateigrösse (bei kleinen Dateien sogar kleiner als Binary PList und komprimiertes XML PList)	

Fazit

Die Tatsache, dass JSON kompakter und serverseitig besser unterstützt wird als PList, hat uns dazu bewogen, JSON als Datenübertragungsformat zu wählen. Dass Apple für das Versenden von Push Notifications und die *In App Purchase API* selbst auch JSON verwendet, lässt hoffen, dass in einem kommenden iOS-Release auch JSON APIs von Apple enthalten sind.

Referenzen

- JSONDefinition http://json.org (Stand 11.6.11)
- JSON RFC http://tools.ietf.org/html/rfc4627 (Stand 11.6.11)
- PList http://en.wikipedia.org/wiki/Property_list (Stand 11.6.11)
- PList dtd http://www.apple.com/DTDs/PropertyList-1.0.dtd (Stand 11.6.11)
- In App Purchase Programming Guide http://developer.apple.com/library/ios/#documentation/NetworkingInternet/Conceptual/StoreKitGuide/Introduction/Introduction.html (Stand 11.6.11)

6.2.2 Datenspeicherung auf dem iOS

6.2.2.1 NSData

Zum Ablegen der Daten auf dem Dateisystem wurde die Klasse `NSData` verwendet. Sie eignet sich zum Speichern von Binärdaten bestens. Auch das Speichern in einer Datei ist recht einfach zu programmieren.

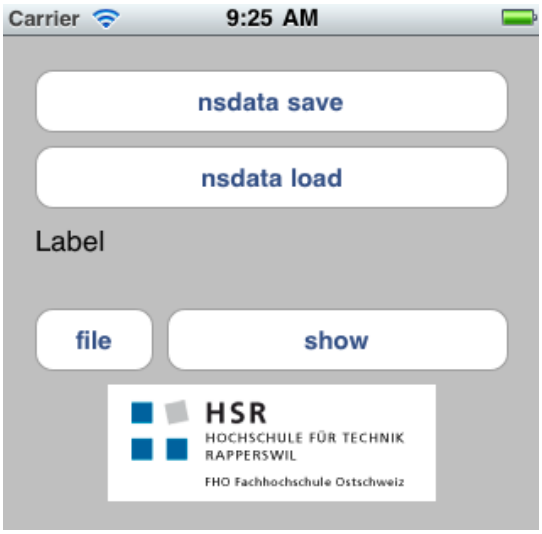

Abbildung 10: Demo NSData

In der Test-App SavaData wird ein Bild aus dem Internet in einem `NSData` Objekt gespeichert. Anschliessend wird das Bild auf dem Dateisystem abgelegt:

```
[data writeToFile:dataFilePath
atomically:YES];
```

Nun kann das Bild wieder geladen werden. Dazu wird die folgende Zeile ausgeführt:

```
NSData * newData = [NSData dataWithContentsOfFile:dataFilePath];
```

Nachdem das Bild in ein `UIImage` konvertiert wurde, kann es beliebig weiterverwendet werden. In unserer App wird das Bild in einer `UIImageView` dargestellt.

Referenzen

- http://developer.apple.com/library/mac/#documentation/Cocoa/Reference/Foundation/Classes/NSData_Class/Reference/Reference.html (Stand 11.6.11)

6.2.2.2 Sicheres Speichern von Credentials

Neben der normalen Datenspeicherung in iOS gibt es auch einen speziell gesicherten Bereich, die Key Chain. Sie dient zum sicheren Ablegen von Zugangsdaten, wie Passwörter oder Zertifikaten. Zugriff auf die Key Chain erhält man mit der richtigen Bundle Seed ID.

In der Test-App zur Key Chain wurde ein Wert abgelegt und anschliessend wieder aufgerufen. Dazu wurde das Beispiel “Simple Secure Storage in iOS” verwendet.

Wichtig:
Auf dem iPhone wird der Zugriff auf die Key Chain über das Provisioning-Profile geregelt. Dieses wird beim Signieren fest mit der App verbunden. Über verschiedene Programm-Versionen muss jeweils immer dasselbe Provisioning-Profile (ev. modifiziert) verwendet werden.

Nach dem Löschen einer Applikation bleiben die Passwörter erhalten. - Dieses Verhalten ist gewollt, da die Key Chain zwischen den Apps geteilt wird (shared Key Chain).

Referenzen

- http://www.manicgaming.com/2010/10/simple-secure-storage-in-ios/ (Stand 11.6.11)
- http://stackoverflow.com/questions/3671499/iphone-Key Chain-items-persist-after-application-uninstall (Stand 11.6.11)

6.2.3 Verschicken von Push Notifications

6.2.3.1 Verwenden der APNs-Bibliothek

Für die Verwendung der java-apns Bibliothek wurde eine Konsolenapplikation geschrieben. Damit die nötigen Funktionen zur Verfügung standen, musste die Datei apns-0.1.5-jar-with-dependencies.jar ins Projekt eingebunden werden.

Die Klasse ApnsService wird instanziiert und mit den nötigen Informationen versorgt (Pfad des Zertifikats, Token und Meldungen).

Für den erfolgreichen Betrieb müssen die TCP Ports 2195 und 2196 geöffnet sein.

6.2.3.2 iPhone für Notifications registrieren

Das Beziehen eines Device Token lässt sich in wenigen Zeilen realisieren und besteht aus zwei Teilen.

1. Nachdem die App aufgestartet ist, wird die Methode registerForRemoteNotifications aufgerufen.
2. Der zweite Block ist eine „Callback"-Methode, in der der Device Token als Parameter für die App zur Verfügung steht.

```
-(void) application: (UIApplication *)application
      didFinishLaunchingWithOptions:(NSDictionary *)options {
      // Register this app, on this device
      if (application.remoteNotificationsAreActive) {
            [application registerForRemoteNotifications];
      }
}

-(void)application:
      didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)token {
      // Use token here
      ...
}
```

6.2.4 Backgroundtasks

Um die Task-Completion auch zu testen, wurde auf ein bestehendes Projekt zurückgegriffen (Ursprung: siehe Referenzen). In dem Beispiel werden grosse Rechnungen simuliert. Um alles etwas zu vereinfachen, wurde das "Berechnen" durch Sleeps ersetzt:

```
[NSThread sleepUntilDate:[NSDate dateWithTimeIntervalSinceNow:1]];
NSLog(@"1");

[NSThread sleepUntilDate:[NSDate dateWithTimeIntervalSinceNow:2]];
NSLog(@"2");

[NSThread sleepUntilDate:[NSDate dateWithTimeIntervalSinceNow:27]];
NSLog(@"27");   // oder 4
```

Austesten und Ablauf der Ereignisse

- Beispiel mit 7 Sekunden warten
  - 1 2 4 werden angezeigt

  - Es wird kein stopCrunching gemeldet. Das macht Sinn, da das iOS die Ressourcen nicht entzogen hat.

- Grösserer Sleep mit 1, 2 & 27 Sekunden
  - 1, 2 werden angezeigt
  - Home gedrückt
  - die 27 erscheint ebenfalls

- Wieder mit 1, 2 & 27 Sekunden (Aufstarten einer anderen App)
  - 1, 2
  - zweite App läuft
  - Die Meldung 27 wird wieder einwandfrei dargestellt, trotz gestarteter Zweit-App.

Das stopCrunching konnte nicht provoziert werden. Das liegt wahrscheinlich daran, dass genügend Ressourcen auf dem Gerät vorhanden sind.

Sämtliche Tests wurden auf folgendem Gerät durchgeführt: iPod Touch, Software Version: 4. 2. 1

Referenzen

- Repository https://github.com/volonbolon/Task-Completion (Stand 11.6.11)
- Doku http://volonbolon.net/post/1080236607/more-multitasking-in-ios-task-completion (Stand 11.6.11)

6.2.5 Testumgebung zur iPhone App

Damit die App unabhängig von der bei Crealogix in Entwicklung stehenden Infrastruktur getestet werden konnte, wurde eine eigene Umgebung aufgezogen. Die Testumgebung besteht aus drei Servlets, die auf einem Tomcat 6.0 laufen.

SSL & Login

Der Server besitzt ein selfsigned SSL-Zertifikat, das das Verschlüsseln der Verbindung erlaubt. Der Zugriff auf alle Servlets ist geschützt. Über die gesicherte Verbindung können sich die User mittels HTTP Basic Authentication einloggen.

Das Einrichten von Basic Authentication ist in [BasicAuth] beschrieben.

Userverwaltung

Sobald auf eines der Servlets zugegriffen wird und ein User erforderlich ist, wird automatisch in einer Collection ein User erstellt. Der User wird jeweils aus dem Username des angemeldeten Users erstellt. Die erlaubten User sind in der Tomcat-Konfiguration abgelegt.

Impulse

Die Impulse befinden sich im Verzeichnis C:\CentradoContent\json und werden beim Start des Servers eingelesen.

Medien

Der Tomcat Server hostet ebenfalls den statischen Content. Dazu ist eine Ergänzung in server.xml (im Host-Abschnitt) nötig:

```
<Context path="/StaticContent"
    docBase="C:\CentradoContent\Public" debug="0" reloadable="true"/>
```

Die Inhalte sind dann unter http://152.96.56.36/StaticContent/... verfügbar. Eine Datei des Beispiel-Impuls' ist somit unter
http://152.96.56.36/StaticContent/Impulse/Nacken/img-k3-0001-1.png

verfügbar.

Die Medien selber sind im Verzeichnis C:\CentradoContent\Public\Impulse in ihrem jeweiligen Ordner abgelegt.

Persistenz
Der Server besitzt keine Persistenz. Die User werden nach jedem Start neu erstellt. Somit haben die User nach einem Neustart des Servers keine Impulse und kein DeviceToken mehr zugeordnet.

Die Schnittstelle und die einzelnen Servlets sind im Abschnitt 8.3 REST-Schnittstelle des Centrado Servers beschrieben.

Einrichten

- **Server-Port**
  Der Server-Port wurde auf 80 umgestellt. Diese Änderung betrifft server.xml .
  `<Connector connectionTimeout="20000" port="80" .....`
- **Deploying**
  Die APNs- und JSON-Bibliotheken müssen im Server-Verzeichnis abgelegt werden:
  apache-tomcat-6.0.18\lib

6.2.6 SSL-Verbindung auf dem Server einrichten

Die SSL-Verbindung wird mit einem Zertifikat gesichert, welches in einem Keystore abgelegt wird. Das Zertifikat ist selfsigned und wird von der iPhone-App nicht standardmässig akzeptiert.

Keystore und Zertifikat
Keystore mit einem Zertifikat erstellen.

```
keytool -genkey -alias tomcat -keyalg RSA -keystore C:\CentradoContent\keystore
```

Das Passwort für den Keystore und das Zertifikat „tomcat" müssen identisch sein, damit der Tomcat auch das Zertifikat lesen kann.

Keystore in Tomcat verwenden (SSL-Connector)
Beim Tomcat muss die Datei server.xml angepasst werden.

```
<!-- Define a blocking Java SSL Coyote HTTP/1.1 Connector on port 443 -->
<Connector port="443" protocol="org.apache.coyote.http11.Http11Protocol"
     SSLEnabled="true"     maxThreads="150"     scheme="https" secure="true"
     clientAuth="false"    sslProtocol="TLS"
     keystoreFile="C:\CentradoContent\keystore"
     keystorePass="centrado"       />
```

Der Standardpfad für den Keystore wäre C:\Dokumente und Einstellungen\[Benutzername]\.keystore . Wir legen ihn jedoch in C:\CentradoContent ab.

7 Externes Design

Dieses Kapitel beschreibt das Design der Benutzeroberfläche und die Navigation.

7.1 Paper-Prototype

Für die Erstellung des GUIs wurden einige Paper-Prototypen erstellt, die die beiden UseCases „Login“ (UC01) und „Impuls anzeigen“ (UC02) abdecken. Ein weiteres Feature, die Erinnerungsfunktion, wurde ebenfalls skizziert.

Die Paper-Prototypen hätten durchaus umgesetzt werden können. Es kam aber nicht dazu, da kurz nach der Vollendung der Paper-Prototypen die Design-Vorschläge von Crealogix kamen. Diese wurden dann, mit kleinen Änderungen, umgesetzt.

Die ausgearbeiteten Paper-Prototypen befinden sich im Ordner GUI.

7.2 GUI-Map

Die GUI-Map zeigt an, durch welche Screens der Benutzer der App navigieren kann. Sie gibt somit auch einen Überblick über die realisierten Features.

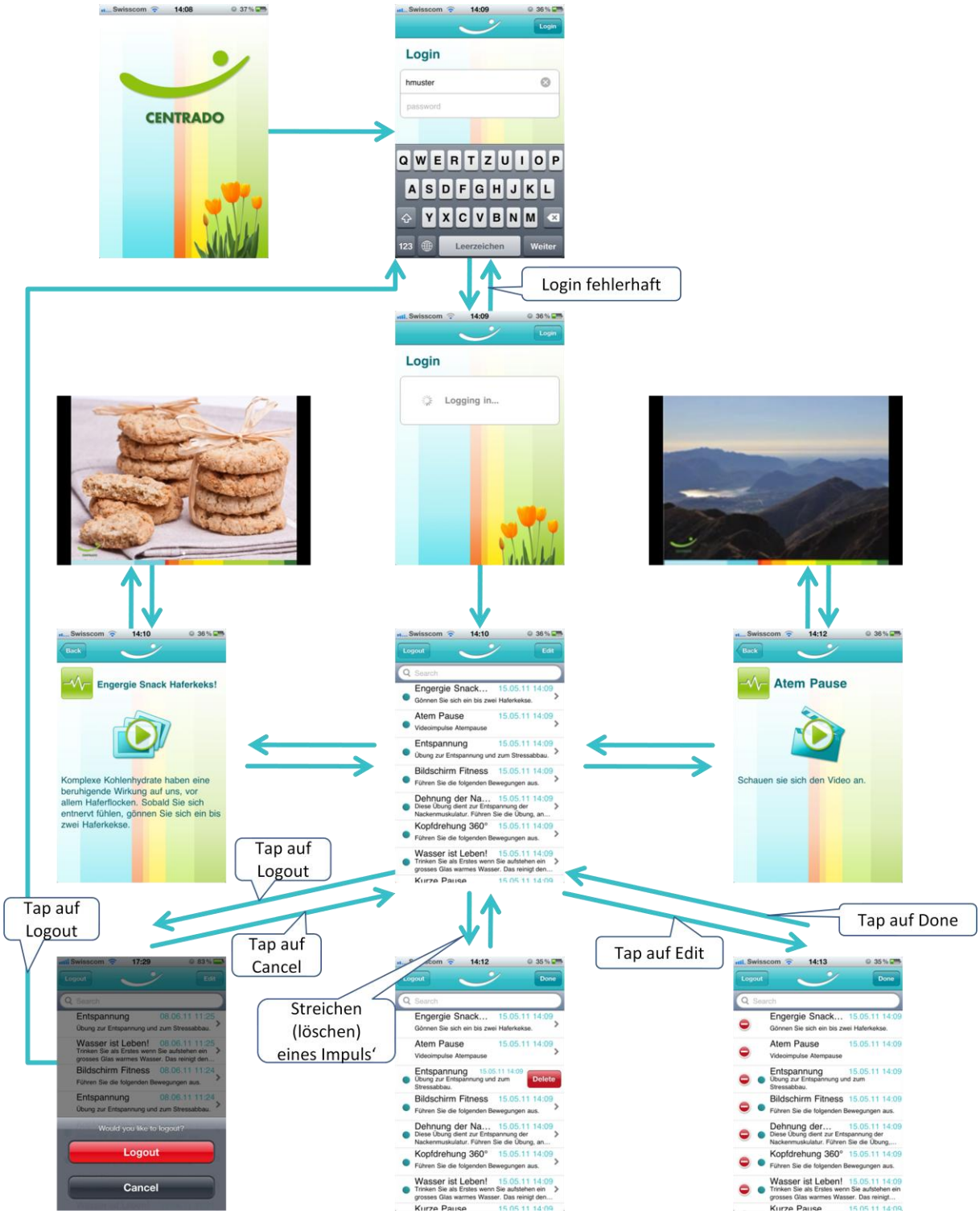

Abbildung 11: GUI-Map

8 Architektur und Design

In diesem Kapitel wird der Aufbau und die Funktionsweise der iPhone App und der REST-Schnittstelle vorgestellt. Dieses Kapitel erklärt ebenfalls, wie die unterschiedlichen Komponenten miteinander interagieren.

Für die komplexeren Abläufe, wie das Registrieren des Token oder das Auslösen eines Impuls', existieren Sequenzdiagramme, die den Verlauf der Nachrichten beschreiben.

8.1 Überblick

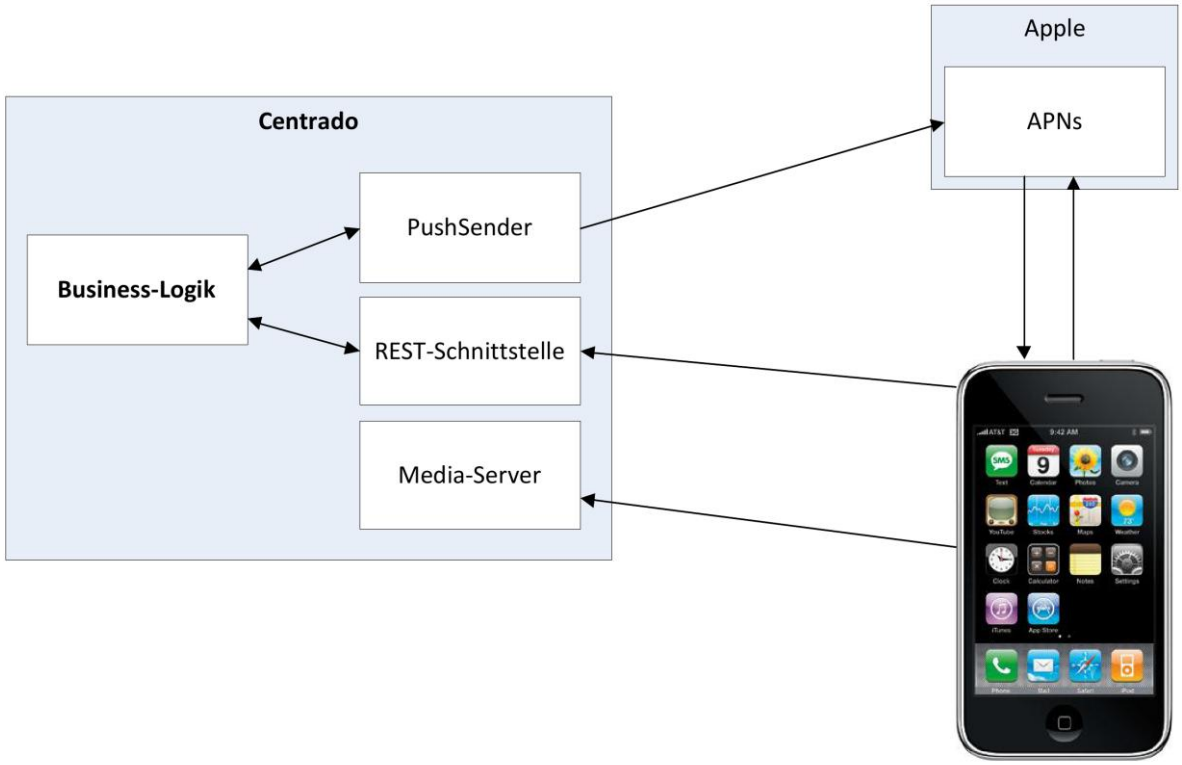

Abbildung 12: Architektur

Die Centrado iPhone App steht nicht für sich alleine, sondern arbeitet mit andern Komponenten zusammen. In Abbildung 12 sind diese dargestellt. Die Pfeile stehen für eine Interaktion der Komponenten, wobei die Pfeilrichtung angibt, wer die Kommunikation initiiert.

Beschreibung der Komponenten

Komponente	Beschreibung
iPhone App	Die Centrado App nimmt *Push Notifications* entgegen und zeigt Impulse an. Der Benutzer kann seine Login-Daten abspeichern und die empfangenen Impulse lokal verwalten.
APNs	Apple Push Notification service. Ein Service zum Verschicken von Push Notifications, der von Apple zur Verfügung gestellt wird. Für die Nutzung wird ein gültiges Client-Zertifikat benötigt.
Push Sender	Der Push Sender ist eine Java-Applikation, die sich mit dem Apple Push Notification Service (APNs) von Apple verbindet. Einmal verbunden, kann die Applikation beliebig viele Push' an die Centrado iPhone App auf verschiedenen Geräten verschicken.

REST-Schnittstelle	Die Serverkomponente, mit der die iPhone App interagiert. Die Verbindungen sind verschlüsselt und der Zugriff ist über Basic Authentication gesichert.
Media-Server	Der Media-Server stellt sämtliche Medien für die App zur Verfügung. Dies sind vor allem Bilder und Videos. Um Inhalte zu beziehen, benötigt man keine Authentifizierung.
Business-Logik	Die Business-Logik ist die zentrale Intelligenz des Systems. Hier wird über das Absenden der Push' entschieden und die Geräte-Token gespeichert. Die Business-Logik verwendet eine Datenbank (nicht gezeichnet) zum Ablegen der Daten.

8.2 Interaktion der Komponenten

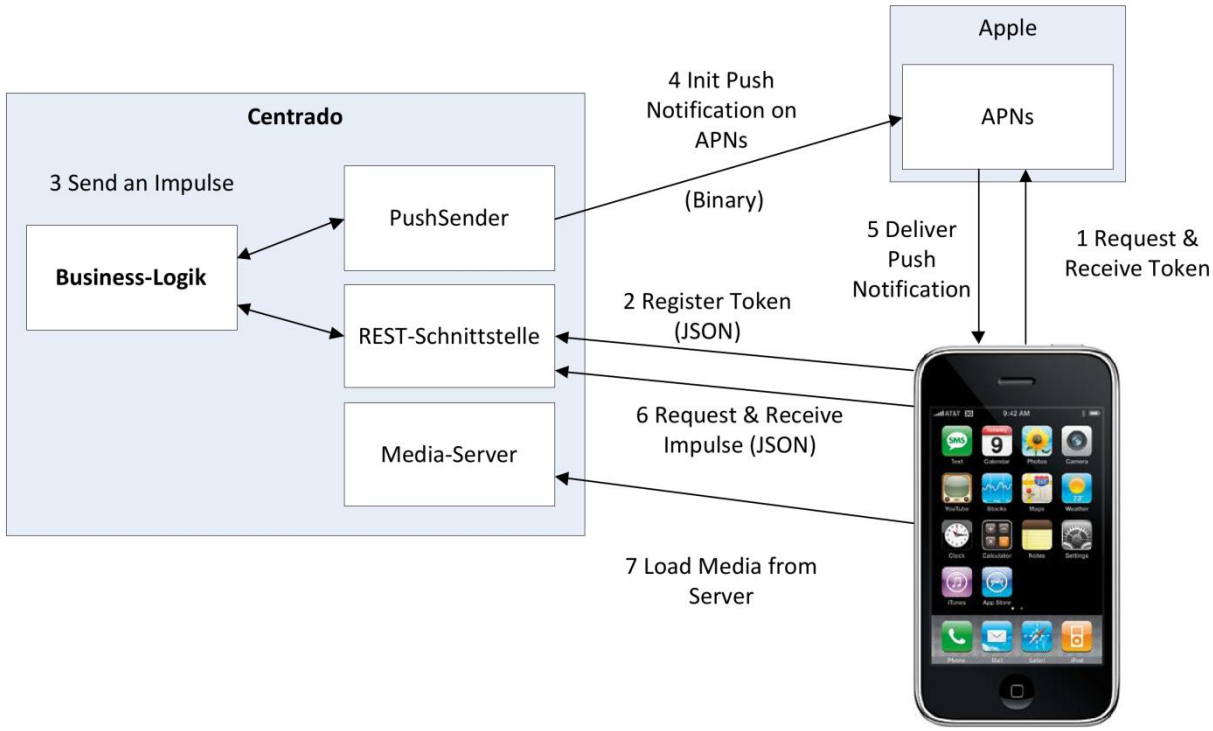

Abbildung 13: Interaktion der Komponenten

Sobald die App gestartet wurde, fordert sie einen Device Token beim APNs an (1). Der Token wird nach dem Login an die REST-Schnittstelle gesendet und auf dem Server abgelegt (2). Das System ist nun bereit für das Versenden von Impulsen.

Die Business-Logik entscheidet nun, wann und welchen Impuls sie an einen Benutzer verschickt (3). Dazu verbindet sich der PushSender mit dem APNs und löst dort den Push aus (4). Anschliessend wird der Push auf das Gerät verschickt (5). Auf dem iPhone wird die Push Notification angezeigt. Nur wenn der User auf View / Anzeigen tippt, wird die Push Notification der App zugestellt. Die App lädt den Impuls über die REST-Schnittstelle (6). Wenn der Impuls angezeigt wird, werden die benötigten Medien vom Media-Server geladen (7).

8.2.1 Registrierung des Device Tokens

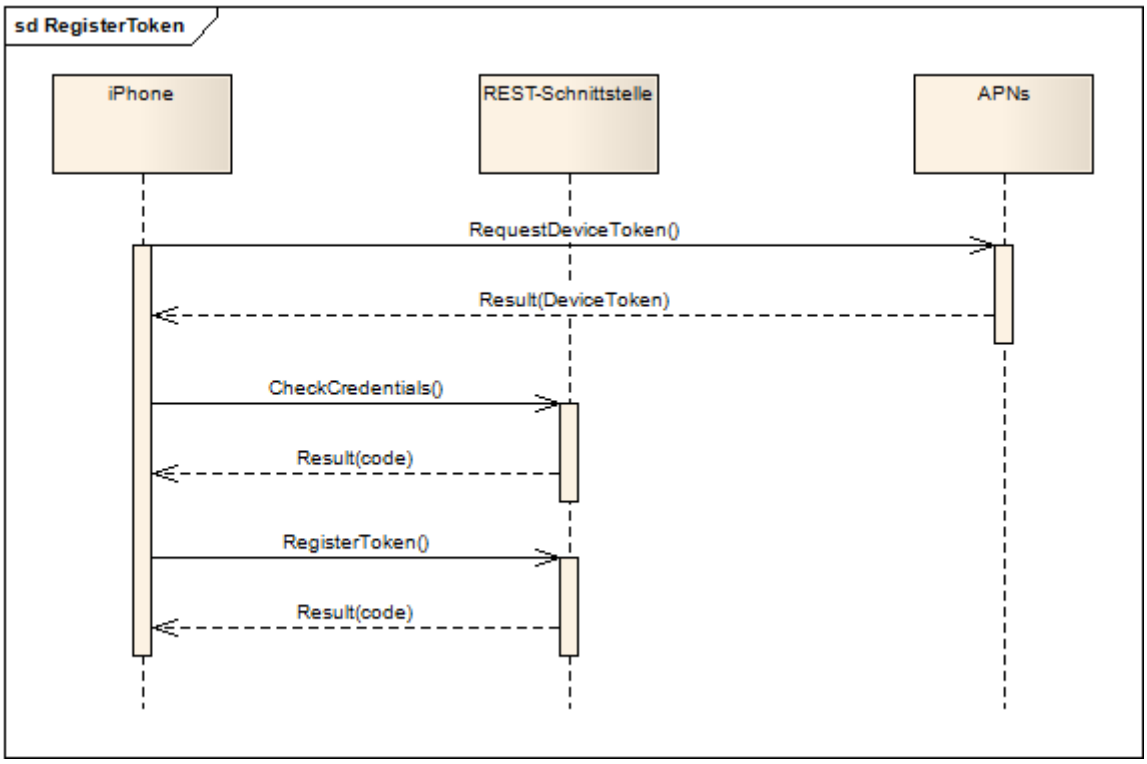

Abbildung 14: Device Token anfordern und registrieren

Damit das iPhone vom APNs erreicht werden kann, muss der Device Token bekannt sein. Dazu fordert das Gerät beim APNs einen Token an. Sobald das iPhone den Device Token erhalten und sich erfolgreich eingeloggt hat, wird der Token auf der REST-Schnittstelle abgelegt.

8.2.2 Auslösen eines Impuls‘

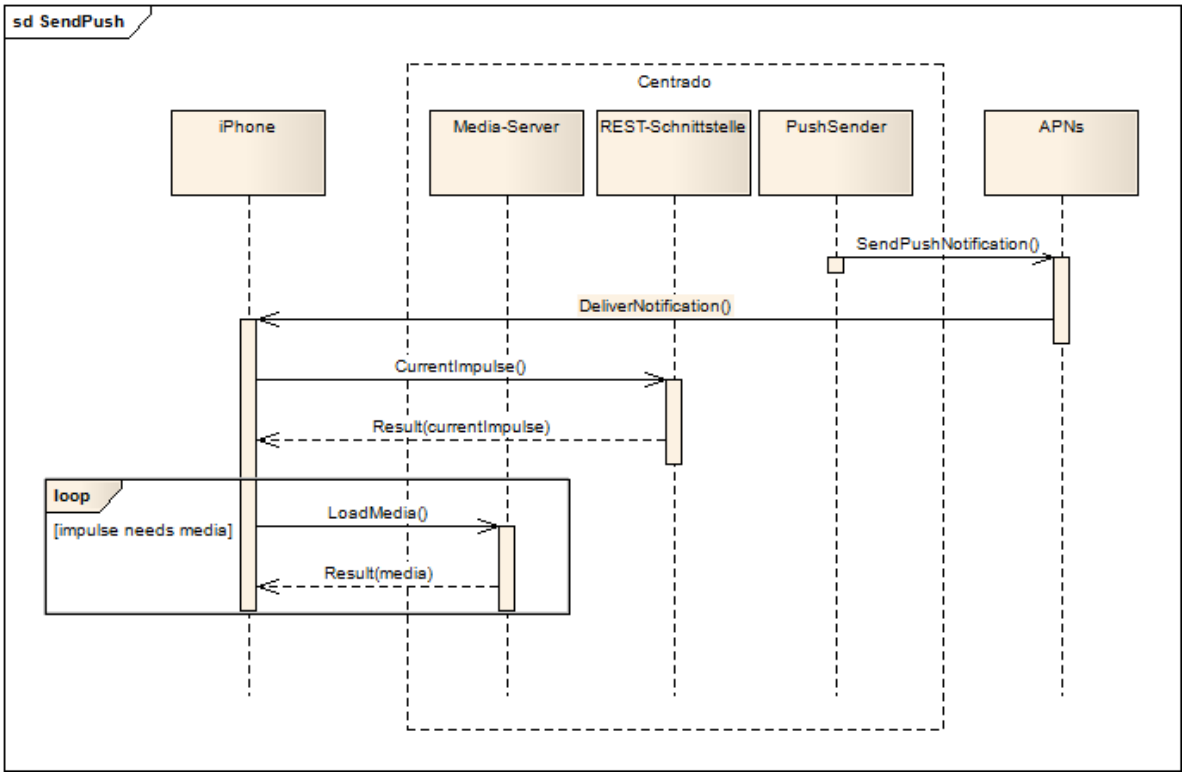

Abbildung 15 : Auslösen eines Impuls'

Impulse werden im Produktivsystem durch eine Engine ausgelöst, im Testsystem durch den Aufruf einer URLs.

Der Push Sender verbindet sich mit dem APNs und löst eine Push Notification aus. Diese wird dann an das Gerät geliefert. Die App wird durch die Push Notification aktiviert und lädt die Impulse von der REST-Schnittstelle. Beim Anzeigen des Impulses werden alle Medien vom Media-Server geladen und anschliessend dargestellt.

8.3 REST-Schnittstelle des Centrado Servers

Die Schnittstelle zwischen dem Server und dem iPhone wird RESTFul implementiert (siehe [RESTFul]). Anfragen gehen immer vom iPhone an die Schnittstelle. Soll eine Kommunikation vom Centrado Server zum iPhone hergestellt werden, so wird eine Push Notification über den APNs zum iPhone gesendet.
Auf Grund dieser Push Notification kann das iPhone dann die Kommunikation mit dem Server aufnehmen.

Die verschiedenen Services sind unter unterschiedlichen Pfaden erreichbar. Daten werden im JSON-Format ausgetauscht. Angaben über Verschlüsselung und Login siehe 6.2.5 Testumgebung zur iPhone App.

8.3.1 CurrentImpulse

https://<host>/<server>/CurrentImpulse

Der Service liefert einen Array von Impulsen im JSON-Format. Es liefert die Impulse für den angemeldeten Benutzer. Die Sprache für die Impulse wird aus den Benutzereinstellungen auf dem Server ermittelt.

Request Methode	GET
Parameter	keine
Body	keiner
Response Header	200 OK
Response Body	Impulse als JSON-Dokument. Der String ist als UTF-8 codiert.

Aufbau des Response Body
Auf der Root-Ebene besteht das JSON-Dokument aus einem Objekt, das einen Member namens `itmes` hat. Dieses enthält einen Array von Impulse Objekten. Das Impulse Objekt ist in der folgenden Tabelle beschrieben.

Member	Value	Beschreibung
single	`true` oder `false`	`true` bedeutet, dass es sich um einen Spontanimpuls handelt. `false` bedeutet, dass es ein Programmimpuls ist.
soundURL	`null`	-- wird nicht mehr verwendet.
imageURLs	*`array`*	Ein Array von Strings. Diese enthalten die URLs der Bilder, die zu diesem Impuls dargestellt werden sollen. Die Reihenfolge im Array bestimmt die Reihenfolge, in der die Bilder dargestellt werden sollen.
timestamp	*`number`*	Zeitpunkt, zu dem der Impuls dem Benutzer zugeordnet wurde. Die Zahl beschreibt den Zeitpunkt in Unixzeit. Siehe auch [Unixzeit].
duration	*`number`*	Anzahl Minuten, die für die Ausführung des Impulses benötigt werden.
title	*`string`*	Titel des Impulses.

description	*string*	Beschreibung des Impulses. Ein ausführlicher Text, der angibt, was zu diesem Impuls gemacht werden soll.
webURL	*string* oder null	Eine URL zu einer Webseite, die weiterführende Informationen zum Impuls enthält.
action	*string*	Bezeichnet, wie der Impuls konsumiert werden soll. Normalerweise sind dies „Lesen“ oder „Ausführen“
language	*string*	Code der Sprache, in der die Inhalte der Felder title, description, descriptionShort und action verfasst sind.
descriptionShort	*string*	Eine Zusammenfassung des Feldes description.
workingID	*string*	ID, die den Impuls identifiziert.
videoURL	*string* oder null	Eine URL, die auf ein Video verweist, das zu diesem Impuls angezeigt werden soll.

Alle URLs müssen absolut definiert werden, da das JSON-Dokument selbst nicht als Kontext für relative URLs verwendet werden kann.

Abbildung 16: Visualisiertes JSON-Dokument mit zwei Impulsen

Verhalten bei Fehlern
Fehler werden über die HTTP-Statuscodes mitgeteilt. Wenn vorhanden, soll der definierte HTTP-Statuscode verwendet werden (siehe [Statuscode]). Ein Fehler, der vom Test-Server simuliert werden kann, ist 500, internal error. Weitere mögliche Fehler sind:

- Benutzername/Passwort falsch (401)
- Login ungültig (403)
- Serverfehler (5xx)

8.3.2 RegisterToken

https://<host>/<server>/RegisterToken

Mit dem Service RegisterToken wird der Device Token des iPhones unter dem angemeldeten Benutzer abgelegt. Pro Benutzer kann ein Token abgelegt werden. Ein bestehender Token wird jeweils überschrieben. Falls sich auf einem iPhone zuerst Benutzer A und danach Benutzer B einloggt, wird der Device Token T zuerst für den Benutzer A und dann für den Benutzer B registriert. Der Server ist dafür verantwortlich, dass T zu jedem Zeitpunkt nur einem Benutzer zugeordnet ist. Andernfalls würde der Benutzer B auch Push Notifications erhalten, die für Benutzer A sind.

Vorsicht:

- Wird auf dem iPhone ein Benutzerwechsel durchgeführt, so ist der Token auf dem Test-Server unter beiden Benutzern abgelegt (Abgabezustand Bachelorarbeit).
- Da Push Notifications ein Teil von iOS sind, wird die App auch gestartet, falls der Benutzer sich bewusst ausgeloggt hat und nicht mehr mit der App arbeiten will. Mit einem „unregister“ beim Logout, könnte das Problem behoben werden.

Request Methode	POST
Parameter	Keine
Request Body	Device Token und Gerätename als JSON-Dokument. Der String ist als UTF-8 codiert.
Response Body	Leer
Response Header	200 oder 206 (No Content)

Aufbau des Request Body
Der Request Body besteht aus einem JSON Objekt.

Member	Value	Beschreibung
DeviceToken	*string*	Die Daten des Device Token als Hex-String
DeviceName	*string*	Bezeichnung des iPhones zu dem der `DeviceToken` gehört.

Bedient wird der ganze Server über das Servlet SendPush.

8.3.3 SendPush

https://<host>/<server>/SendPush

Der Service SendPush wird auf dem Test-Server zur Steuerung des Verhaltens benutzt. Er zeigt die aktuellen Benutzer an und ob sie einen Device Token gespeichert haben. In der selben Ansicht werden auch die verfügbaren Impulse gezeigt.

```
Users:
impCnt Name   Token
 2     ck     650986aa8de.....b64ce4c604a53cb683e3a (IFS iPhone)
 3     ae     f7914558796.....47c10234a5fdf8bfd373e (iPhone AE)
 9     p7r    0baeff1ccda.....39a233f25cba191aafc6e (PS's iPhone)

Impulses:
 id           pushMessage
 text         Kurze Pause (10min)
 bildschirm   Bildschirm Fitness (15min)
 marketing    Entspannung (1min)
 wasser       Wasser ist Leben! (1min)
 kopfdrehung  Kopfdrehung 360° (3min)
 kecks        Engergie Snack Haferkeks! (1min)
 pic          Dehnung der Nackenmuskeln (3min)
 vid          Atem Pause (5min)
```

Server-Befehle

Impuls an einen Benutzer schicken

https://<host>/<server>/SendPush?target=ae&id=vid

Parameter

target: Benutzername

id: ID des Impulses

Alle Impulse dem angemeldeten Benutzer anfügen:

https://<host>/<server>/SendPush?fill

Alle Impulse des angemeldeten Benutzers löschen (Impulse bleiben auf dem iPhone gespeichert):

https://<host>/<server>/SendPush?clear

Wahlweise kann auch ein Serverfehler simuliert werden:

https://<host>/<server>/SendPush?error

Der Server zeigt dann die folgende Meldung:

`SERVER SIMULATES ERROR IN CurrentImpulse`

Device Token des angemeldeten Benutzers setzen:

https://<host>/<server>/SendPush?token=abcdef

Parameter

token: Device Token als Hex-String

8.4 Aufteilung in logische Layer

Die Centrado iPhone App ist in logische Layer aufgeteilt. Abbildung 17 gibt eine Übersicht über die Layer. Abbildung 18 zeigt die Layer mit ihren Abhängigkeiten und Abbildung 19 zeigt die Objekte innerhalb der Layer und die Abhängigkeiten.

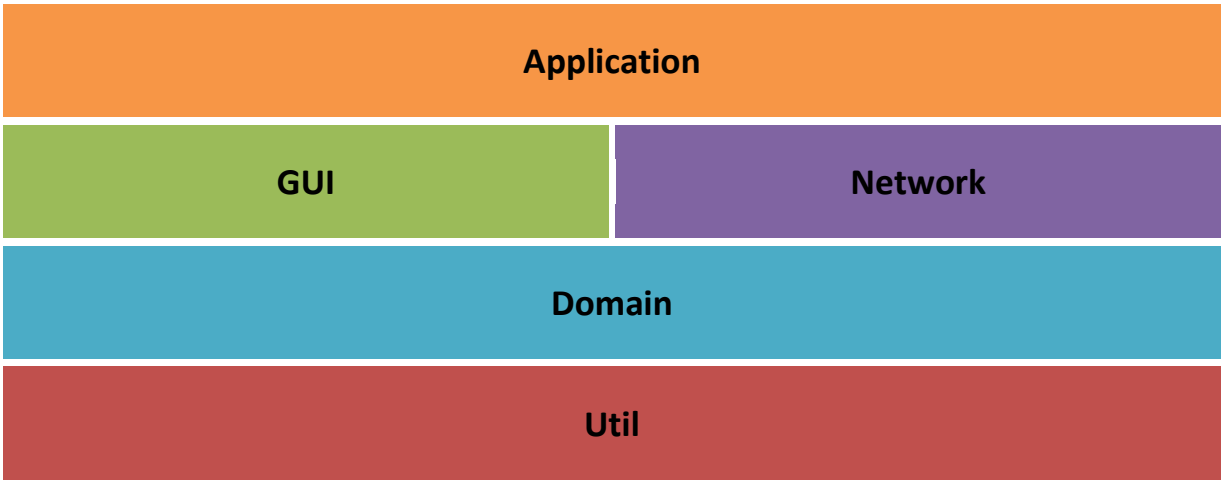

Abbildung 17: Aufteilung der iPhone App in Layer

Die Layer haben folgende Aufgaben:

Layer	Aufgaben
Application	Die Klassen des Application Layers steuern den Ablauf der Applikation.
GUI	Die Klassen des GUI Layers sind für die Darstellung und Benutzerinteraktion verantwortlich.
Network	Die Klassen des Network Layers sind für die Kommunikation mit der REST-Schnittstelle verantwortlich.
Domain	Im Domain Layer befinden sich Klassen, die die Domain Konzepte repräsentieren. Der Domain Layer ist ebenfalls für die Persistierung der Objekte verantwortlich
Util	Der Util Layer beinhaltet Klassen, die allgemeine Basisfunktionalität zur Verfügung stellen.

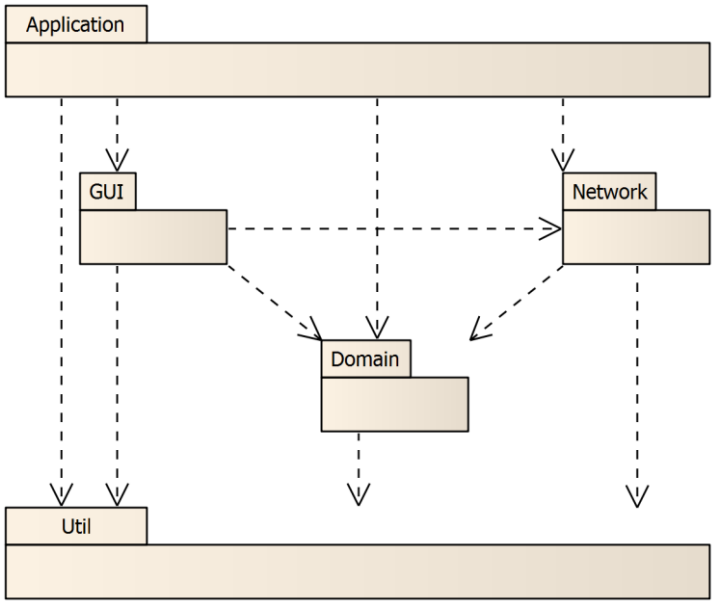

Abbildung 18: Layer mit ihren Abhängigkeiten

Der Aufbau entspricht einem Loose Layering. Somit ist es z.B. den Klassen des Application Layers gestattet, direkt auf Klassen des Domain oder Util Layers zuzugreifen.

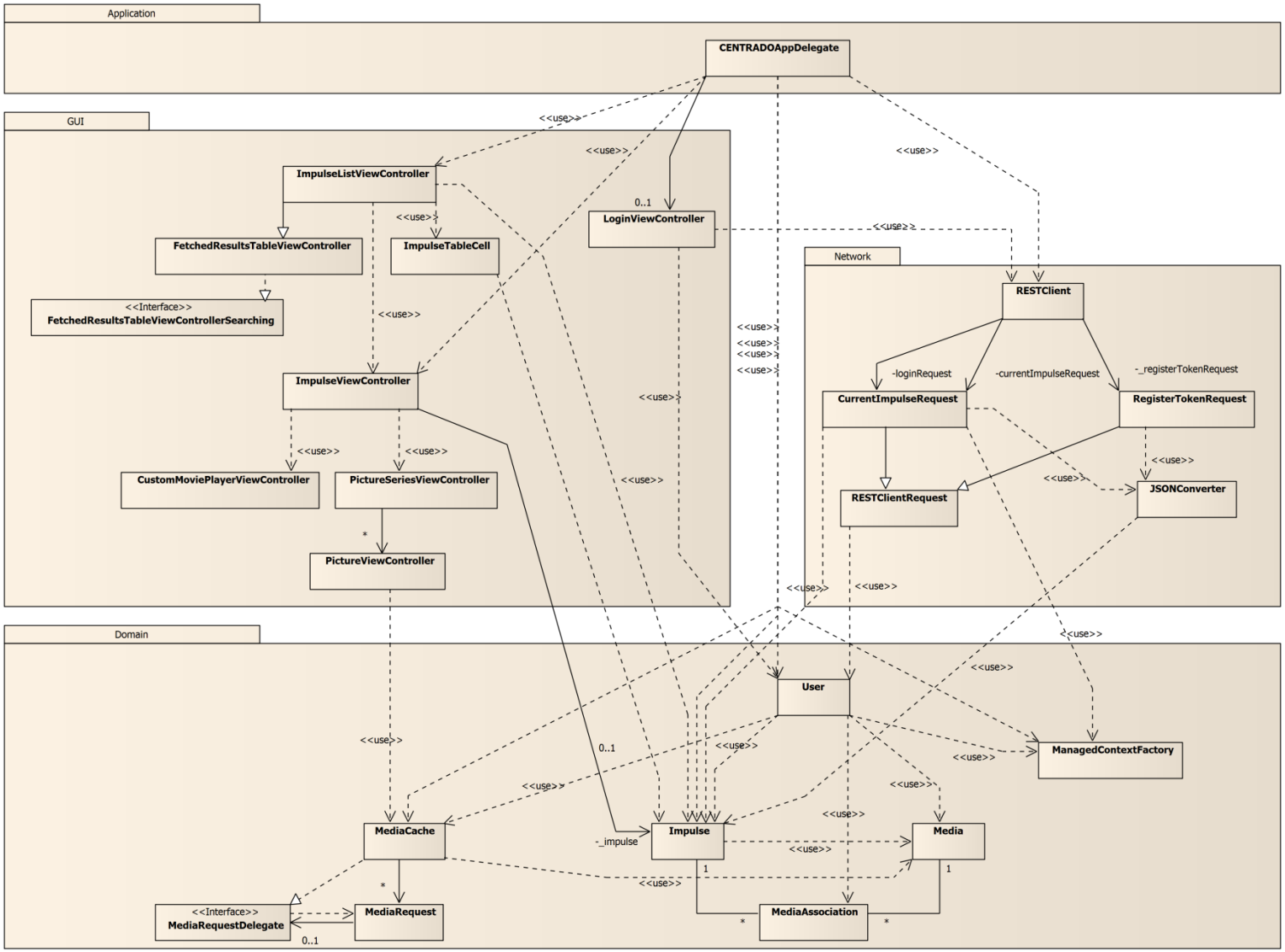

Abbildung 19: Klassen in Layern mit Abhängigkeiten (zur besseren Übersichtlichkeit wurde der Util Layer weggelassen)

8.5 Application Layer

Der Application Layer enthält nur die Klasse CENTRADOAppDelegate. Diese wird vom Cocoa Framework aufgerufen, wenn Ereignisse auftreten, wie z.B. „Starten der App abgeschlossen“, wenn die App in den Hintergrund wechselt oder eine Push Notification empfangen wurde.

8.5.1 Abweichungen und Verbesserungsmöglichkeiten

In Abbildung 19 sieht man die Abhängigkeiten der Klassen. Dabei fällt auf, dass die Klasse `LoginViewController` aus dem GUI Layer die Klasse `RESTClient` des Network Layers benutzt. Somit ist GUI von Network abhängig. Dies ist der Fall, weil die View Controller in GUI nicht nur für die Darstellung und Interaktion zuständig sind, sondern auch den Ablauf der Applikation beeinflussen. Dies könnte mit Task-Controllern behoben werden.

Konzept der Task Controller

Task Controller sind Klassen, die die Ablaufsteuerung der Applikation für bestimmte Tasks übernehmen. Sie gehören somit zum Application Layer. Ein Task Controller wird durch ein bestimmtes Ereignis ausgelöst. Dabei wird eine neue Instanz des Task Controllers erzeugt. Für den Login könnte eine Klasse `LoginController` definiert werden. Diese würde als erstes den `LoginViewController` benutzen, um die Login View darzustellen und sich für das Ereignis registrieren, dass der Loginvorgang ausgelöst wurde. Der Loginvorgang würde daher nicht mehr vom `LoginViewController` ausgeführt, sondern vom `LoginController`. Dieser würde dann den `RESTClient` für den Login auf dem Server benutzen. Nach erfolgreichem Login würde der `LoginController` mit dem `LoginViewController` die Login View wieder ausblenden.

Ein Task Controller selbst könnte wiederum von einem andern Task Controller benutzt werden. Es könnte so ein `PushNotificationController` erstellt werden, der den Ablauf kontrolliert, wenn eine Push Notification empfangen wurde. Wenn der Benutzer zum Zeitpunkt des Empfangs der Push Notification nicht eingeloggt ist, muss sich dieser noch einloggen. Dazu könnte der `PushNotificationContoller` den `LoginController` aufrufen. Dieser übernimmt dann die Kontrolle für den Login-Vorgang. Wenn der Task des `LoginControllers` beendet ist, löst er einen Callback aus. Mit diesem wird der `PushNotificationController` benachrichtig, so dass er die Kontrolle wieder übernehmen und mit Hilfe des `ImpulseViewControllers` den neusten `Impulse` darstellen kann.

Durch den Einsatz von Task Controllern würden sich Abhängigkeiten vermeiden lassen. Speziell die View Controller würden einfacher wiederverwendbar. Folgende Abhängigkeiten liessen sich mit Task Controllern eliminieren.

Abhängigkeit	Lösung
LoginViewController -> RESTClient	Der `LoginController` könnte auf Callbacks des `LoginViewControllers` (GUI) reagieren und die Aktionen auf dem `RESTClient` auslösen.
User -> MediaCache	Der Benutzerwechsel könnte von einem `SwitchUserController` übernommen werden. Dieser könnte das Leeren des `MediaCache` und das Löschen der `Impulse` des vorhergehenden Benutzers übernehmen. Aktuell wird dies alles in der Methode `switchUser:withPassword:` der Klasse `User` gemacht.

MediaCache -> Media	Die Methode `cleanUpUnusedMediaObjectsUsingContext:` löscht nicht mehr verwendete Bilder aus dem Cache. Ein `CacheCleanupController` könnte überprüfen, ob die `Media` Objekte noch einem `Impulse` zugeordnet sind und dann den `MediaCache` aufrufen, um nicht mehr verwendete Bilder aus dem Cache zu entfernen.

8.6 GUI Layer

Im GUI Layer befinden sich Klassen, die für die Darstellung von Informationen und die Benutzerinteraktion zuständig sind. In diesem Abschnitt werden die Klassen kurz beschrieben. Die Klassen `FetchedResultsTableViewController` und `PictureSeriesViewController` werden weiter unten noch detailliert beschrieben.

Klasse / Interface	Beschreibung
CustomMoviePlayerViewController	Ein View Controller, der von `MPMoviePlayerViewController` erbt und diesen so einschränkt, dass er Videos nur im Querformat anzeigt.
FetchedResultsTableViewController	Ein `UITableViewController`, der als Basisklasse für View Controller verwendet werden kann, die Mengen von `NSManagedObject` Objekten als Tabelle darstellen.
FetchedResultsTableView-ControllerSearching	Ein Protokoll, das eine optionale Methode für das Suchen in einem `FetchedResultsTableViewController` definiert.
ImpulseListViewController	Ein View Controller für die Darstellung aller Impulse, die nicht gelöscht sind.
ImpulseTableCell	Eine `UITableViewCell` ist für die Darstellung von Impulsen. Mit Hilfe dieser Klasse können die Zellen für die Darstellung von Impulsen in einer `UITableView` im *Interface Builder* gestaltet werden.
ImpulseViewController	Ein View Controller für die Darstellung eines Impulses.
LoginViewController	Ein View Controller für die Darstellung des Login-Screen.
PictureSeriesViewController	Ein View Controller für die Darstellung von Bildserien. Er stellt immer ein Bild auf einmal dar und ermöglicht das Vor- und Zurückblättern.
PictureViewController	Ein View Controller für die Darstellung eines Bildes, das von einem Server geladen wird.

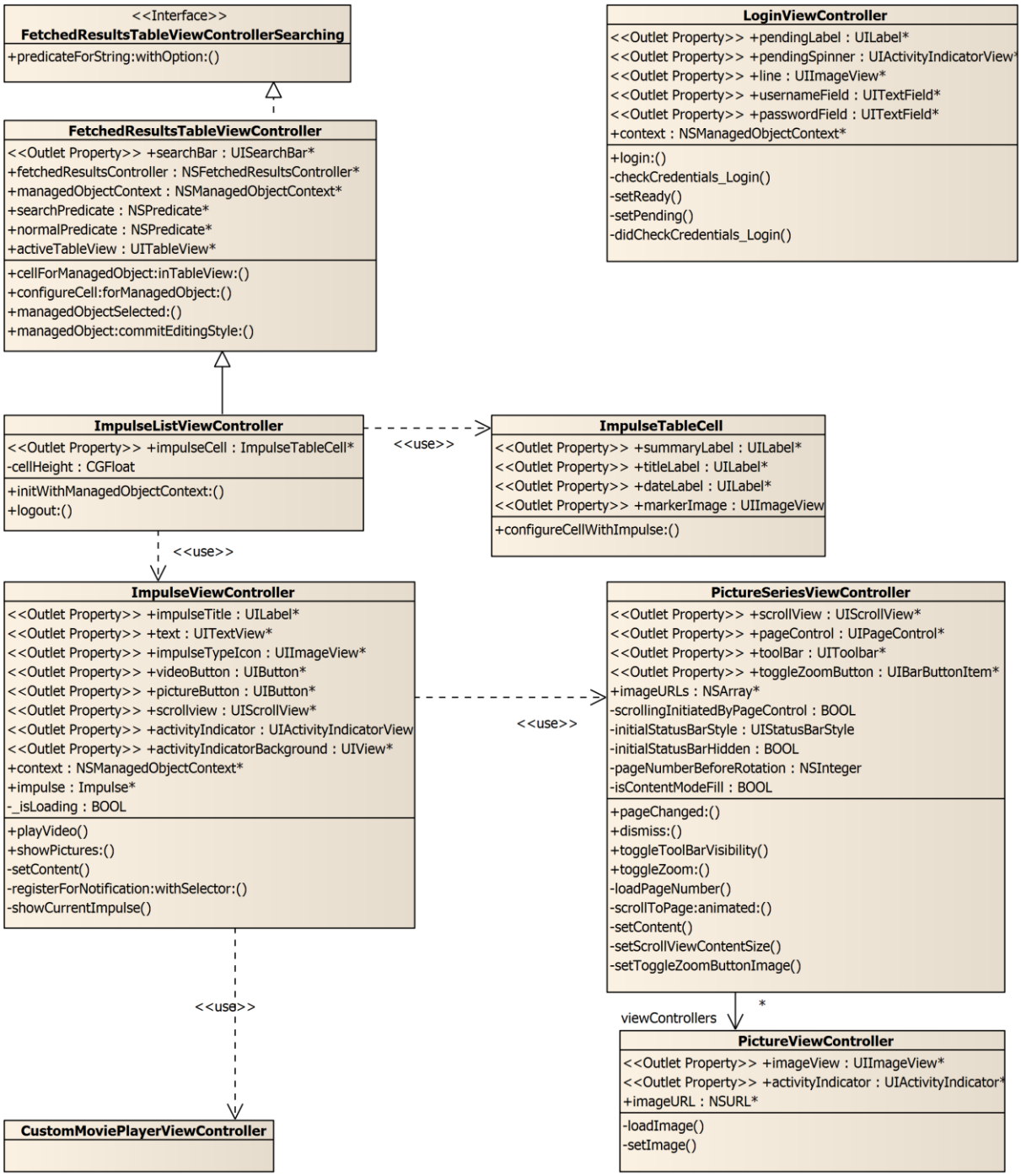

Abbildung 20: Klassen des GUI Layers

8.6.1 FetchedResultsTableViewController

Die Klasse `FetchedResultsTableViewController` ist eine Basisklasse, um Tabellen mit `NSManagedObjects` darzustellen. Sie bietet neben der Tabellendarstellung von `NSManagedObjects` auch die Möglichkeit in den dargestellten Resultaten zu suchen.

8.6.1.1 Verwendung

Um den `FetchedResultsTableViewController` zu verwenden, wird von diesem geerbt.

Initializer und Initialisierung

Der Initializer ist `initWithStyle:(UITableViewStyle)style`. Der Parameter style bestimmt, ob die Tabelle normal oder mit abgerundeten Ecken dargestellt werden soll. Um den

`FetchedResultsTableViewController` zu verwenden, müssen folgende Properties initialisiert werden:

Property	Beschreibung
fetchedResultsController	Ein `NSFetchedResultsController`, der die Definition der Suche für die Grundmenge enthält. Auf diesem wird auch die Sortierreihenfolge definiert.
managedObjectContext	Ein `NSManagedObjectContext,` um auf die Core Data Objekte zuzugreifen.
navigationItem.rightBarButtonItem	Wenn diesem Property ein Objekt zugewiesen wird, kann die Tabelle bearbeitet werden. Es ist dann möglich, Elemente aus der Tabelle zu löschen.
searchBar	Eine `UISearchBar`, die verwendet wird, um in den Resultaten zu suchen. Das Property ist als *Outlet* markiert und kann somit auch im *Interface Builder* definiert werden. Wenn keine searchBar definiert wurde, wird - sofern benötigt - beim Laden eine Standard `UISearchBar` erzeugt.

Methoden überschreiben
Die folgenden Methoden können überschrieben werden, um das Verhalten und die Darstellung anzupassen.

-(void)configureCell:(UITableViewCell *)cell forManagedObject:(NSManagedObject *)managedObject
Diese Methode wird aufgerufen, wenn eine Zelle ein ManagedObject darstellen soll. Sie sollte immer überschrieben werden, da die Basisimplementierung den Zelltitel mit der `description` des `NSManagedObjects` initialisiert. Die Informationen in `description` sind jedoch nur für Entwickler gedacht.

-(void)managedObjectSelected:(NSManagedObject*) managedObject
In dieser Methode kann auf das Selektieren einer Zelle reagiert werden. Als Parameter wird das `NSManagedObject` übergeben, das in dieser Zelle dargestellt wird.

-(void)managedObject:(NSManagedObject *)managedObject commitEditingStyle:(UITableViewCellEditingStyle)editingStyle
In dieser Methode kann auf das Bearbeiten einer Zelle reagiert werden. Dies kann das Löschen, aber auch das Verschieben innerhalb der Tabelle sein. Verschieben ist jedoch nur möglich, wenn bei der Initialisierung des `fetchedResultsControllers` keine Sortierung angegeben wurde.

-(UITableViewCell*) cellForManagedObject:(NSManagedObject *)managedObject inTableView:(UITableView *)tableView
Diese Methode wird aufgerufen, wenn die Table View eine neue Zelle für die Darstellung eines Managed Object benötigt. Durch Überschreiben dieser Methode kann man eigene `UITableViewCells` verwenden. Mit Hilfe des Parameters `managedObject`, der angibt, welches Objekt in dieser Zelle dargestellt werden soll, können je nach Inhalt auch unterschiedliche Table View Cells zurückgegeben werden. Der Parameter `tableView` gibt an, für welche Tabelle die Zelle bestimmt ist. Er ermöglicht auch, den *Cell reuse* Mechanismus der `UITableView` zu verwenden. Dieser sammelt Zell-Objekte, die sich nicht mehr im sichtbaren Teil der Tabelle befinden ein und legt sie in einem Pool ab, aus dem sie

wieder geholt werden können. Dadurch existieren nur ein paar Zellen mehr als aktuell dargestellt werden und es müssen nicht immer neue Objekte erzeugt und wieder zerstört werden.

-(CGFloat)tableView:(UITableView *)tableView
heightForRowAtIndexPath:(NSIndexPath *)indexPath
Wenn eigene Table View Cells verwendet werden, sollte diese Methode auch überschrieben werden. Sie definiert die Höhe einer Zelle. Wenn alle Zellen vom selben Typ sind oder die gleiche Höhe haben, kann hier ein konstanter Wert zurückgegeben werden. Die Methode sollte keine aufwändigen Operationen durchführen. Dies ist auch der Grund, dass hier der `indexPath` und nicht das ManagedObject, das zu diesem `indexPath` gehört, übergeben wird.

-(NSPredicate*)predicateForString:(NSString*)searchString
withOption:(NSInteger)option
Die Methode `predicateForString:withOption:` wird aufgerufen, wenn der Benutzer Eingaben im Suchfeld der Tabelle macht. Der Parameter `searchString` enthält den eingegebenen Text. Der Parameter `option` enthält den Index der gewählten Option. Als Resultat wird ein `NSPredicate` erwartet. Dieses wird anschliessend mit dem Prädikat, mit dem der FetchedResultsController bei der Initialisierung ausgestattet wurde, UND-verknüpft.

Wenn die ableitende Klasse diese Methode nicht implementiert, wird keine SearchBar in der Tabelle angezeigt.

Abbildung 21: SearchBar mit Optionen

8.6.1.2 Funktionsweise

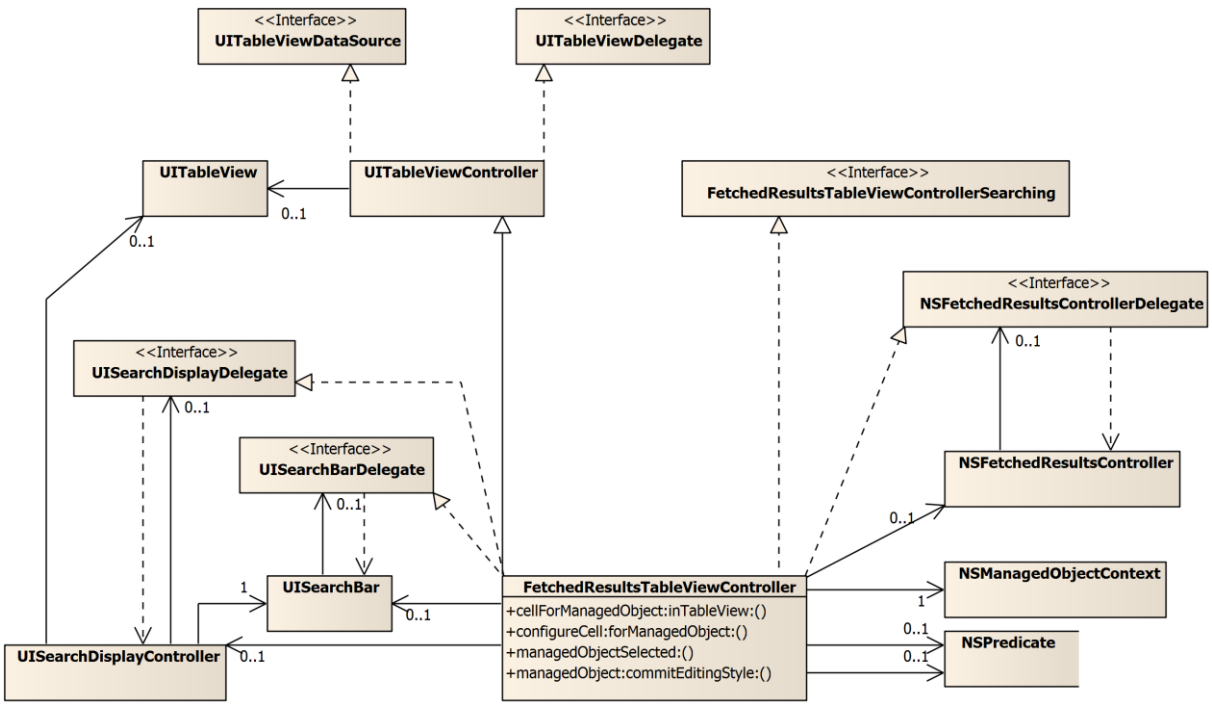

Abbildung 22: FetchedResultsTableViewController mit Beziehungen

In Abbildung 22 ist der FetchedResultsTableViewController mit den Beziehungen zu den wichtigsten Klassen, die er verwendet, dargestellt. Auf der linken Seite befinden sich die Klassen für die Darstellung der Tabelle und der Suche. Auf der rechten Seite befinden sich die Klassen, die für Core Data benötigt werden. Stark vereinfacht gesagt ist der FetchedResultsTableViewController ein UITableViewController, der ein NSFetchedResultsController als Datenquelle verwendet und eine vereinfachte Handhabung des UISearchDisplayControllers anbietet.

UITableViewDataSource und UITableViewDelegate
Der FetchedResultsTableViewController erbt von UITableViewController. Dadurch implementiert er auch die Protokolle UITableViewDataSource und UITableViewDelegate, welche für die Verarbeitung der Ereignisse verantwortlich sind, die eine UITableView auslöst. Die Aufrufe der Methoden des UITableViewDataSource Protokolls werden mit Hilfe des NSFetchedResultsController beantwortet. Neben dem Bereitstellen der Daten, die dargestellt werden sollen, gehört das Erzeugen von Zellen ebenfalls zum Umfang des UITableViewDataSource Protokolls. Der Aufruf tableView:cellForRowAtIndexPath: wird mit den Aufrufen von cellForManagedObject:inTableView: und configureCell:forManagedObject: bearbeitet. Beide Methode können in abgeleiteten Klassen einzeln überschrieben werden.

Für die Ereignisse des UITableViewDelegates wird der indexPath der Zelle in das Managed Object übersetzt, das in dieser Zelle dargestellt wird. Dieses wird dann an Methoden übergeben, die von den erbenden Klassen überschrieben werden können. Es sind die Methoden managedObjectSelected: und managedObject:commitEditingStyle:.

Suchen
Der FetchedResultsTableViewController implementiert das Protokoll FetchedResultsTableViewControllerSearching. Dieses definiert die optionale Methode predicateForString:withOption:, welche der FetchedResultsTableViewController jedoch selbst nicht implementiert. In viewDidLoad: wird überprüft, ob eine erbende Klasse die Methode implementiert hat. Wenn dies zutrifft, wird im Tabellen Header ein Suchfeld hinzugefügt. Falls das Property searchBar zu diesem Zeitpunkt gesetzt ist, wird diese Search Bar verwendet. Falls nicht, so wird eine Standard UISearchBar erzeugt. Die UISearchBar wird von einem UISearchDisplayController referenziert. Dieser ist unter anderem für das Wechseln der normalen Ansicht zur Ansicht der Suchresultate und zurück, sowie für das Verwalten der Search Bar zuständig. Der UISearchDisplayController besitzt eine eigene UITableView zur Darstellung der Suchresultate. Für diese dient der FetchedResultsTableViewController ebenfalls als UITableViewDataSource und UITableViewDelegate. Um zu bestimmen, ob die Methoden dieser Protokolle von der normalen Table View oder von der Table View des Search Display Controllers aufgerufen wurden, kann das Property active des Search Display Controllers abgefragt werden.

NSFetchedResultsController
Der fetchedResultsController wird sowohl für die Anzeige der Grundmenge, sowie für die Darstellung der Suchresultate verwendet. Um dies zu ermöglichen, wird beim Setzen des fetchedResultsController Properties das Predicate des Fetched Results Controllers ausgelesen und in normalPredicate gespeichert. Wenn die Suche aktiviert wird, liefert die Methode predicateForString:withOption: ein NSPredicate, das als searchPredicate abgelegt wird. Während der Suche werden normalPredicate und searchPredicate UND-verknüpft und dem fetchedResultsController zugewiesen.

8.6.2 PictureSeriesViewController

Mit dem `PictureSeriesViewController` können Bildserien im Vollbildmodus dargestellt werden.

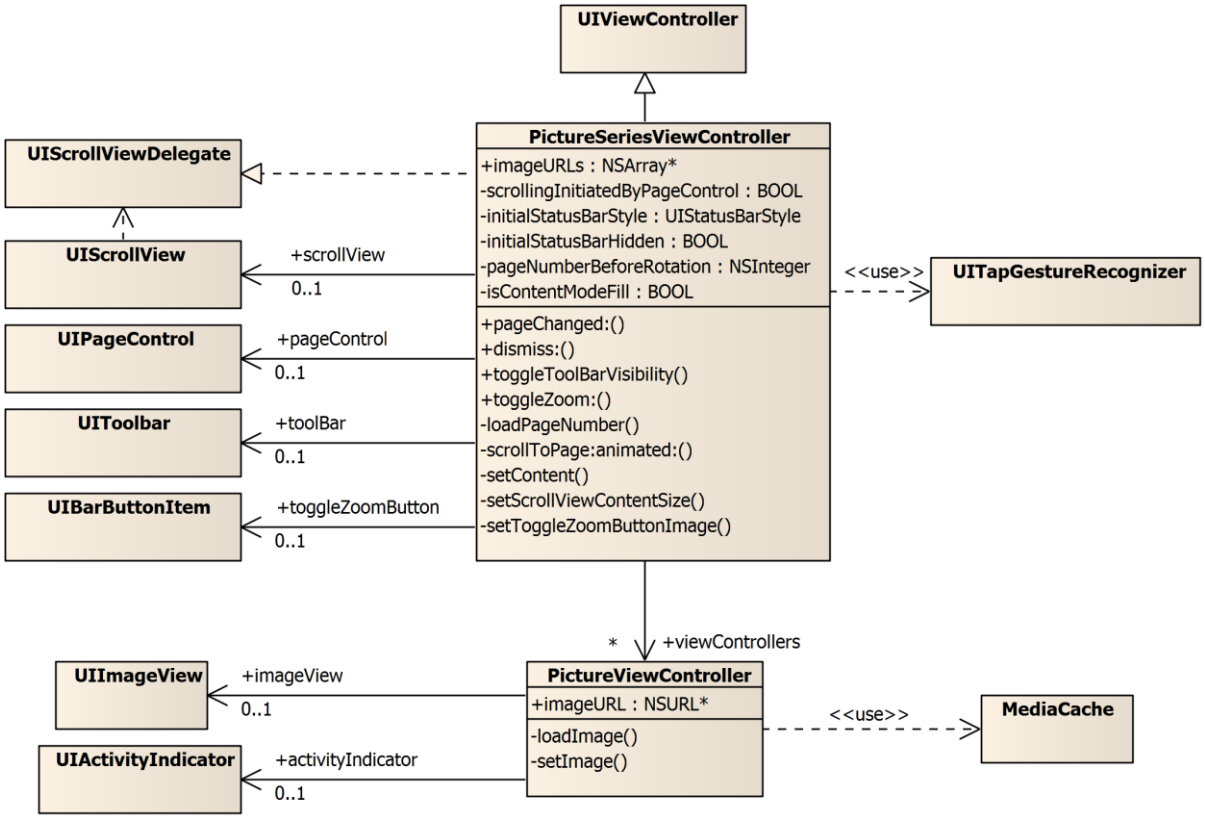

Abbildung 23: PictureSeriesViewController und abhängige Klassen

8.6.2.1 Anwendung

Der `PictureSeriesViewController` kann mit dem parameterlosen Initializer initialisiert werden. Die darzustellende Bildserie wird über das Property `imageURLs` gesetzt. Die Bildserie ist ein `NSArray`, das mit `NSURL` Objekten gefüllt ist. Die URLs zeigen auf Bilder auf einem Server.

Wenn der `PictureSeriesViewController` fertig initialisiert ist, kann er durch einen andern View Controller mit der Methode `presentModalViewController:animated:` angezeigt werden.

8.6.2.2 Aufbau und Funktion

Der `PictureSeriesViewController` erbt von `UIViewController`. In `viewDidLoad` wird die `view` des Controllers so initialisiert, dass sie den ganzen Bildschirm bis zum Rand füllt. Da die StatusBar durch den Controller manipuliert wird, wird deren Zustand in `viewDidLoad` abgefragt und gespeichert. Beim Verlassen wird der Zustand wieder hergestellt.

Die gesamte Fläche, die der `PictureSeriesViewController` zum Anzeigen zur Verfügung hat, wird durch eine `UIScrollView` ausgefüllt. In dieser wird pro darzustellendes Bild ein `PictureViewController` angezeigt. Vor der `scrollView` liegt ein `UIPageControl`. Dieses zeigt die Position des aktuellen Bildes in der Bildserie an und ermöglicht das Vorwärts- und Rückwärtsnavigieren durch einen Tap links oder rechts des Positionsindikators. Die Navigation löst die Action `pageChanged:` aus.

Die einzelnen PictureViewController werden bei Bedarf geladen. Dies geschieht in den Methoden des UIScrollViewDelegate Protokolls. Es werden jeweils der PictureViewController für das aktuelle, sowie das vorhergehende und das nachfolgende Bild geladen.

ToolBar
Der PictureSeriesViewController hat eine UIToolbar. Diese enthält einen Button, der die Action dismiss: auslöst, um zurück zur vorherigen Ansicht zu gelangen. Ein weiterer Botton vom Typ UIBarButtonItem wäre für das *Outlet* toggleZoomButton vorgesehen. Mit diesem hätte man die Möglichkeit, zwischen den Zoomstufen Fit (ganzes Bild einpassen) und Fill (ganzen Bereich bis an den Rand mit Bild füllen) zu wechseln. Um diesen Wechsel auszulösen, müsste der Button die Action toggleZoom: auslösen.

Ein UITapGestureRecognizer, der auf scrollView registriert wird, löst die Action toggleToolBarVisibility aus. Diese bewirkt das Aus- bzw. Einblenden der toolBar und der StatusBar.

PictureViewController
Die einzelnen Bilder werden mit einem PictureViewController dargestellt. Dieser enthält eine UIImageView und einen UIActivityIndicator. Wenn der Controller geladen wird und die URL eines Bildes in imageURL Property ist, wird das Bild mit Hilfe des MediaCache geladen. Das Laden erfolgt asynchron. Beim Aufruf der MediaCache Methode requestImage:callback:onTarget: wird im Parameter callback die Methode angegeben, die das UIImage entgegen nimmt, wenn der MediaCache das Bild fertig geladen hat.

Während des Ladevorgangs wird der activityIndicator angezeigt.

8.6.2.3 Verbesserungsmöglichkeiten

Zustand der Status Bar
Das Speichern des Zustands der Status Bar in der Methode viewDidLoad ist im Grunde zu früh. Der passende Zeitpunkt dafür wäre viewDidAppear. So wie der PictureSeriesViewController aktuell angewendet wird, ist dies kein Problem. Wenn jedoch der PictureSeriesViewController nicht gleich nach dem Erzeugen angezeigt wird, könnte sich der Zustand der Status Bar zwischenzeitlich verändern. Dies würde dazu führen, dass sich die Zustände der Status Bar vor und nach der Anzeige des PictureSeriesViewControllers unterscheiden.

Besseres Verhalten bei temporären Fehlern
Wenn der MediaCache kein Bild zu der angegebenen URL laden konnte, liefert er nil zurück. Dies kann vorkommen, wenn das Bild noch nicht im Cache gespeichert ist und der Server auf Grund eines temporären Netzwerkunterbruchs (z.B. Fahrt durch einen Tunnel) nicht erreicht werden kann. Da der Ladevorgang durch Aufruf der Callback-Methode als abgeschlossen gilt, bleibt das Bild schwarz, auch wenn der Server nach dem Unterbruch wieder erreichbar wäre. Wenn der PictureSeriesViewController verlassen und wieder geöffnet wird, wird erneut versucht die Bilder zu laden.

Der PictureViewController könnte mit einem Flag versehen werden, das angibt, ob das Bild erfolgreich geladen wurde. Der PictureSeriesViewController könnte dieses Flag in der Methode loadPageNumber: überprüfen und falls das Laden des Bildes fehlgeschlagen ist, das Nachladen des Bildes veranlassen.

8.7 Network Layer

Die Klassen des Network Layers sind für die Kommunikation mit der REST-Schnittstelle zuständig. In diesem Abschnitt werden die Klassen kurz beschrieben. Weiter unten wird die Klasse `RESTClientRequest` ausführlich beschrieben.

Klasse	Beschreibung
CurrentImpulseRequest	Stellt eine Anfrage an die REST-Schnittstelle für `CurrentImpulse`. Die Klasse wertet die Anfrage aus und legt die neuen Impulse in der Datenbank ab. Die Anfrage wird ebenfalls dazu verwendet, um die *Credentials* des Benutzers auf dem Server zu prüfen.
JSONConverter	Mit dem `JSONConverter` werden Antworten, die als JSON empfangen werden, in Objekte umgesetzt. Die Klasse wird ebenfalls dazu benutzt, um Objekte in JSON umzuwandeln, um diese an die REST-Schnittstelle zu senden.
OperationResult	Fasst das Resultat der Methode `checkCredentials:andCallback:` zusammen.
RegisterTokenRequest	Stellt eine Anfrage an die REST-Schnittstelle für `RegisterToken` dar. Sie sendet den Device Token und den Gerätenamen an den Server.
RESTClient	`RESTClient` ist eine Fassade für die Kommunikation mit der REST-Schnittstelle. Er erzeugt und verwaltet die Anfrage Objekte und meldet - wenn nötig - Resultate an den Aufrufer zurück. Der `RESTClient` fasst neue Anfragen zusammen, bis die Serverantwort der alten Anfrage fertig empfangen wurde.
RESTClientRequest	`RESTClientRequest` ist eine Basisklasse für Anfragen an die REST-Schnittstelle. Sie implementiert das Verhalten, das allen Anfragen gemeinsam ist.

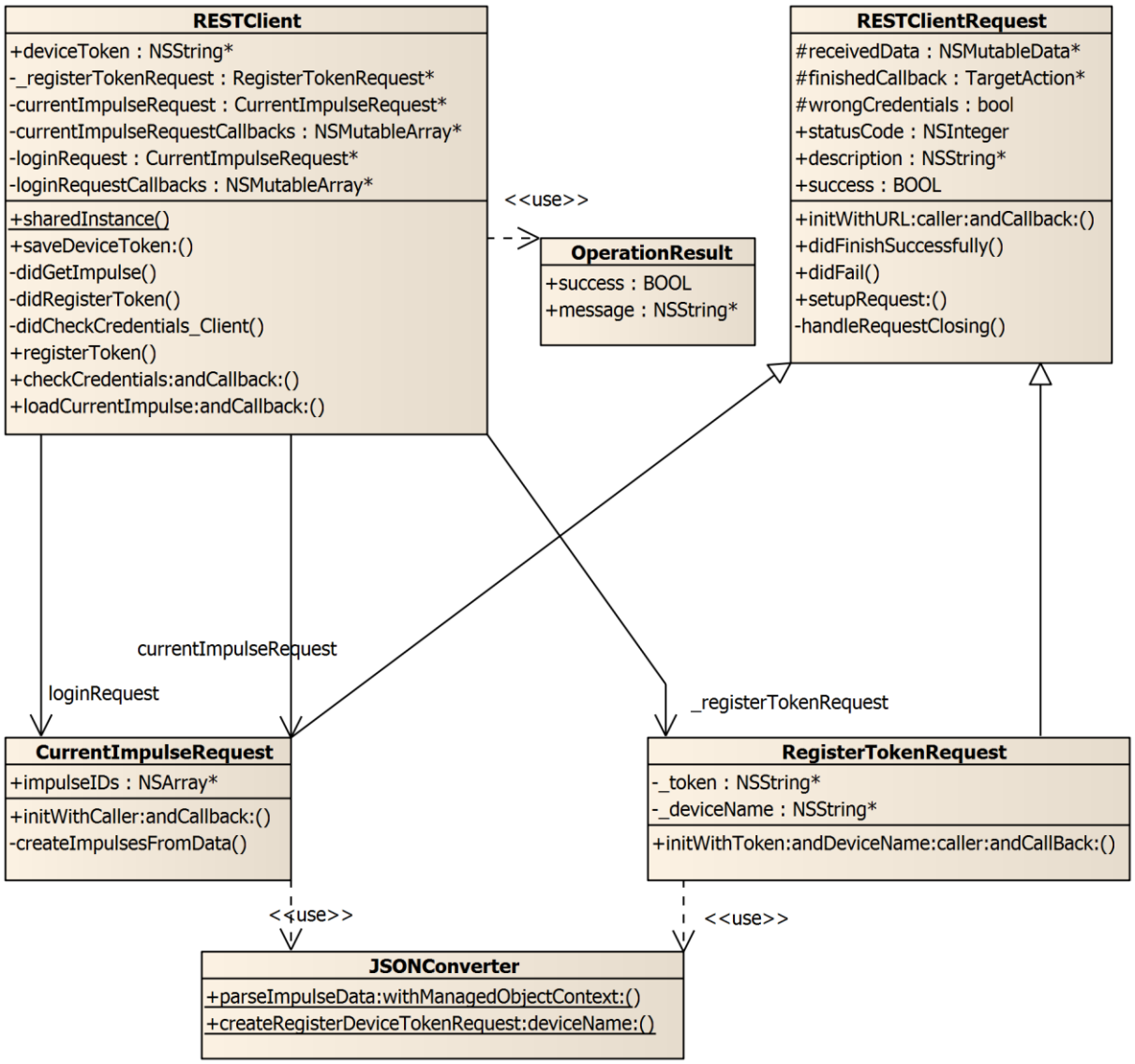

Abbildung 24: Klassen des Network Layers mit Abhängigkeiten

8.7.1 RESTClientRequest

`RESTClientRequest` ist die Basisklasse für `CurrentImpulseRequest` und `RegisterTokenRequest`. Diese Klasse implementiert Funktionen, die in den abgeleiteten Klassen benötigt werden.

Die Basisklasse realisiert die folgenden Aufgaben:

- Feststellen, ob der Request erfolgreich war oder nicht. Wenn `success` den Wert `NO` hat, kann über `statusCode` und `description` ermittelt werden, weshalb der Request nicht erfolgreich war.
- Dem Initializer können ein Objekt und ein Selector als Callback übergeben werden, der ausgeführt werden soll, wenn der Request fertig bearbeitet wurde. Der Callback wird sowohl bei Erfolg, wie auch bei einem Fehlschlagen des Requests ausgelöst.
- RESTClientRequest bestimmt, welche Zertifikate akzeptiert werden und kann den Client gegenüber dem Server mit Basic Authentication authentisieren.
- Diverse Funktionen für den Empfang von Daten & Response
- Erweiterungspunkte: Methoden mit Default-Verhalten, die bei Bedarf von den abgeleiteten Klassen überschrieben werden können.

Erweiterungspunkte

`setupRequest:(NSMutableURLRequest *)request`
: Mit dieser Methode lässt sich der Request anpassen. Dies ist nötig, falls nicht GET verwendet werden soll, sondern z.B. ein POST mit Payload.

Property `success`
: In der Basisimplementierung gibt dieses Property genau dann `YES` zurück, wenn der Request erfolgreich war und der `statusCode` 200 ist. Das Property kann jedoch so überschrieben werden, dass auch andere `statusCodes`, wie z.B. 206 (Erfolgreich, aber kein Body), als Erfolg gewertet werden.

`didFinishSuccessfully`
: Wird aufgerufen, wenn der Request erfolgreich abgeschlossen wurde.

`didFail`
: Wird aufgerufen, wenn der Request nicht erfolgreich beendet wurde.

`handleRequestClosing`
: Wird ausgeführt, wenn der Request zu Ende ist. Wenn der Request erfolgreich beantwortet wurde, wird aus dieser Methode `didFinishSuccessfully` aufgerufen, sonst `didFail`. Im Anschluss wird der Callback ausgeführt.

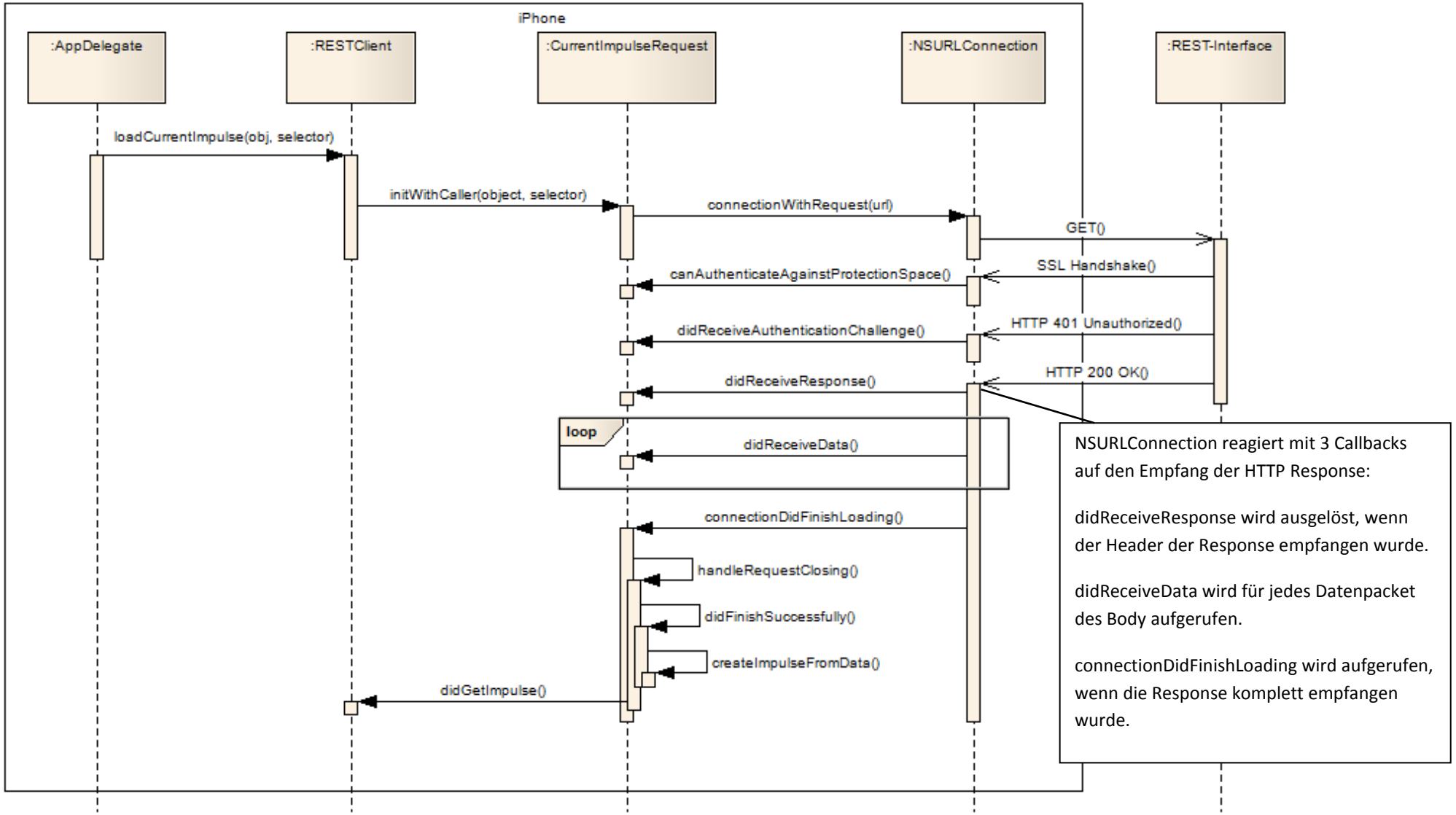

Abbildung 25: Ablauf des Ladens der Impulse mit CurrentImpulseRequest (Basisklasse: RESTClientRequest) inklusive Netzwerkkommunikation mit der REST-Schnittstelle

8.8 Domain Layer

Der Domain Layer enthält Klassen, die die Domain Konzepte repräsentieren, sowie Klassen, die zu deren Verwaltung benutzt werden. Dieser Abschnitt gibt eine Übersicht über alle beteiligten Klassen. Weiter unten werden die Klassen detaillierter beschrieben.

Klasse	Beschreibung
Impulse	Die Klasse `Impulse` repräsentiert das Domain Konzepte Delivery. Da eine Delivery genau einen Impuls hat, werden die zwei Konzepte zusammengefasst. Neben den persistenten Properties enthält die Klasse verschiedene Hilfsmethoden.
ManagedContextFactory	Die `ManagedContextFactory` ist für das Bereitstellen des Context für die persistenten Objekte verantwortlich.
Media	`Media` repräsentiert Bilder oder Videos.
MediaAssociation	`MediaAssociation` ist die Zuordnung von Bildern und Videos zu Impulsen. Mit Hilfe des `rank` Properties wird die Reihenfolge von Bildern in Bildserien gespeichert.
MediaCache	`MediaCache` ist für das Laden und Speichern der Bilder verantwortlich.
MediaRequest	Ein `MediaRequest` entspricht einem Ladevorgang eines Bildes.
MediaRequestDelegate	Der `MediaRequestDelegate` definiert eine Methode, mit der ein `MediaRequest` seinen Auftraggeber benachrichtigen kann, wenn er fertig ist.
User	Repräsentiert das Domain Konzept User.

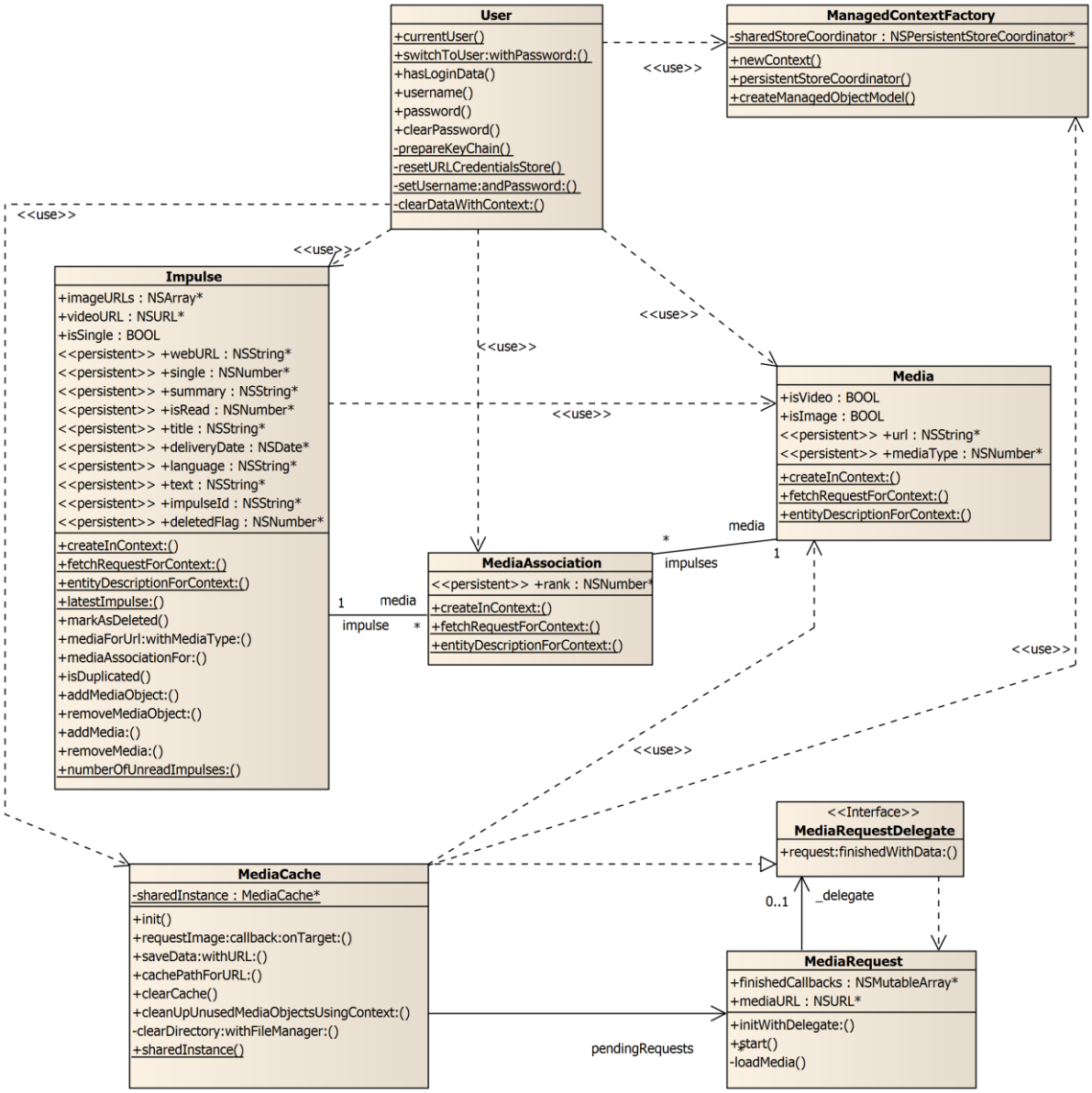

Abbildung 26: Klassen des Domain Layers mit Abhängigkeiten

8.8.1 Managed Object Context

Der Managed Object Context ist für das Persistieren der Impulse zuständig. Da die Centrado App mehrere Contexts besitzt, die durch einen Koordinator verbunden sind, wird das Zusammenspiel in diesem Abschnitt genauer erklärt.

Meistens gibt es in der Centrado App genau einen Context, den GUI-Context. Er stellt die verschiedenen Impulse für die `ImpulseListView` und die `ImpulseView` zur Verfügung. Es kann aber sein, dass kurzeitig zwei Contexte existieren, nämlich immer dann, wenn ein `CurrentImpulseRequest` (Impulse laden) ausgelöst wird. Nach dem `CurrentImpulseRequest` alle Daten von der REST-Schnittstelle erhalten hat, müssen die Impulse noch persistiert werden.

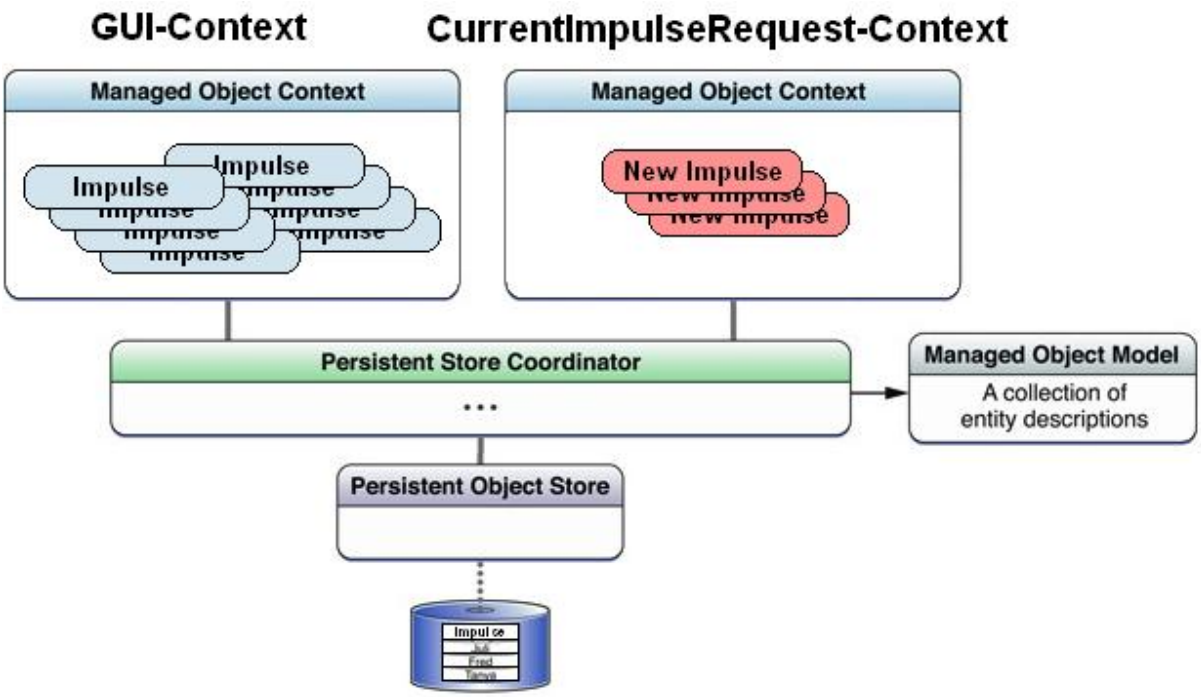

Abbildung 27: Managed Object Context (Quelle: [PSCO])

Neue Impulse persistieren:

- `CurrentImpulseRequest` holt sich als Erstes mit Hilfe der `ManagedContextFactory` einen neuen Context.
- Die neuen Impulse werden gleich nachdem sie geparst wurden im Context abgelegt.
- Im nächsten Schritt werden die doppelten Impulse wieder aus dem Context entfernt.
- Anschliessend wird der Context gespeichert.

Nun müssen sämtliche Impulse aus dem aktuellen Context noch in den GUI-Context übertragen werden:

- Nach dem Save-Aufruf auf dem aktuellen Context, wird eine `NSManagedObjectContextDidSaveNotification` verschickt, die sogleich vom `CENTRADOAppDelegate` empfangen wird.
- Auf dem GUI-Context (`managedObjectContextForUI`) werden die Änderungen nun eingefügt (merge). Da die versendete Notification alle Änderungen des neuen Context enthält, kann sie direkt vom GUI-Context zur Aktualisierung verwendet werden.
- Der `ImpulseListViewController` muss diese Änderungen ebenfalls erfahren und darstellen. Die nötige Funktionalität wird durch den `NSFetchedResultController` bereitgestellt. Er erhält sämtliche Änderungen vom GUI-Context und löst über seinen Delegate (`ImpulseListViewcontroller`) die Aktualisierung des GUI aus.

Der `ImpulseViewController` bekommt von der Aktualisierung des GUI-Contexts nichts mit. Er wird separat (z. B. beim Eingang einer Push Notification) aktualisiert.

Schema Migration

Falls es in einer neuen Version der App eine Schemaänderung der Datenbank gibt, so ist eine Speicherung der Daten mit erhöhtem Aufwand verbunden. In diesem Fall können verschiedene App-Verhalten programmiert werden:

- Die optimale Variante wäre die Daten-Migration, bei der die Daten aus dem alten Schema ins neue konvertiert / eingefügt werden. Dies ist relativ aufwendig und erfordert bei mehreren Schema-Versionen auch ebenso viele Varianten des Migrations-Codes.
- Die Lightweight Migration ist ein Verfahren, bei dem Core Data einfache Schemaänderungen

erkennen und selbständig die DB-Struktur anpassen kann. Dies ist jedoch nur für einfache Änderungen, die keine zusätzlichen Informationen benötigen, möglich. Das Hinzufügen einer Spalte, die null-Werte erlaubt, ist möglich. Wenn die neue Spalte aber keine null-Werte zulässt und keine Default Werte definiert, ist keine Lightweight Migration möglich.

- Die einfachste Variante ist das Löschen und Neuanlegen der Datenbank. Somit müssen die Daten nicht migriert werden. Die gelöschten Daten können dann, falls wieder benötigt, erneut vom Server heruntergeladen werden.

In der Centrado App wurde die zweite und dritte Variante umgesetzt. Wenn keine Lightweight Migration möglich ist, wird die Datenbank gelöscht. Dies ist vor allem deshalb möglich, weil der Benutzer keine Daten ändern kann. Die Änderungen würden beim Löschen der Datenbank verloren gehen.

Persistente Objekte

In der Centrado App müssen drei Klassen persistiert werden: `Impulse`, `Media` und `MediaAssociation`.

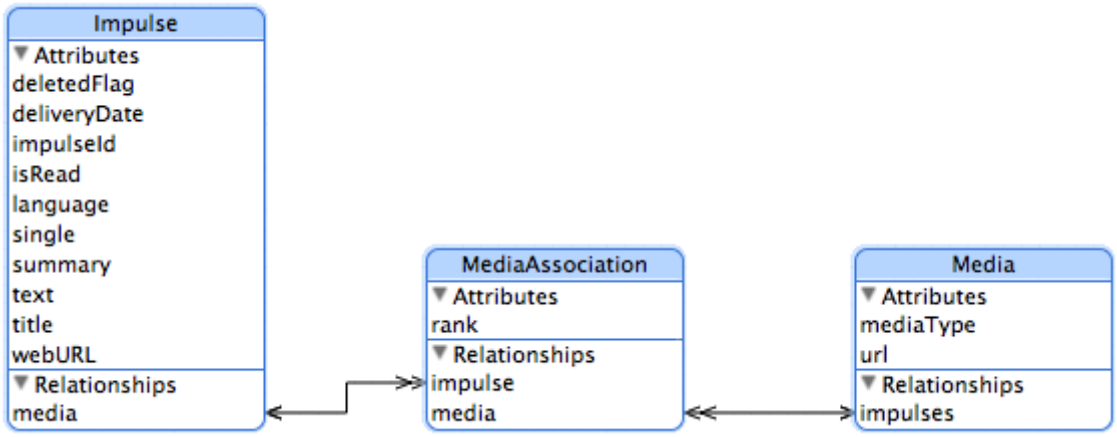

Abbildung 28: Model der persistenten Klassen

Alle drei Klassen besitzen folgende Hilfsmethoden:

`createInContext:`
: Erstellt ein neues Managed Object im angegebenen Context. Das Erstellen ist ein einfacher Aufruf einer statischen Methode. Da aber die Klasse des zu erstellenden Objektes als String angegeben werden muss, können auf diese Weise Fehler vermieden werden.

`entityDescriptionForContext:`
: Mit dieser Methode kann die NSEntityDescription mit den Metadaten für die entsprechende Klasse ausgelesen werden (für FetchRequests benötigt).

`fetchRequestForContext:`
: Erstellt einen `NSFetchRequest` zum Laden von Managed Objects. Um das Laden auf einen Typ zu beschränken, wird gleich nach dem Erstellen des Request der Typ der zu ladenden Managed Objects gesetzt.

Besonderheiten des Impuls‘

Der `Impulse` besitzt ein Attribut `deletedFlag`. Wenn dieses gesetzt ist, gilt der `Impulse` als gelöscht. Der Impulse darf nicht aus der Datenbank gelöscht werden, da dieser Status nur lokal ist. Auf dem Server sind immer noch alle Impulse vorhanden. Bei einer Anfrage werden alle Impulse an die App gesendet. Diese kann dann mit Hilfe des Attributes deliveryDate und impulseId herausfinden, welche Impulse bereits in der lokalen Datenbank abgelegt sind.

Das Attribut darf nicht deleted genannt werden. Dies scheint ein reserviertes Wort zu sein. In der Dokumentation zu Core Data ist dazu nichts aufgeführt. Dennoch traten bei der Entwicklung der App,

bei der Verwendung von deleted als Attributname, Inkonsistenzen bei der Behandlung der Ereignisse für die Benutzeroberfläche auf.

Methoden

`latestImpulse:`
: Liest den aktuellsten Impuls aus dem Context. In der Abfrage an den Context wird darauf geachtet, dass gelöschte Impulse ignoriert werden. Damit der richtige Impuls gefunden wird, werden alle Impulse nach Datum sortiert, sodass der aktuellste `Impulse` gleich zurückgeliefert werden kann.

`numberOfUnreadImpulses:`
: Ähnlich wie bei `latestImpulse:` wird ein Request an den Context geschickt. Für das Auslesen werden zwei Filter gesetzt: Impuls ist nicht gelöscht und nicht gelesen.

`markAsDeleted`
: Setzt das `deletedFlag` eines Impuls'. Anschliessend werden alle seine `MediaAssociations` gelöscht.

Properties

`imageURLs`

- Der Setter (`setImageURLs:`) nimmt einen `NSArray` von Bildern entgegen. Anschliessend legt er die für die Persistierung nötigen, Objekte vom Type `Media` und `MediaAssociation` an. Damit die Bilder in der richtigen Reihenfolge bleiben, wird jede `MediaAssociation` mit einer Nummer (`rank`) versehen.
- `ImageURLs` lädt die `MediaAsscciations` eines Impuls'. Die geladenen Associations werden nach den `rank` sortiert.
  Sofern ein Media ein Bild ist, wird der URL aus dem Media kopiert und in den Array eingefügt. Der Array wird anschliessend zurückgegeben.

`videoURL`

- `setVideoURL:` legt eine Instanz der Klasse `Media` an, in der der URL zum Video gespeichert wird. Das `Media` wird dann über eine `MediaAssociation` mit dem Impuls verlinkt.
- `videoURL` durchsucht die Medien des Impuls' nach einem Video und liefert den URL des Videos, über die Association, aus dem Media-Object zurück.

Besonderheiten von Media

Das Attribut `mediaType` definiert, ob es sich beim Media Objekt um ein Bild oder Video handelt. Da mediaType als Zahl definiert ist, können auch weitere Typen definiert werden.

Properties

`isVideo` (readonly) liefert einen Bool zurück, der aussagt, ob ein Media ein Video ist.
`isImage` (readonly) liefert einen Bool zurück, der aussagt, ob ein Media ein Bild ist.

8.8.2 Benutzerangaben abrufen und löschen

Die Klasse User ist für das sichere Ablegen der Benutzerdaten zuständig. Um das Handling etwas zu vereinfachen, wurde eine Bibliothek verwendet (siehe 6.2.2.2 Sicheres Speichern von Credentials). Die Zugriffe auf diese Bibliothek werden alle durch die Klasse `User` gekapselt.

Die Klasse `User` hat zwei statische Methoden: `currentUser` und `switchToUser:withPassword:`. Die Methode `currentUser` gibt das `User` Objekt zum aktuell angemeldeten Benutzer zurück. `switchToUser:withPassword:` ändert die Benutzerangaben. Alle restlichen Methoden sind Instanz-Methoden. Somit werden alle benutzerspezifischen Methoden auf dem User Objekten ausgeführt.

Nachfolgend ist ein Sequenzdiagramm abgebildet, das die wichtigsten Operationen auf `User` zeigt.

Funktionsname	Beschreibung
currentUser	CurrentUser liefert den aktuellen User zurück. Falls nötig wird einer erstellt.
switchToUser:withPassword:	Speichert Benutzername und Passwort in der Key Chain ab. Löscht den *Credential Store* (ev. zwischengespeicherte Benutzeran-gaben). Sofern der User geändert hat, wird bei der `ManagedContextFactory` ein neuer Context angefordert und alle `Impulse`, `Media` und `MediaAssociation` Objekte gelöscht. War dies der Fall, so muss der `MediaCache` ebenfalls geleert werden.
username	Liest den Benutzernamen aus der Key Chain.

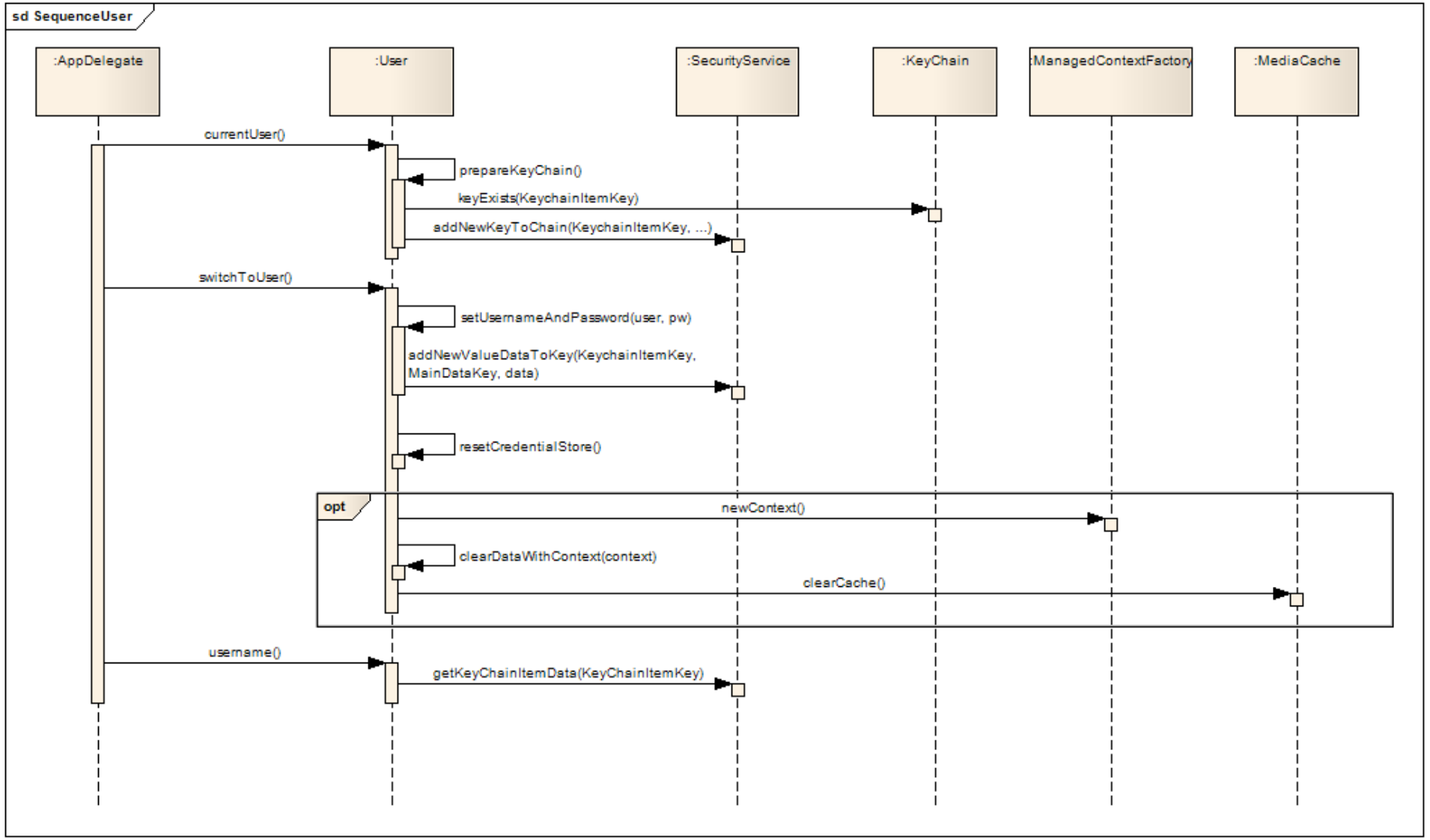

Abbildung 29: Sequenzdiagramm zu ausgewählten Methoden der Klasse User

8.8.3 MediaCache

`MediaCache` ist eine Klasse, mit der Bilder von einem Server geladen und für die Offlineverwendung zwischengespeichert werden können.

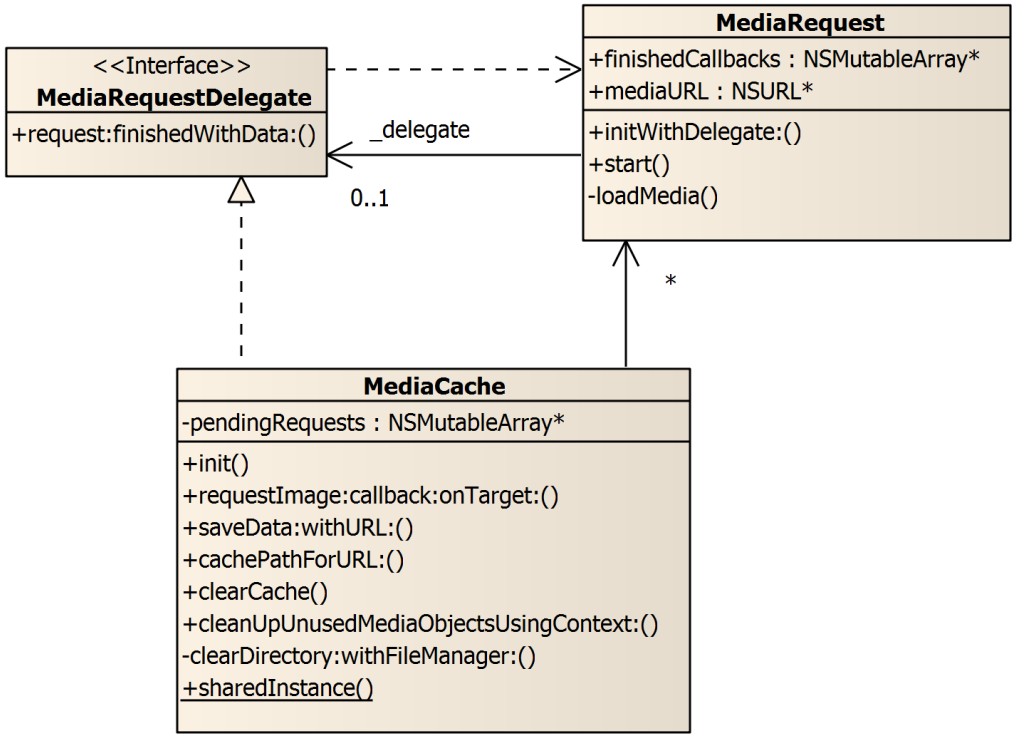

Abbildung 30: Klassenstruktur des MediaCache

8.8.3.1 Verwendung

MediaCache wird als Singleton verwendet, da es nur einen Cache gibt und die Anfragen koordiniert werden. Die Klassenmethode `sharedInstance` gibt die Instanz zurück. Sämtliche Methoden sind thread-safe.

Um ein Bild anzufordern, wird die Methode `requestImage:(NSURL)imageURL callback:(SEL)callback onTarget:(id) target` aufgerufen. Dies löst das asynchrone Laden des Bildes, das mit dem Parameter `imageURL` angegeben wird, aus. Mit `callback` und `target` werden das Objekt und die Methode bezeichnet, an die das geladene Bild übergeben werden soll.

Die Methode `cleanUpUnusedMediaObjectsUsingContext:` überprüft alle `Media` Objekte, ob sie noch über eine `MediaAssociation` einem `Impulse` zugeordnet sind. Wenn ein `Media` Objekt keinem `Impulse` mehr zugeordnet ist, wird das Bild, das in `Media` angegeben ist, aus dem Cache gelöscht. Wenn sich das Bild noch nicht im Cache befindet, geschieht nichts.

Die Methode `clearCache` löscht alle Bilder, die aktuell im Cache gespeichert sind. Dies geschieht unabhängig vom Zustand der `Media` Objekte.

8.8.3.2 Aufbau und Funktion

Dateistruktur des Caches

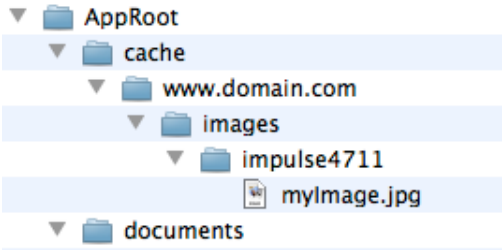

Abbildung 31: Dateistruktur im Cache für ein Bild mit der URL http://www.domain.com/images/impulse4711/myImage.jpg

Die Bilder, die von MediaCache gespeichert werden, werden als Dateien im lokalen Dateisystem des iPhones abgelegt. In Abbildung 31 ist mit AppRoot das Verzeichnis bezeichnet, in welchem die App arbeiten darf. Das Unterverzeichnis cache ist von Apple für Caches vorgesehen und wird beim Synchronisieren mit iTunes nicht gesichert ([iAPG] Seite 17).

Die Struktur innerhalb des cache-Verzeichnisses ist folgendermassen aufgebaut. Der Host-Teil der URL wird als Verzeichnisname auf der ersten Ebene verwendet. Darunter wird der Pfad-Teil der URL als Unterverzeichnisse abgebildet. Schlussendlich wird das Bild in einer Datei abgelegt, die den Namen der Datei in der URL hat.

Bild laden

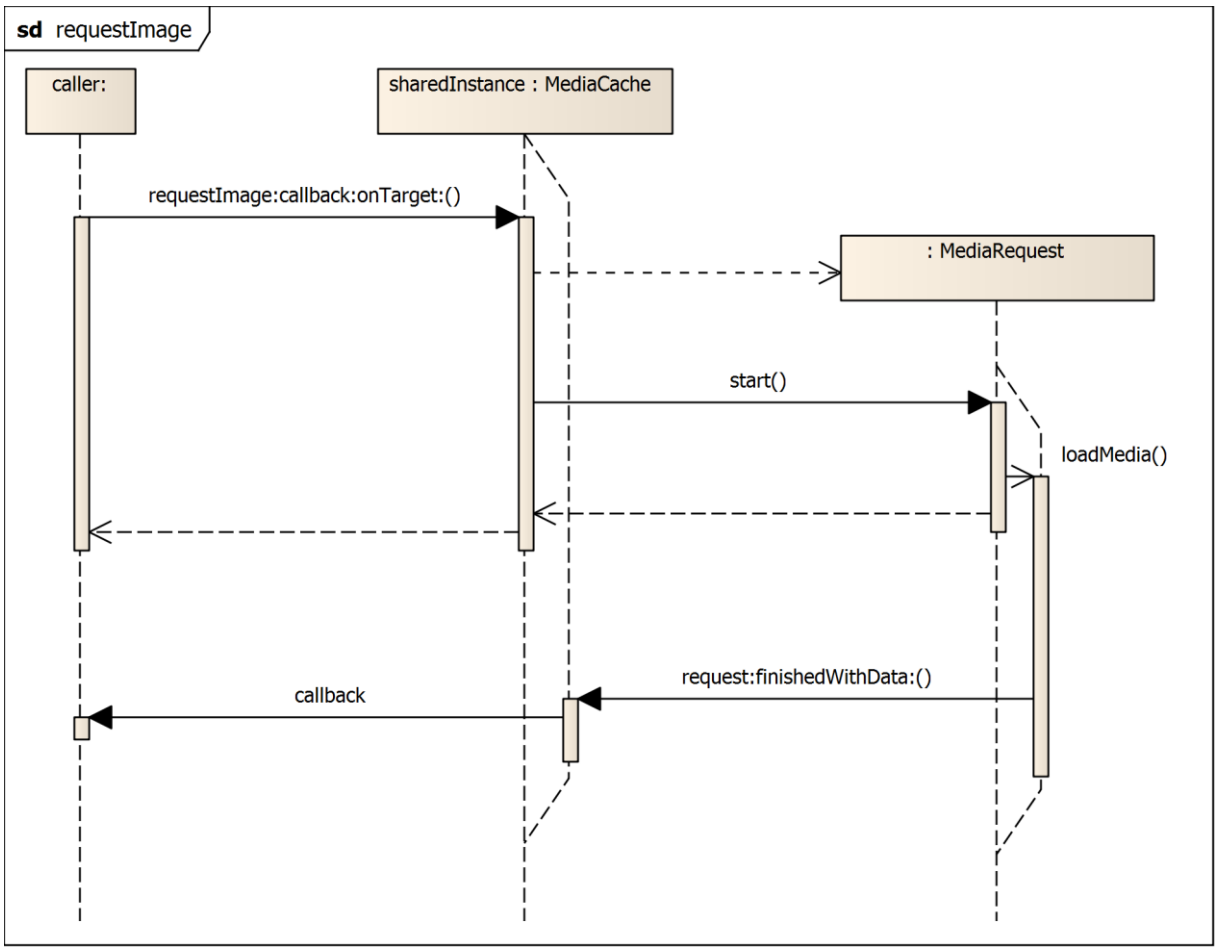

Abbildung 32: Laden eines Bildes, das noch nicht im Cache gespeichert ist (vereinfacht dargestellt)

Um ein Bild zu laden, wird in `requestImage:(NSURL)imageURL callback:(SEL)callback onTarget:(id) target` als erstes geprüft, ob sich das Bild bereits im Cache befindet. Wenn dies zutrifft, wird das Bild aus dem Cache geladen und der `callback` wird synchron ausgeführt.

Ist das Bild noch nicht im Cache, wird ein `MediaRequest` Objekt erzeugt. Dieses wird mit dem `MediaCache` als `_delegate` initialisiert. Über den `_delegate` meldet der `MediaRequest` mit `request:FinishedWithData:` zurück, wenn die Bilddaten fertig geladen wurden. Das `MediaRequest` Objekt wird im Dictionary `pendingRequests` mit `imageURL` als Key abgelegt.

Bevor mit der Methode `start` das eigentliche Laden der Bilddaten ausgelöst wird, werden die Parameter `callback` und `target` im `MediaRequest` Objekt im Array `finishedCallbacks` abgelegt. In der Methode `start` wird für das Laden ein Hintergrundthread erzeugt, der die Methode `loadMedia` ausführt. Nach dem Anstossen des Ladevorgangs im Hintergrund, kehren die Aufrufe von `start` und `requestImage:callback:onTarget:` zurück und der Aufrufer hat die Kontrolle zurück.

Wenn während der Bearbeitung des `MediaRequests` ein weiteres Mal `requestImage:callback:onTarget:` für die gleiche URL aufgerufen wird, kann dies mit Hilfe des Eintrages in `pendingRequests` festgestellt werden. Es wird kein neues `MediaRequest` Objekt erzeugt, sondern lediglich `callback` und `target` zu den `finishedCallbacks` des bestehenden `MediaRequests` hinzugefügt.

Wenn der `MediaRequest` im Hintergrund die Daten fertig geladen hat, ruft er `request:FinishedWithData:` auf dem `MediaCache` auf. Dieser speichert die Daten in eine Datei im lokalen Dateisystem. Mit der Methode `cachePathForURL:` wird der Pfad im lokalen Dateisystem ermittelt. Anschliessend wird ein `UIImage` aus den Daten erzeugt und es werden die `callbacks`, die in `finishedCallbacks` abgelegt wurden, auf dem Mainthread ausgeführt. Nachdem alle `callbacks` ausgeführt wurden, wird der `MediaRequest` aus den `pendingRequests` des `MediaCache` entfernt.

Alle Zugriffe auf `MediaCache.pendingRequests` und `MediaRequest.finishedCallbacks` sind synchronisiert und somit thread-safe.

Cache leeren
Mit der Methode `clearCache` werden alle Objekte im Cache gelöscht. Dazu wird der komplette Inhalt des cache-Verzeichnisses der App gelöscht.

Cache aufräumen
Die Methode `cleanUpUnusedMediaObjectsUsingContext:(NSManagedObjectContext*)context` entfernt alle Medien, die von keinem `Impulse` mehr gebraucht werden, aus dem Cache. Dazu wird über alle Media Objekte in der Datenbank iteriert und überprüft, ob das `Media` Objekt noch `MediaAssociations` hat. Sind keine `MediaAssociations` vorhanden, wird über `cachePathForURL:` die Datei im cache-Verzeichnis ermittelt und falls diese vorhanden ist gelöscht. Anschliessend wird das nicht mehr benötigte `Media` Objekt gelöscht.

8.8.3.3 Einschränkungen und Verbesserungsmöglichkeiten

Dateistruktur im Cache
Das Mapping zwischen URL und lokaler Dateistruktur, das in der Methode `cachePathForURL:` gemacht wird, hat zwei Nachteile. Zum einen werden Bilder mit einer URL, die die Bilder nicht über den Pfad, sondern über Parameter referenziert ,falsch geladen. Zwei Bilder mit der URL http://server/service?image=42 und http://server/service?image=43 würden beide auf die lokale

Datei service abgebildet. Zum andern könnte es beim aktuellen Mapping vorkommen, dass die URL bzw. der resultierende Pfad länger ist als dass dies das lokale Dateisystem zulassen würde.

Beide Probleme könnten umgangen werden, wenn in der Methode `cachPathForURL:` aus der kompletten URL ein SHA1 Hashcode erzeugt würde. Dieser ist 20 Byte lang und ist bei einigen Hundert Cacheeinträgen mit sehr hoher Wahrscheinlichkeit (Grössenordnung $1:10^{15}$) kollisionsfrei [SHA1], [BdAttk].

Maximale Cache Grösse
Die aktuelle Implementierung des MediaCache hat keine Begrenzung, wie viel Speicher für das Cachen von Bildern verwendet werden darf.

8.9 Util

Der Util Layer enthält Klassen mit Grundfunktionalität, die nicht spezifisch für die Centrado App sind. Dieser Layer enthält auch die Libraries für die Key Chain und den JSON-Parser und -Generator. Die Klassen werden in diesem Abschnitt kurz beschrieben.

Klasse	Beschreibung
TargetAction	`TargetAction` fasst ein Objekt und ein Selektor zusammen. Mit dieser Struktur lassen sich Callbacks einfach in einem Array speichern. Einmal erzeugte Objekte dieser Klasse können nachträglich nicht mehr verändert werden.
Utilities	`Utilities` enthält eine Sammlung von Methoden, die allgemeine Funktionalität bereitstellen.

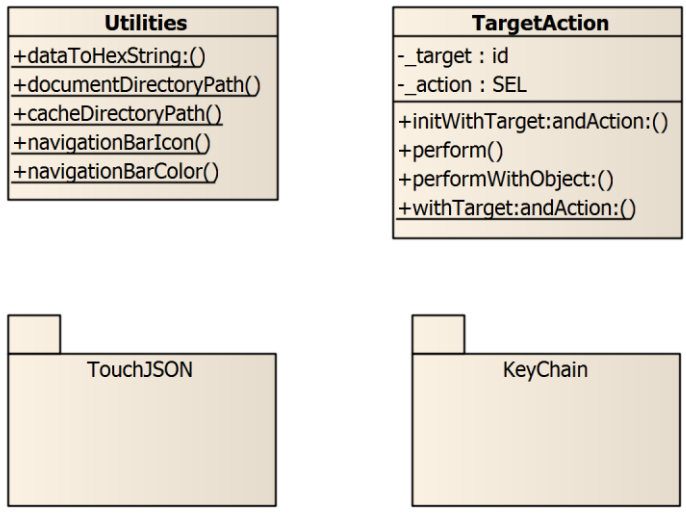

Abbildung 33: Klassen und Packages des Util Layers

Package	Beschreibung
KeyChain	Enthält Klassen der Library für den Zugriff auf die Key Chain
TouchJSON	Enthält Klassen der Library für das Parsen und Generieren von JSON.

8.10 Threads

Die Centrado iPhone App arbeitet an einigen Stellen mit mehreren Threads. Dabei werden verschiedene Konzepte eingesetzt um Konflikte zu verhindern.

GUI

Alle GUI-Objekte wie Views, Buttons etc. dürfen nur im Mainthread bearbeitet werden. Die Objekte sind aus Performancegründen nicht synchronisiert. Um GUI-Objekte in Code zu bearbeiten, der potenziell nicht auf dem MainThread ausgeführt wird, wird `performSelectorOnMainThread:withObject:waitUntilDone:` verwendet, um die Methode zur Bearbeitung der GUI-Objekte auf dem Mainthread auszuführen.

Core Data

Für Core Data Objekte gilt, mit Ausnahme des `NSPersistentStoreCoordinator`, Thread Confinement. Dies bedeutet, dass die Objekte nur aus genau einem Thread erzeugt und bearbeitet werden dürfen. Sie dürfen nicht an andere Threads übergeben werden. Dies erspart die Notwendigkeit, die Operationen auf diesen Objekten zu synchronisieren. Falls z.B. ein `Impulse` von einem Thread zur Bearbeitung an einen Hintergrundthread übergeben werden soll, muss statt des `Impulse` Objekts seine Object ID übergeben werden. Mit einem `NSManagedObjectContext`, der für den Hintergrundthread erzeugt wurde, kann dann über die Object ID der entsprechende `Impulse` geladen und bearbeitet werden.

Der `NSPersistentStoreCoordinator` ist eine Ausnahme. Es darf für einen Data Store, z.B. die Datenbank in der die Impulse gespeichert werden, nur eine Instanz des `NSPersistentStoreCoordinator` geben. Deshalb ist seine Erzeugung und der Zugriff auf ihn synchronisiert. Die einzelnen `NSManagedObjectContext` Objekte, die den `NSPersistentStoreCoordinator` verwenden, sorgen beim Zugriff auf ihn intern für die nötige Synchronisation.

Singleton

Die Klassen `User` und `MediaCache` werden als Singleton verwendet. Alle Methoden auf diesen beiden Klassen sind thread-safe implementiert. Wo notwendig, wurde Synchronisation verwendet.

8.11 Verhalten der App

In diesem Abschnitt wird das Verhalten der App genauer erklärt. Die Abläufe, die im Hintergrund stattfinden, sind für den Benutzer nicht immer sofort ersichtlich und werden deshalb hier noch eingehend erklärt.

Beim Öffnen und Schliessen der App treten verschiedene Events ein. Ebenso beim Eingang von Push Notifications.

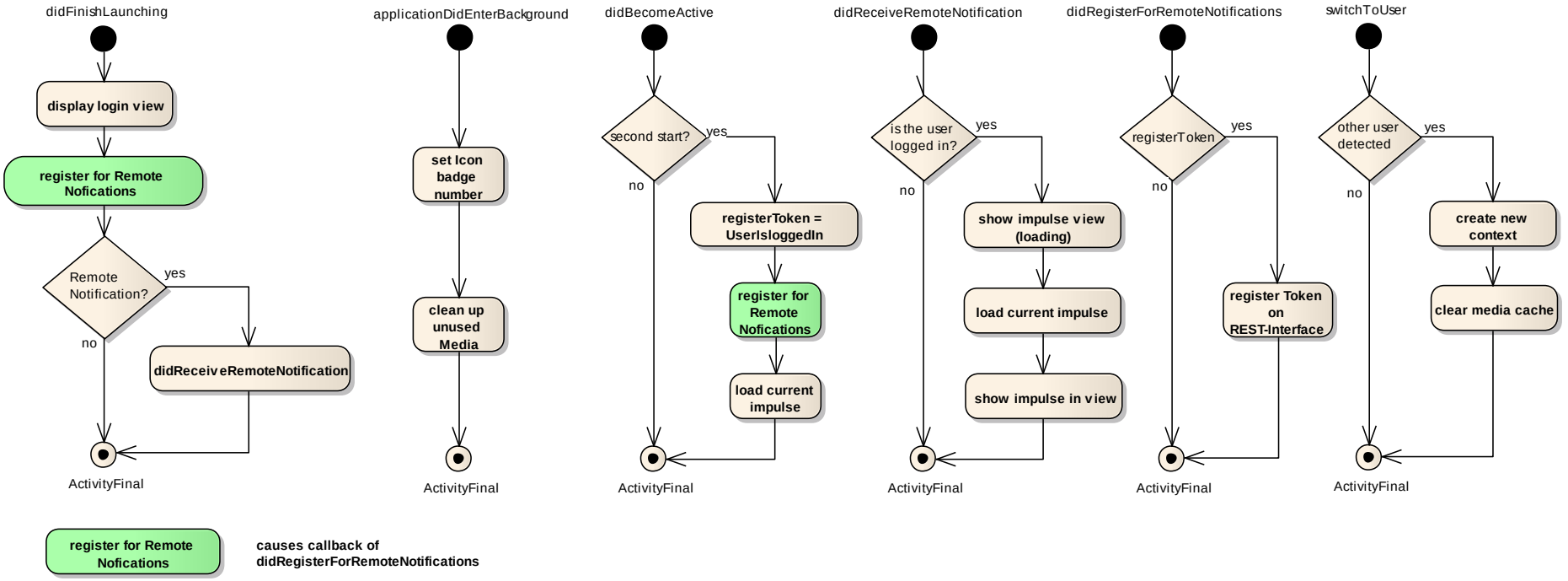

Abbildung 34: Ereignisse und Abläufe, die ausgelöst werden

didFinishLaunching
Die App registriert sich für Push Notifications. Dies ist ein asynchroner Aufruf. Der Callback löst `didRegisterForRemoteNotifications` aus.

Anschliessend wird der Login-Screen angezeigt. Findet dieser *Credentials* in der Key Chain, so wird der Benutzer automatisch eingeloggt.

Verhalten des `LoginViewControllers`:
Der `LoginViewController` hat noch etwas mehr Funktionalität (nicht im Diagramm). Sobald die View fertig geladen ist, wird überprüft, ob Login-Daten vorhanden sind. Falls ja, so loggt sich die App automatisch mit den Daten aus der Key Chain ein. Bei einem erfolgreichen Login registriert sich die App bei der REST-Schnittstelle mit `RegisterToken`.

applicationDidEnterBackground
Beim Ausblenden der App wird die Zahl des Icon Badge gesetzt. Dazu werden die ungelesenen Impulse gezählt und dann die Anzahl auf den Badge geschrieben. Wenn alle Impulse gelesen sind, wird die Zahl auf 0 gesetzt, wodurch der Icon Badge verschwindet.

Danach werden nicht mehr verwendete Medien aus dem *MediaCache* gelöscht.

didBecomeActive
Die Methode didBecomeActive wird aufgerufen, wenn die App in den Vordergrund kommt. Zum ersten Mal geschieht dies beim Starten der App. Wird die Methode zum zweiten Mal durchlaufen und ist der Benutzer eingeloggt, so wird `registerToken` gesetzt. Dies ermöglicht dem Callback des `registerForRemoteNotifications` das Registrieren des Device Tokens bei der REST-Schnittstelle. Der Device Token wird somit bei jedem Öffnen der App an die REST-Schnittstelle geschickt. Dies erscheint etwas viel, da aber Apple nirgends schreibt, wie lange ein Token gültig ist, macht das regelmässige Schicken Sinn.

Anschliessend werden die Impulse geladen.

didReceiveRemoteNotification
Sofern der Benutzer eingeloggt ist, wird die Impulse View angezeigt. Zu Beginn zeigt diese nur “Loading“ an. Dann werden die Impulse geladen und der aktuellste Impuls in der View dargestellt.

Hinweis: Ist der Benutzer beim Eingang einer Push Notification nicht angemeldet, so wird die App zwar geöffnet, die View aber nicht auf den aktuellen Impuls gewechselt.

didRegisterForRemoteNotifications
Sofern `registerToken` gesetzt ist, wird der Device Token bei der REST-Schnittstelle mit `RegisterToken` registriert. Die Registrierung bei der REST-Schnittstelle erfolgt nur, falls die App das zweite Mal aktiviert wird. Beim Aufstarten wird die Registrierung zusammen mit dem Login ausgeführt.

SwitchToUser
Loggt sich ein anderer Benutzer ein, so wird dies von der App detektiert: Es wird ein neuer `ManagedObjectContext` erzeugt und der `MediaCache` gelöscht.

8.11.1 Löschen der Benutzerdaten (Cache)

Bei einem Logout aus der App wird nur das Passwort gelöscht, der Benutzername und gespeicherte Impulse bleiben erhalten.

Meldet sich jedoch ein anderer Benutzer auf dem System an, so werden sämtliche Daten des vorherigen Benutzers gelöscht. Die Daten sind dann aber nicht verloren, sondern können jederzeit wieder vom Server bezogen werden. Einzige Ausnahme: Alle Impulse sind wieder ungelesen und auch die gelöschten werden wieder angezeigt. Damit nicht beliebig viele Impulse wieder gelöscht werden müssen, sollte die vom Server gelieferte Impulsanzahl z. B. auf 10 beschränkt werden.

8.11.2 Anzahl ungelesener Nachrichten (Icon Badge)

Das Programm-Icon der App ist, sofern es ungelesene Impulse hat, mit einem roten Badge und der Anzahl ungelesener Impulse ergänzt. Die Zahl wird aktualisiert, wenn die App in den Hintergrund wechselt. Beim Empfang einer Push Notification wird sie auf 1 gesetzt.

Dies entspricht in den meistens Fällen der Wahrheit, sofern der Benutzer seine Impulse immer sofort liest. Es wäre aber möglich, dass 5 Impulse noch ungelesen sind und eine Push Notification empfangen aber nicht angezeigt wird. Der Badge sollte dann 6 anzeigen, steht aber auf 1.

Um die Anzeige in jeder Situation konsistent zu halten, müsste der Server die Anzahl der ungelesenen Impulse auf dem iPhone ebenfalls kennen. Dadurch könnte er in der Push Notification die korrekte Zahl mitsenden, die als Badge angezeigt werden soll.

Dies ist eine kleine Unschönheit, die sich aber nur mit grösserem Aufwand beheben lässt (Anzahl ungelesene Nachrichten an Server schicken und dort ebenfalls verwalten). Sobald die App aber geschlossen wird, zählt sie die Anzahl Nachrichten und aktualisiert den Badge.

9 Ergebnisse und Gesamtfazit

9.1 Erreichte Ziele

Während der Bachelorarbeit wurde eine umfassende technische Analyse erstellt und die Implementierung einer iPhone App realisiert. Die iPhone App selbst deckt alle mit dem Auftraggeber vereinbarten Funktionen ab. Diese wurden während der Projektzeit Schritt für Schritt programmiert und anschliessend dem Auftraggeber präsentiert.
Um auch die Qualität der App zu gewährleisten, wurde kontinuierlich mit Unit-Tests gearbeitet. Am Ende wurde ein Systemtest durchgeführt, welcher dann auch das korrekte Funktionieren der App bestätigte.

In der App wurden die folgenden Funktionen realisiert:

- Der Benutzer kann sich Impulse anzeigen lassen. Beim Öffnen des Impuls' werden der Text und ein Media-Button (für Bild oder Video) angezeigt. Mit einem Tap auf den Button werden die Medien angezeigt.
- Sobald ein Bild zum ersten Mal angezeigt wird, ist es im Cache gespeichert. Impulse können so auch offline betrachtet werden.
- Sämtliche Impulse können in einer Inbox lokal verwaltet werden. Sobald ein Impuls vom Server geladen wurde, wird er persistiert und in der Inbox angezeigt. Mit einem Tap auf den Impuls kann der Impuls angezeigt werden. Für die Darstellung der Impulse wurde die Basisklasse `FetchedResultsTableViewController` erstellt. Views mit Listen können so recht einfach erstellt werden, da ein grosser Teil der benötigten Funktionalität schon in der Basisklasse realisiert ist.
- Damit der Benutzer auf neue Impulse aufmerksam gemacht werden kann, wird mit Push Notifications gearbeitet. Bei einer eingehenden Notification bleibt das Gerät still (kein Ton oder Vibration). Beim Entsperren des Geräts wird dem Benutzer die Notification angezeigt und er kann die App mit einem Tap auf View (Anzeigen) öffnen.
- Die Daten eines Benutzers sind durch einen Login geschützt und können so nur vom Benutzer selber eingesehen oder bearbeitet werden.
  Die Anmeldedaten selber sind in der *Key Chain* sicher abgelegt und werden für den automatischen Login verwendet. Damit die Benutzerdaten auf App-Ebene geschützt sind, muss sich der Benutzer abmelden[3].

Für den Betrieb und das Testen wurde ein Server eingerichtet. Der Server besteht aus den Teilen Servlets, *Push Sender* und *Media-Server*:

- Servlets: Die Servlets implementieren die REST-Schnittstelle zur Kommunikation mit dem iPhone. Die Verbindung zum Server, auf dem die Servlets gehostet werden, ist mit einem Zertifikat gesichert. Zum Benutzen der Servlets, muss sich der Benutzer mit Name und Passwort anmelden.
  - CurrentImpulse: Stellt eine Liste von Impulsen im JSON-Format zur Verfügung. Dieses Servlet ist in der Lage, einen Fehler zu simulieren. Mit dem simulierten Fehler konnte die App umfassender getestet werden.

[3] Falls die Code-Sperre aktiv ist, sind die Benutzerdaten bereits geschützt, ein Abmelden bei der App ist dann nicht notwendig.

    - RegisterToken: Mit RegisterToken kann das iPhone seinen Device Token ablegen, sodass es anschliessend für Push Notifications erreichbar ist.
    - SendPush: Dieses Servlet ist nicht Teil der REST-Schnittstelle und dient zum manuellen Auslösen von Impulsen. Es ersetzt daher im Wesentlichen die *Engine*, die Impulse selbständig verschicken kann.
- Push Sender: Der Push Sender verschickt die Push Notifications an das iPhone. Dazu verbindet er sich mit dem *APNs*.
- *Media-Server*: Der Media-Server stellt sämtliche Medien für die App zur Verfügung. Die Medien sind nicht geschützt und können über HTTP geladen werden.

9.2 Offene Punkte

Während der Arbeit sind noch ein paar offene Punkte aufgetaucht. Es handelt sich dabei um funktionale und strukturelle Änderungen und Erweiterungen.

Funktionale Änderungen

- Anzeigen der `ImpulseView` nach einer Push Notification: Falls der Benutzer nicht eingeloggt war, wird der verschickte Impuls nach dem Login nicht automatisch angezeigt. Der Impuls befindet sich jedoch zuoberst in der Inbox und ist als ungelesen markiert.
- Der Badge auf dem App Icon zeigt an, wie viele ungelesene Impulse ein Benutzer hat. Da das Lesen eines Impuls‘ dem Server nicht mitgeteilt wird, weiss er nicht, welche Impulse der Benutzer bereits gelesen hat. Deshalb wird beim Versand von Push Notifications der Badge jeweils auf 1 gesetzt. Dies stimmt dann, wenn der Benutzer seine Impulse regelmässig liest. Beim Verlassen der App wird der korrekte Wert gesetzt.

Strukturelle Änderungen

- REST-Protokoll: Das REST-Protokoll funktioniert einwandfrei. Serverseitig muss jedoch darauf geachtet werden, dass nicht zu viele Impulse verschickt werden. Bei einer schlechten Verbindung und zu vielen Impulsen wird die Ladezeit ziemlich lang. Es gibt verschiedene Möglichkeiten das Problem zu lösen:
    - Die Anzahl der Impulse, die der Server zur Verfügung stellt, könnte mit einem Maximum beschränkt werden.
    - Der Server schickt nur noch neue Impulse, die noch nie via REST-Schnittstelle geladen wurden. Möchte ein Benutzer wieder alle Impulse auf seinem z.B. neuen Gerät haben, so könnte er auf dem Server „Beim nächsten Verbinden der App alle Impulse zustellen“ einstellen.
    - Der Server schickt nur noch die Referenzen zu den Impulsen. Da so kein Text mehr verschickt werden muss, sind die Nachrichten um einiges kürzer.
- Die Impulse werden zurzeit alle mit dem `ImpulseViewController` dargestellt, egal ob es Bild- oder Videoimpulse sind. Nicht benötigte UI-Elemente werden ausgeblendet. Zusammen mit der Loading-Ansicht sind dies drei verschiedene Ansichten, die der `ImpulseViewController` steuert. Spätestens dann, wenn eine weitere Ansicht dazukommt, sollte alles separiert werden, um die Verantwortlichkeiten des Controllers zu verteilen und mehr Flexibilität bei der UI-Gestaltung der einzelnen Ansichten zu ermöglichen.

9.3 Bekannte Probleme und Einschränkungen

- Wenn der Medien-Server nicht verfügbar ist und es wird versucht, ein Video anzuzeigen, so friert der Player ein. Da der Timeout auf iOS standardmässig relativ lange eingestellt ist, steht der Player auch lange still (75 Sekunden). Ist das Gerät nicht mit dem Internet verbunden

(z.B. Airplane-Modus), so wird der Player sofort wieder geschlossen, die App kann danach weiter verwendet werden.
- Dasselbe Problem tritt auch beim Login ein.

Diese beiden Fälle treten eher selten auf, da der Server eigentlich immer verfügbar ist. Falls das Gerät nicht mit dem Internet verbunden ist, werden die Timeouts nicht abgewartet. Die Timeouts können wahrscheinlich eingestellt werden.

9.4 Ausblick

Die App hat einen guten Funktionsumfang und ist bereit für die Übergabe. Es wurde jedoch bereits während den Sitzungen über weitere Features diskutiert, die jedoch nicht zum Umfang der Bachelorarbeit gehören.

Mögliche neue Features

- Erinnerung für Impulse: Falls der Benutzer einmal keine Zeit hat, einen Impuls zu betrachten, kann er in der App eine Zeit einstellen, zu der er nochmals über den Impuls informiert werden möchte.
- Tagebuch-Funktion: Der Auftraggeber hatte ein mögliches Tagebuch erwähnt, bei dem der Benutzer seine Erfahrungen und Erkenntnisse aus dem Bereich Work Life Balance niederschreiben kann.

Weiteres Vorgehen

Crealogix wird nach der Übernahme des Projekts noch einige Änderungen umsetzen. Eine Liste aller möglichen Arbeitspakete befindet sich im Dokument [Arbeitspakete].
Nach dem Testen muss die App noch in den App-Store gebracht werden. Dies wird ebenfalls durch Crealogix erfolgen.

10 Glossar

Begriff	Definition
(Icon-) Badge	Roter Punkt auf dem Icon einer App. Die Zahl zeigt die Anzahl ungelesener Nachrichten (Impulse) an.
APNs	Apple Push Notification Service: Dieser Service verschickt eine Notification an den Benutzer (Dialogbox). Der User hat dann die Möglichkeit, die Notification zu akzeptieren. Nach dem Akzeptieren wird sogleich die App (mit passender App-ID) geöffnet, die Notification verarbeitet und das Ereignis / die Daten dem Benutzer angezeigt.
App	Kurzform für iOS Applikation - eine Anwendung für Apple iOS-Geräte (iPhone, iPod und iPad)
App ID	Die Applikations-ID erlaubt das eindeutige Identifizieren der App. Sie besteht aus Description, Bundle Seed ID und Bundle Identifier.
Cell Reuse	Ein Mechanismus, mit dem Zellen einer Tabelle wiederverwendet werden können, wenn sie nicht mehr im sichtbaren Bereich der Tabelle sind.
Cocoa	Cocoa ist die objektorientierte Programmierschnittstelle für Mac OS X und iOS
Credential Store	Ein Speicher in iOS Apps, in dem Logindaten für Netzwerkverbindungen während einer Sitzung gespeichert werden.
Credentials	Credentials sind Benutzername und Passwort.
Device Token	Der Token ist eine ID für ein iPhone oder ein anderes apns-fähiges Gerät. Sobald sich ein Gerät beim APNs registriert hat, ist es im Besitz eines gültigen Tokens. Beispiel: 6968a353e42f79145587968019012345646317e2e925407c10234a5fdf8bfd373ea473d96d
Engine	Die Engine verschickt Impulse. Auf Grund der erfassten Benutzerdaten entscheidet sie selbständig, welcher Benutzer wann, welche Impulse erhält. Ein Benutzer erhält meistens einen Impuls pro Tag.
Funktionssatz	Ein Funktionssatz beinhaltet verschiedene Teilfunktionen. Während der Bachelorarbeit wird der erste Funktionssatz “FU01: Anzeigen von Impulsen inklusive deren lokale Speicherung“ umgesetzt.
Impuls	Ein Impuls ist eine Meldung, die dem Benutzer zugestellt wird. Der Impulse enthält eine Nachricht (Text) und ist mit Medien ergänzt: Bilder oder Video.
Interface Builder	Ein grafischer Editor, um Benutzeroberflächen für iOS-Applikationen zu erstellen. Der Interface Builder ist ein Teil der XCode Entwicklungsumgebung.
Key Chain	Die Key Chain ist ein sicherer Ort für Login-Daten.
Media-Server	Teil des Centrado Servers, der Bilder und Videos hostet.
Objective-C	C-ähnliche Programmiersprache, siehe 6.1.2 Objective-C
Outlet	Ein speziell markiertes Property in einem *View Controller*, auf das der Interface Builder ein UI-Element binden kann.
Provisioning Profile	Das Provisioning Profile ist Teil des Development-Prozesses. Es sagt aus, welche App ID für welche Geräte ist und von welchen Developern entwickelt (signiert) werden darf.
Public Hot Spot	Public Hot Spots sind öffentliche WLANs.
Push Notification	Meldung vom APNs an ein Gerät. Push Notifications können mit Erlaubnis des Benutzers Apps starten.

Push Sender	Der Push Sender verbindet sich mit dem APNs und verschickt Push Notifications an die Geräte.
REST-Schnittstelle	Die REST-Schnittstelle liefert sämtliche Impulse. Die Geräte können ihren Token dort ablegen.
Status Bar	Oberster Abschnitt des iPhone Bildschirms, in dem die Netzqualität, die Uhrzeit und der Batteriezustand angezeigt werden.
View Controller	Ein Controller wird im Cocoa MVC „View Controller“ genannt, um ihn von anderen Controllern zu unterscheiden.
Work Life Balance	Der Begriff Work-Life-Balance steht für einen Zustand, in dem Arbeit und Privatleben miteinander in Einklang stehen (aus [WLB]).
XCode	Entwicklungsumgebung für Mac und iOS-Geräte.

11 Literaturverzeichnis

Referenz	Quelle
[APNsPHP]	David Mytton: How to build an Apple Push Notification provider server (tutorial) http://blog.boxedice.com/2009/07/10/how-to-build-an-apple-push-notification-provider-server-tutorial/ (Stand 11.6.11)
[Arbeitspakete]	Dokument: 03_Arbeitspakete aus Unfuddle.xlsx
[Aufgabenstellung]	Dokument: BA-EugsterKoella-CentradoApp-Aufgabenstellung.docx (H. Rudin)
[BackExec]	http://developer.apple.com/library/ios/#documentation/iphone/conceptual/iphoneosprogrammingguide/BackgroundExecution/BackgroundExecution.html
[BasicAuth]	Dokument: Quellen/Basic Auth in Tomcat.docx
[BdAttk]	http://en.wikipedia.org/wiki/Birthday_attack (Stand 10.6.11)
[CocoaMVC]	Cocoa Fundamentals Guide – Kapitel: Cocoa Design Patterns; Seite 163 ff; Apple; Version 2010-11-15 http://developer.apple.com/library/ios/#documentation/Cocoa/Conceptual/CocoaFundamentals/CocoaDesignPatterns/CocoaDesignPatterns.html
[iAPG]	iPhone Application Programing Guide; Apple; Revision 2011-02-24 http://developer.apple.com/library/ios/documentation/iphone/conceptual/iphoneosprogrammingguide/iPhoneAppProgrammingGuide.pdf (Stand 10.6.2011)
[POSA96]	Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal; Pattern Oriented Software Architecture – A System of Patterns; 1996
[ProtoW2]	Dokument: Protokolle/Protokoll110301.docx
[PSCO]	http://developer.apple.com/library/mac/#documentation/DataManagement/Devpedia-CoreData/persistentStoreCoordinator.html#//apple_ref/doc/uid/TP40010398-CH27-SW1
[RESTFul]	http://de.wikipedia.org/wiki/Representational_State_Transfer
[SHA1]	http://en.wikipedia.org/wiki/SHA-1 (Stand 10.6.11)
[Statuscode]	http://de.wikipedia.org/wiki/HTTP-Statuscode
[UNIXZeit]	http://de.wikipedia.org/wiki/Unixzeit
[WLB]	http://de.wikipedia.org/wiki/Work-Life-Balance

12 Abbildungsverzeichnis

13 Anhang

13.1 JSON-Dokument mit zwei Impulsen


```
{
 "items": [
  {
   "single": false,
   "soundURL": null,
   "imageURLs": [

   ],
   "timestamp": 1306424299,
   "duration": 5,
   "title": "Atem Pause",
   "description": "Schauen Sie sich den Video an.",
   "webURL": null,
   "action": "Ausführen",
   "language": "DE",
   "descriptionShort": "Videoimpulse Atempause",
   "workingID": "vid",
   "videoURL": "http:\/\/152.96.56.36\/StaticContent\/Impulse\/AtemPause\/mov-e1-
0004d_iPhone(Cellular).3gp"
  },
  {
   "single": true,
   "soundURL": null,
   "imageURLs": [
    "http:\/\/152.96.56.36\/StaticContent\/Impulse\/Nacken\/img-k3-0001-1.png",
    "http:\/\/152.96.56.36\/StaticContent\/Impulse\/Nacken\/img-k3-0001-2.png",
    "http:\/\/152.96.56.36\/StaticContent\/Impulse\/Nacken\/img-k3-0001-3.png"
   ],
   "timestamp": 1306424307,
   "duration": 3,
   "title": "Dehnung der Nackenmuskeln",
   "description": "Führen Sie folgende Übung anhand der Bilder aus und verbleiben Sie in der jeweiligen Position ca. 20 Sekunden. Nehmen Sie eine aufrechte und entspannte Sitzhaltung ein. Führen Sie Ihr rechtes Ohr Richtung rechte Schulter. Atmen Sie entspannt, lassen Sie die Schultern locker fallen. Geniessen Sie die Dehnung der linken Nackenmuskulatur. Zählen Sie langsam auf 23 und verbleiben Sie so lange in dieser Haltung. Führen Sie Ihren Kopf zur Mitte zurück und dehnen Sie die andere Seite.",
   "webURL": null,
   "action": "Ausführen",
   "language": "DE",
   "descriptionShort": "Diese Übung dient zur Entspannung der Nackenmuskulatur. Führen Sie die Übung  anhand der Bilder aus.",
   "workingID": "pic",
   "videoURL": null
  }
 ]
}
```

13.2 Lizenz zu java-apns

```
Copyright 2009, Mahmood Ali.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

  * Redistributions of source code must retain the above copyright
    notice, this list of conditions and the following disclaimer.
  * Redistributions in binary form must reproduce the above
    copyright notice, this list of conditions and the following disclaimer
    in the documentation and/or other materials provided with the
    distribution.
  * Neither the name of Mahmood Ali. nor the names of its
    contributors may be used to endorse or promote products derived from
    this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

13.3 Lizenz zu TouchJSON

```
Copyright (c) 2008 Jonathan Wight

Permission is hereby granted, free of charge, to any person
obtaining a copy of this software and associated documentation
files (the "Software"), to deal in the Software without
restriction, including without limitation the rights to use,
copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the
Software is furnished to do so, subject to the following
conditions:

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
```

```
WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
OTHER DEALINGS IN THE SOFTWARE.
```

13.4 Lizenz zu KeyChain und SecurityService Klassen

```
Copyright (c) 2010 Dan Byers

Permission is hereby granted, free of charge, to any person
obtaining a copy of this software and associated documentation
files (the "Software"), to deal in the Software without
restriction, including without limitation the rights to use,
copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the
Software is furnished to do so, subject to the following
conditions:

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
OTHER DEALINGS IN THE SOFTWARE.
```