SnapIt: Effizienter und Fehlertoleranter Foto-Upload auf Android Mobilsystemen

Studienarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Herbstsemester 2010

Autoren: Christoph Süess, Raphael Nagy
Betreuer: Prof. Dr. Markus Stolze
Projektpartner: IFS

1 Abstract

Abteilung	Informatik
Namen der Studierenden	Christoph Süess, Raphael Nagy
Studienjahr	HS 2010
Titel der Studienarbeit	SnapIt: Effizienter und Fehlertoleranter Foto-Upload auf Android Mobilsystemen
Examinatorin / Examinator	Prof. Dr. Markus Stolze
Themengebiet	Internet-Technologien und –Anwendungen / Kommunikationssysteme / Software
Projektpartner	IFS
Institut	Institut für Software

Kurzfassung

Seit mehreren Jahren haben sich in der Schweiz online Partyportale etabliert. Der Schwerpunkt ihrer Tätigkeit liegt beim Publizieren von hochwertigen Party- und Eventfotos. Diese Bilder werden von semiprofessionellen Fotografen geschossen und nach dem jeweiligen Anlass auf der Webseite veröffentlicht.

Weltweit werden täglich 300‘000 Android-Smartphones, mit meist integrierter Kamera, aktiviert. Es ist abzusehen, dass auch in der Schweiz diese Geräte damit zum Fotoapparat der Wahl werden. Mit sich ständig verbessernden Foto-Chips wird es möglich, dass Party-Fotos nun von „jedermann“ mit dem Smartphone geschossen werden und Party-Portale mit benutzergenerierten Inhalten gefüllt werden können. Dies hat diverse Vorteile gegenüber den bestehenden Partyportalen. Der Hauptvorteil liegt darin, dass die Fotos schon nach kurzer Zeit auf der Webseite abrufbar sind. Ideal wäre hierbei wenn die Fotos automatisch einer Party- oder Eventlokation zugeordnet würden. Diese wird anhand der übermittelten Geo-Koordinaten bestimmt.

Die technische Voraussetzung für einen solchen Service ist eine genaue Lokalisierung des Telefons (mittels GPS) und einer effizienten und fehlertoleranten Übertragung der Fotos vom Gerät auf das Party-Portal. In dieser Studienarbeit wurde mit SnapIt ein System entwickelt welches prototypisch für Andoid-Smartphones demonstriert, wie sich Bilder effizient, fehlertolerant und anonym von Mobiltelephonen übermitteln lassen. Es wurde auch analysiert wie gut die Geo-Lokalisierung funktioniert, und welche Herausforderungen sich aus der momentan noch ungenügenden Geo-Lokalisierung der Telefone ergeben.

Der zentrale Bestandteil der Arbeit war das Implementieren einer soliden und stabilen Server-Client-Kommunikation, welche sich um die Problematik des Verbindungsunterbruchs kümmert. Die daraus resultierende Programmbibliothek stützt sich auf dem REST-Konzept ab und kann problemlos auch für weitere Projekte verwendet werden. Die Java-Serverapplikation ist für das Verarbeiten der Fotos und

der Geodaten verantwortlich. Die Serviceschnittstelle wird auf dem Server mit dem Jersey-REST-Framework realisiert. Mit Hilfe der JavaScript-Library jQuery werden die Bilder dynamisch auf der Beispiel-Webseite angezeigt.

2 Management Summary

Seit mehreren Jahren haben sich in der Schweiz online Partyportale etabliert. Der Schwerpunkt ihrer Tätigkeit liegt beim Publizieren von hochwertigen Party- und Eventfotos. Diese Bilder werden von semiprofessionellen Fotografen geschossen und nach dem jeweiligen Anlass auf der Webseite veröffentlicht. Die Fotos werden mit professionellen digitalen Spiegelreflexkameras mit aufgesetztem Blitz geschossen, daher werden die Bilder sehr scharf und weisen eine hohe Auflösung auf. Des Weiteren sind darauf häufig nur Personen oder Personengruppen zu sehen. Der Fotograf wählt dabei seine Fotomotive selber aus. Er versucht gezielt das Geschehen in ein gutes Licht zu rücken. Während der Nachbearbeitung löscht der Fotograf, die in seinen Augen nicht gelungene Aufnahmen. Dadurch wird nur eine Sicht auf die Party gezeigt. Ein weiteres Problem ist, dass meist nur die angesagtesten Clubs bedient werden. Kleine oder weit vom Stadtzentrum entfernte Clubs werden oft gar nie von den Fotografen besucht. Durch das manuelle Hochladen der Bilder gibt es bis zu einem Tag Verzögerung, bis die Fotos im Internet verfügbar sind.

Abbildung 2-1: SnapIt-Applikation

Weltweit werden täglich 300‘000 Android-Smartphones, mit meist integrierter Kamera, aktiviert. Es ist abzusehen, dass auch in der Schweiz diese Geräte damit zum Fotoapparat der Wahl werden, da schon jetzt jedes dritte Mobiltelefon ein Smartphone ist. Mit sich ständig verbessernden Foto-Chips wird es möglich, dass Party-Fotos nun von jedermann mit dem Smartphone geschossen werden und Party-Portale mit benutzergenerierten Inhalten gefüllt werden. Dies hat diverse Vorteile gegenüber den bestehenden Partyportalen. Der Hauptvorteil liegt darin, dass die Fotos schon nach kurzer Zeit auf der Webseite abrufbar sind. Zudem kann bei benutzergenerierten Inhalten viel mehr und vor allem eine grössere Anzahl Clubs abgedeckt werden. Jeder hat dann die Möglichkeit kreativ und gestalterisch mitzuwirken, indem er oder sie eigene Bilder knipst und hoch lädt. Ideal wäre hierbei auch, wenn die Fotos automatisch einer Party- oder Eventlokation zugeordnet würden. Diese wird anhand der übermittelten Geo-Koordinaten bestimmt.

Die technische Voraussetzung für einen Service, der anhand Geo-Koordinaten die Lokation beziehungsweise den Club bestimmen kann und dann Fotos zu dieser Lokation zuordnet, ist eine genaue Lokalisierung des Telefons (mittels GPS oder alternativen Technologein) und einer effizienten und fehlertoleranten Übertragung der Fotos vom Gerät auf das Party-Portal. In dieser Studienarbeit wurde mit SnapIt ein System entwickelt welches prototypisch für Andoid-Smartphones demonstriert, wie sich Bilder effizient, fehlertolerant und anonym von Mobiltelephonen auf einen Server übermitteln lassen. Es wurde auch analysiert wie gut die Geo-Lokalisierung funktioniert, und welche

Herausforderungen sich aus der momentan noch ungenügenden Geo-Lokalisierung der Telefone ergeben.

Der zentrale Bestandteil der Arbeit war das Implementieren einer soliden und stabilen Server-Client-Kommunikation, welche sich um die Problematik des Verbindungsunterbruchs kümmert. Die daraus resultierende Programmbibliothek stützt sich auf dem REST-Konzept ab und kann problemlos auch für weitere Projekte verwendet werden. Die Java-Serverapplikation ist unter anderem für das Verarbeiten der Fotos und der Geodaten verantwortlich. Es werden zudem kleinere Vorschaubilder erstellt und abgespeichert, welche für verschiedene Anzeigemedien gebraucht werden können. Die Serviceschnittstelle wird auf dem Server mit dem Jersey-REST-Framework realisiert. Dieses Framework bietet eine einfache und stabile Implementation des JSR 311. Mit Hilfe der JavaScript-Library jQuery werden die Bilder dynamisch auf der Beispiel-Webseite angezeigt und durch verschiedene Einstellungen wird die vielfältige Konfigurierbarkeit der Serviceschnittstelle in ihren Grundzügen präsentiert.

Abbildung 2-2: SnapIt-Screen: Laden der Lokationen

Das System, welches während dieser Studienarbeit implementiert wurde, bildet den Grundstein für ein zukünftiges Partyportal. Während der Arbeit entstanden aber auch viele Komponenten, die sich für andere Zwecke einsetzen lassen. So ist das zuverlässige Übermitteln von Fotos auf einen Server nicht nur zur Dokumentation des Nachtlebens einsetzbar. Es ist z. B. denkbar, dass Sicherheitspersonal mit solchen Kameras bestimmte Situationen dokumentieren. Hier käme ein bis jetzt noch nicht erwähnter Vorteil zum Einsatz: Übermittelte Bilder können nicht ohne weiteres gelöscht werden, würden wir den entsprechenden Webservice nicht anbieten. Das Zerstören des Aufnahmegeräts würde also das Foto nicht beschädigen.

Mit SnapIt wurden alte Ideen mit neuen Technologien implementiert. So entstand eine innovative und qualitative Android-App.

3 Inhaltsverzeichnis

4 Aufgabenstellung

4.1 Titel

SnapIt: Effizienter und Fehlertoleranter Foto-Upload auf Android Mobilsystemen

4.2 Auftraggeber und Betreuer

Diese Studienarbeit hat keinen direkten Auftraggeber. Als interimistischer Auftraggeber fungiert das Institut für Software.

Ansprechpartner Auftraggeber:

(NA)

Betreuer HSR:

Prof. Dr. Markus Stolze, Institut für Software mstolze@hsr.ch

4.3 Ausgangslage

Smartphones erfreuen sich einer immer grösser werdenden Beliebtheit. Viele dieser Geräte haben eine integrierte Kamera. In dieser SA soll exploriert werden ob sich Fotos welche mit diesen Kameras geschossen wurden auch nutzen lassen um aktuelle Eindrücke von einer Party an potentielle Partybesucher zu übermitteln.

SnapIt ist eine Anwendung welche Partygängern erlaubt Fotos welche mit einem Smartphone (iPhone oder Android) geschossen wurden live und mit Ortsangaben auf einer Webseite hochzuladen. Andere Partygänger können sich hier informieren „was läuft“

4.4 Ziele der Arbeit

In dieser Arbeit soll in einem ersten Schritt eine detaillierte Analyse der Bedürfnisse der zukünftigen Nutzer (Fotoschiesser und Fotobrowser) durchgeführt werden und eine saubere Konkurrenzanalyse von ähnlichen Systemen gemacht werden (Facebook, Foursquare, Tillate, Usgang, Lautundspitz). In dieser Analyse sind auch die sinkenden Benutzerzahlen der Party-Portale zu beachten (http://www.thomashutter.com/index.php/2009/07/killen-social-networks-dieparty-portale/)

In einem zweiten Schritt soll ein funktionsfähiger Prototyp einer Anwendung entwickelt werden. Die Usability der Anwendung soll mittels Usability Tests geprüft werden. Die Wahl der Plattform (iPhone / Android) bleibt dabei den Studenten überlassen.

4.5 Zur Durchführung

Mit den HSR-Betreuern finden in der Regel wöchentliche Besprechungen statt. Zusätzliche Besprechungen sind nach Bedarf durch die Studierenden zu veranlassen.

Alle Besprechungen sind von den Studenten mit einer Traktandenliste vorzubereiten und die Ergebnisse in einem Protokoll zu dokumentieren, das dem Betreuer E-Mail zugestellt wird. Zudem wird eine Projektseite ausgehend von Prof. Stolzes internen SA 2010 Wiki erstellt:

http://sinv0002.hsr.ch/MarkusStolze/wiki.cgi?StolzeArbeitenHS2010

Für die Durchführung der Arbeit ist ein Projektplan zu erstellen. Dabei ist auf einen kontinuierlichen und sichtbaren Arbeitsfortschritt zu achten. An Meilensteinen gemäss Projektplan sind einzelne Arbeitsresultate in vorläufigen Versionen abzugeben. Über die abgegebenen Arbeitsresultate erhalten die Studierenden ein vorläufiges Feedback. Eine definitive Beurteilung erfolgt aufgrund der am Abgabetermin abgelieferten Dokumentation. Die Evaluation erfolgt aufgrund des separat abgegebenen Kriterienkatalogs in Übereinstimmung mit den Kriterien zur SA Beurteilung. Es sollten hierbei auch die Hinweise aus dem abgegebenen Dokument „Tipps für die Strukturierung und Planung von Studien-, Diplom- und Bachelorarbeiten“ beachtet werden.

4.6 Dokumentation

Über diese Arbeit ist eine Dokumentation gemäss den Richtlinien der Abteilung Informatik zu verfassen. Die zu erstellenden Dokumente sind im Projektplan festzuhalten. Alle Dokumente sind nachzuführen, d.h. sie sollten den Stand der Arbeit bei der Abgabe in konsistenter Form dokumentieren. Die Dokumentation ist vollständig auf CD/DVD in 2 Exemplaren abzugeben. Zudem ist eine kurze Projektresultatdokumentation im Wiki von Prof. Stolze zu erstellen dies muss einen Link auf einen YouTube Video enthalten welche das Resultat der Arbeit dokumentiert.

4.7 Termine

Siehe auch Terminplan auf

https://www.hsr.ch/Termine-Diplom-Bachelor-und.5142.0.html

20.09.10 Beginn der Studienarbeit, Ausgabe der Aufgabenstellung durch die Betreuer.

20.12.10 Die Studierenden senden folgende Dokumente der Arbeit per Email zur Prüfung an ihre Betreuer:

- Abstract/Kurzfassung
- A0-Poster

Vorlagen stehen unter den allgemeinen Infos Diplom-, Bachelor und Studienarbeiten zur Verfügung.

23.12.10 Die Studierenden senden das vom Betreuer abgenommene und freigegebene Abstract/Kurzfassung als Word-Dokument an das Studiengangsekretariat (cfurrer(at)hsr.ch).

23.12.10 Abgabe des Berichtes an den Betreuer bis 17.00 Uhr.

4.8 Beurteilung

Eine erfolgreiche SA zählt 8 ECTS-Punkte pro Studierenden. Für 1 ECTS Punkt ist eine Arbeitsleistung von ca. 25 bis 30 Stunden budgetiert. Entsprechend sollten ca. 240h Arbeit für die Studienarbeit aufgewendet werden. Dies entspricht ungefähr 17h pro Woche (auf 14 Wochen) und damit ca. 2 Tage Arbeit pro Woche.

Für die Beurteilung ist der HSR-Betreuer verantwortlich unter Einbezug des Feedbacks des Auftraggebers (welches in diesem Fall entfällt). Die Bewertung der Arbeit erfolgt entsprechend der verteilten Kriterienliste.

5 Projektplan

5.1 Vision

SnapIt sollte es Nachtschwärmern ermöglichen während dem Ausgang Fotos mit ihrem Smartphone zu knipsen und diese direkt im Internet zu veröffentlichen. Dabei werden die Fotos automatisch der Lokation zugeordnet, in welcher sich der Benutzer befindet. Nutzt eine grosse Menge an Personen diese Applikation, entsteht im Internet eine Plattform die das Nachtleben widerspiegelt.

Diese Personen können davon profitieren, indem sie die Fotos von anderen im Internet betrachten und so die Stimmung in den verschiedenen Clubs, Bars und Restaurants erahnen können. SnapIt bietet so auch die Möglichkeit in unbekannten Regionen die passende Ausgangslokation zu finden.

Social-Platforms beweisen, dass Benutzer interessiert sind private Fotos mit der Öffentlichkeit zu teilen. Partyportale zeigen das Bilder von Partys im Internet betrachtet werden. Diese zwei Erkenntnisse verbindet SnapIt mit neuen Möglichkeiten der heutigen Technologien.

SnapIt interessiert sich nicht für die privaten Daten der Benutzer. Durch eine anonyme Anbindung an den Server kann über die gespeicherten Informationen kein Bezug auf den Benutzer, der hinter den Bildern steht, gemacht werden.

Technisch ist besonders die REST-Architektur interessant, welche problemlos auch in anderen Projekten verwendet werden kann.

5.2 Domainmodel

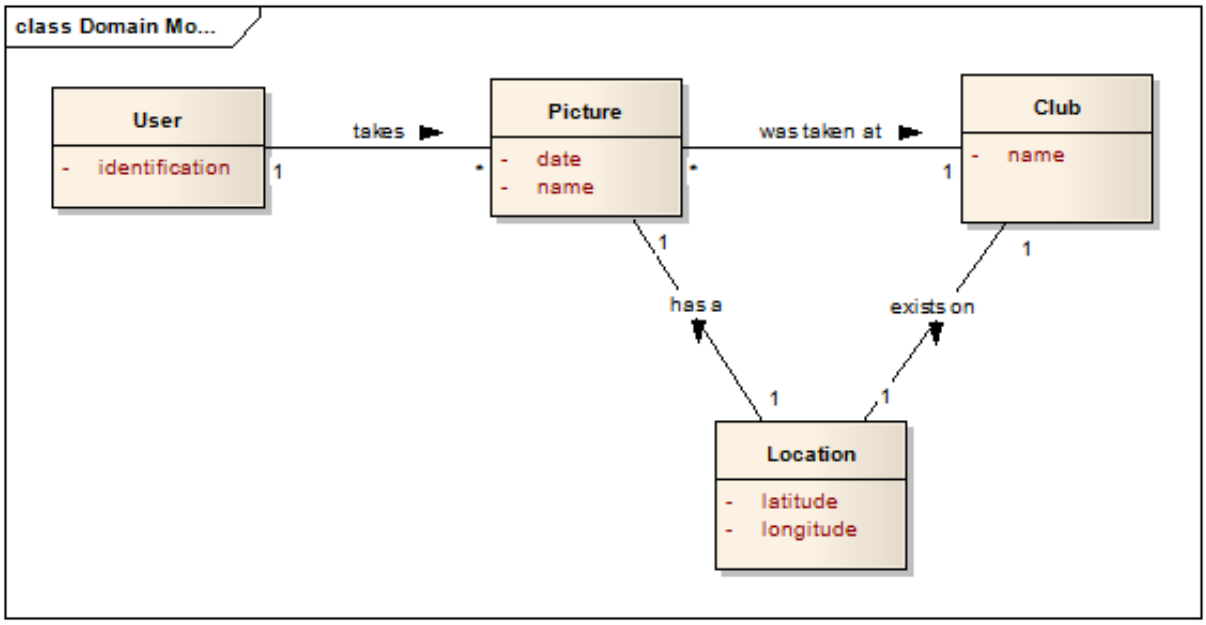

Abbildung 5-1: Domainmodel

- Benutzer knipst Fotos
- Jedem Foto ist eine Lokation zugeordnet. Eine Lokation besteht aus Länge- und Breitengrad.

- Zusätzlich wird ein Foto einem Club zugeordnet

5.3 Anforderungsspezifikationen

5.3.1 Produkt Perspektive

SnapIt soll eine Social-Community-Plattform werden, welche es jedem Nutzer ermöglicht Bilder mit seinem Android-Mobiltelefon aufzunehmen und anonym diese der Allgemeinheit zur Verfügung zu stellen. Jedes Foto wird dabei von unserer Serversoftware einer Lokation zugeordnet. Weitere Details sind der Aufgabenstellung zu entnehmen.

5.3.2 Produkt Funktionen

Folgende Funktionen soll das Produkt aufweisen:

- Fotos schiessen
- Fotos auf eine Webseite hochladen
- Fotos verwalten (relokalisieren und löschen)
- Eigene Fotos auf dem Handy betrachten
- Alle Fotos auf einer Beispiel-Webseite betrachten
- Fotos einer Lokation bzw. Club zuordnen

5.3.3 Benutzer Charakteristik

Benutzer, welche die Applikation SnapIt verwenden sind voraussichtlich junge Leute zwischen 18-30 Jahren oder jung gebliebene Leute, die gerne in den Ausgang gehen und Fotos schiessen. Sie besitzen ein Android-Mobiltelephon und haben ein Datenabonnement von einem Mobilfunknetzbetreiber.

Für die Webseite ist der Benutzerkreis grösser einzuschätzen, da keine eigene Aktion nötig ist, um Bilder zu betrachten. Hier nehmen wir an, dass die Webseite vermehrt von allgemeinen "Surfern" betrachtet wird.

5.3.4 Annahmen

Es müssen keine Annahmen getroffen werden, da die Auftraggeber des Projektes auch die Projektmitarbeiter sind. Technische Unklarheiten werden am Anfang bei jedem Sprint in der Analyse-Phase geklärt.

5.3.5 Spezifische Anforderungen

Sicherheit

Auf Sicherheit wird kein besonderer Wert gelegt, da wir grundsätzlich nur mit anonymen Daten arbeiten.

Verständlichkeit

Unser Ziel ist es die Applikation möglichst einfach zu halten. Einerseits in der Bedienung und andererseits im Code. Durch diese Einfachheit sollten nicht übermässig viele Verständlichkeitsprobleme auftreten beim Benutzer bzw. bei der Wartung der Software.

Erlernbarkeit

Die Applikation sollte intuitiv bedienbar sein. Es gibt ein Produkte-Video auf, welches die wichtigsten Funktionen demonstriert. Auch ein Handbuch erläutert diese Features.

Bedienbarkeit

Es kann davon ausgegangen werden, dass das Produkt SnapIt von Personen genutzt wird, welche Erfahrungen in der Nutzung von Applikationen auf der Android Plattform haben.

Des Weiteren sollte das Produkt folgende Eigenschaften aufweisen:

- Ohne Bedienungsanleitung bedienbar sein: Die Applikation soll sich nicht zu sehr von anderen Android Applikationen unterscheiden. Deshalb werden die Android "User Interface Guidelines" versucht einzuhalten. (http://developer.android.com/guide/topics/ui/index.html)
- Mit dem Finger bedienbar (Touch): Dem Benutzer muss es möglich sein die Oberfläche nur mit dem Finger zu bedienen. Dabei muss auf die Grösse der Schaltflächen und Eingabefelder geachtet werden.

Zuverlässigkeit

Die Zuverlässigkeit ist stark von der Hosting-Umgebung für den Webserver und die Datenbank abhängig. Die Verfügbarkeit liegt im Verantwortungsbereich des Hosters.

Internet-Verbindung

Wir legen viel Wert auf zuverlässige Aufrufe der REST-Webservices. Hierzu werden wir die Applikation möglichst robust bauen. Weitere Details sind in den Architektur-Kapiteln nachzulesen.

Leistung

Die Leistung der Client-Applikation ist stark von der Datenübertragungsrate abhängig. Für die nachfolgenden Leistungsanforderungen gehen wir davon aus, dass der Benutzer sich in einem 3G Netz mit guter Verbindung befindet. Weiter rechnen wir mit maximal fünf gleichzeitigen Benutzerzugriffen. Die Anzahl Fotos in der Datenbank liegt bei ca. 10‘000.

Leistungsanforderungen

Die Android-Applikation soll immer reagierend (responsive) bleiben und in maximal sechs Sekunden starten. Innerhalb der Applikation soll bei jedem Klick des Benutzers innerhalb von einer Sekunde eine neue Ansicht erscheinen, falls dies nicht der Fall ist wird der Benutzer mit einem Ladebalken informiert.

Konfigurierbarkeit

SnapIt kann vom Benutzer nicht konfiguriert werden.

Mehrsprachigkeit

Die gesamte Applikation wurde für die deutsche Sprache entwickelt. Mehrsprachigkeit ist vorerst nicht vorgesehen. Durch das Arbeiten mit Ressource-Dateien, kann auch einfache Weise eine neue Sprache hinzugefügt werden.

Architektur

Die Architektur auf der Handyseite (Clientapplikation) wird vor allem durch das Android-Framework bestimmt, jedoch legen wir besonders grossen Wert auf eine saubere REST Implementation.

Auf Serverseite wird eine einfach wartbare REST-Implementation erstellt und für die Webseite setzen wir auf eine Webtechnologie, welche noch in dem entsprechenden Sprint evaluiert wird.

Update

Die Updatefähigkeit für die Client-Applikation ist automatisch durch den App Store von Google gewährleistet.

Schnittstellen

Benutzerschnittstelle

Eine Android-Applikation dient als Benutzerschnittstelle. Die Benutzeroberfläche wird in dem entsprechenden Sprint noch genauer spezifiziert.

Softwareschnittstelle

Als Softwareschnittstelle zwischen dem SnapIt-Server und dem SnapIt-Client (Android) wird ein REST-Webservice erstellt.

Datenbankschnittstelle

Als Datenbankschnittstelle wird Hibernate/JDBC eingesetzt, um auf die PostgreSQL Datenbank zuzugreifen. Eine genauere Analyse erfolgt im entsprechenden Sprint.

5.4 Vorgehen

Das Projekt wurde nach dem SE-Vorgehensmodel SCRUM durchgeführt. Aus diversen Gründen sind Elemente von anderen Vorgehensmodell miteinbezogen worden, siehe dazu Abschnitt „Lessons Learned“.

5.4.1 Sprints

Die Risiken der einzelnen Sprints sind dem Abschnitt „Risikomanagement“ zu entnehmen.

Sprint 1

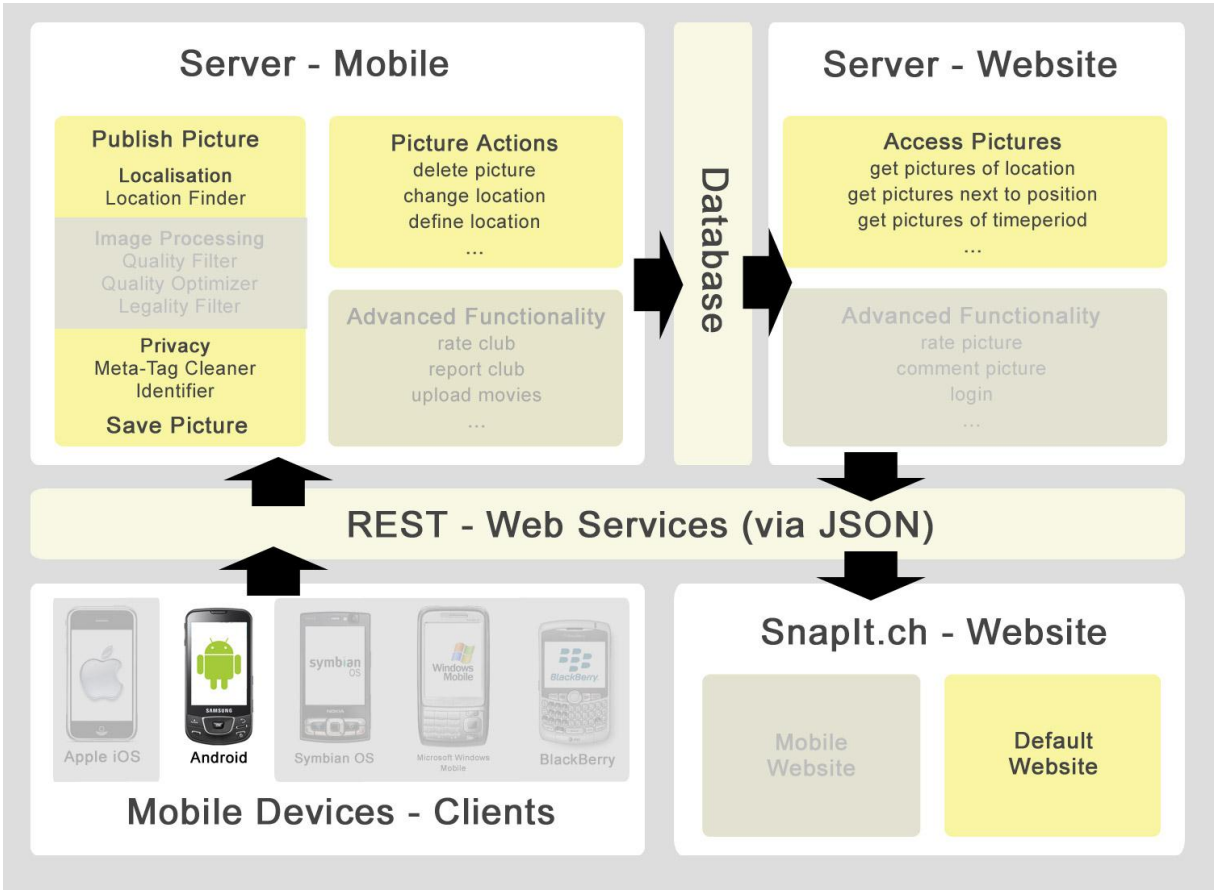

Abbildung 5-2: Projektübersicht

Beschreibung

- **User Story:** Keine User-Story
- **Iterationszeitraum:** 1.-3. Woche (20.09. - 04.10.)
- **Abnahme:** 14.10.2010

Während der ersten Iteration wird das Projekt geplant, die Aufgabe genau analysiert und die Rahmenbedingungen festgelegt. Anhand von Analysen werden die Chancen am Markt und die Konkurrenz ermittelt und dokumentiert. Die zu implementierenden Features werden anhand der User Stories festgelegt und als Use Cases formuliert.

Abnahmetest

Die Initialdokumente werden vom Projektbetreuer Herr Prof. Dr. Stolze abgenommen.

Chores

- SVN aufsetzen
- Entwicklungsumgebung installieren
- MediaWiki installieren

Resultierende Dokumente

- Projektplan (Initial-Dokument)
- Risikomanagement (Initial-Dokument)
- Use Cases
- User Stories
- Markt Analyse

- Konkurrenz Analyse (Initial-Dokument)
- Glossar (Initial-Dokument)

Sprint 2

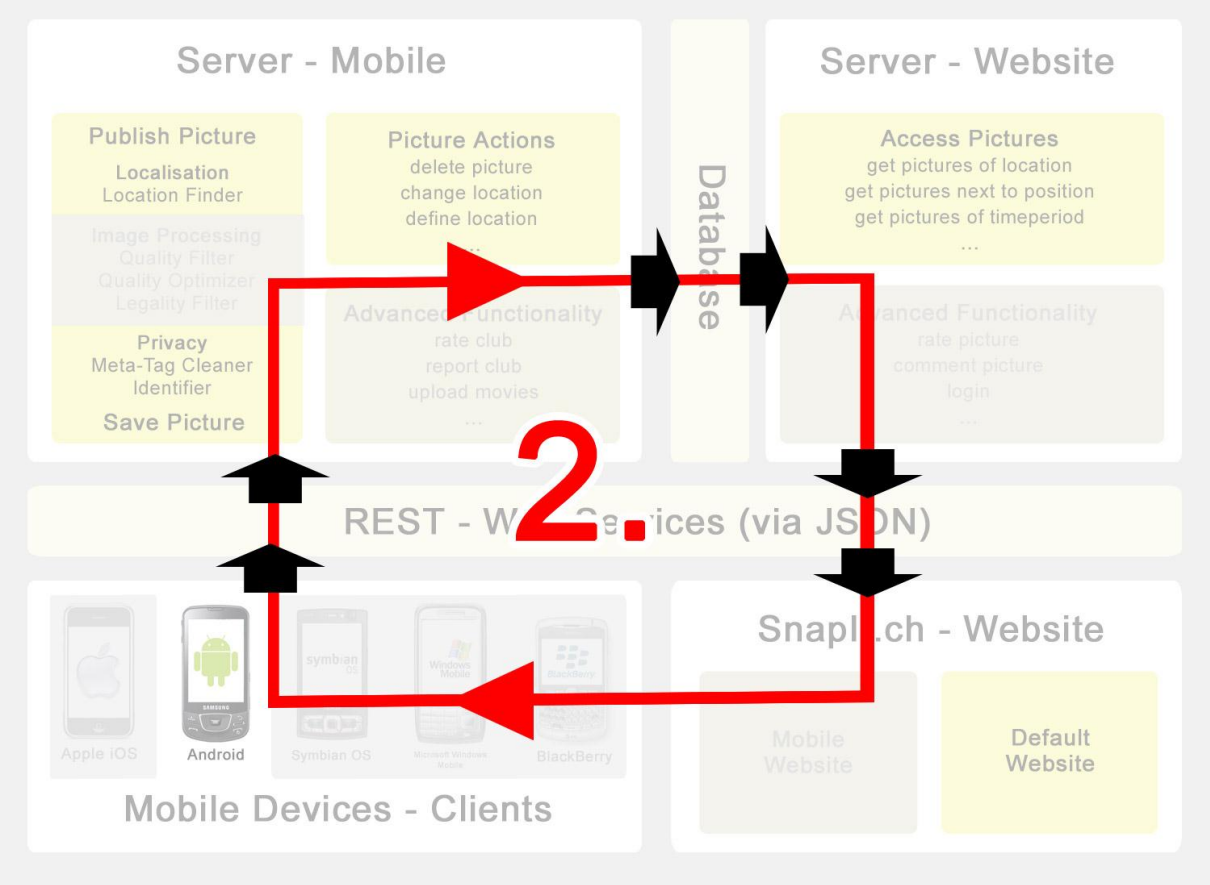

Abbildung 5-3: Übersicht Sprint 2

Beschreibung

- **User Story:** Foto Schiessen - 3,2,1...online!
- **Iterationszeitraum:** 4.-6. Woche (04.10. - 25.10.)
- **Abnahme:** 09.12.2010

Die zweite Iteration widmet sich bereits der Implementation des Systems. Als erstes wird aus dem Domainmodel die Architektur ermittelt bzw. Architektur Ideen gesammelt und analysiert. Der Schwerpunkt liegt im Aufbauen von Knowhow für die verwendeten Technologien und Konzepte. Das Endresultat ist eine stabile und gut dokumentierte Architektur. Der Software-Prototyp implementiert die Architektur und den Ablauf "Foto schiessen, Foto heraufladen, Foto ansehen" auf einfachste Art und Weise.

Abnahme-Test

Die Qualität der serverseitigen Architektur wird anhand von automatisierten Systemtests überprüft. Clientseitig wird der Prototyp anhand eines Testplans getestet.

Resultierende Dokumente

- Knowledge Dokumente
- REST Architektur
- Abnahmetest
- Testplan REST-Architektur

Software: Implementation eines Architektur-Prototypen, "Foto knipsen, Foto heraufladen, Foto ansehen"

Sprint 3

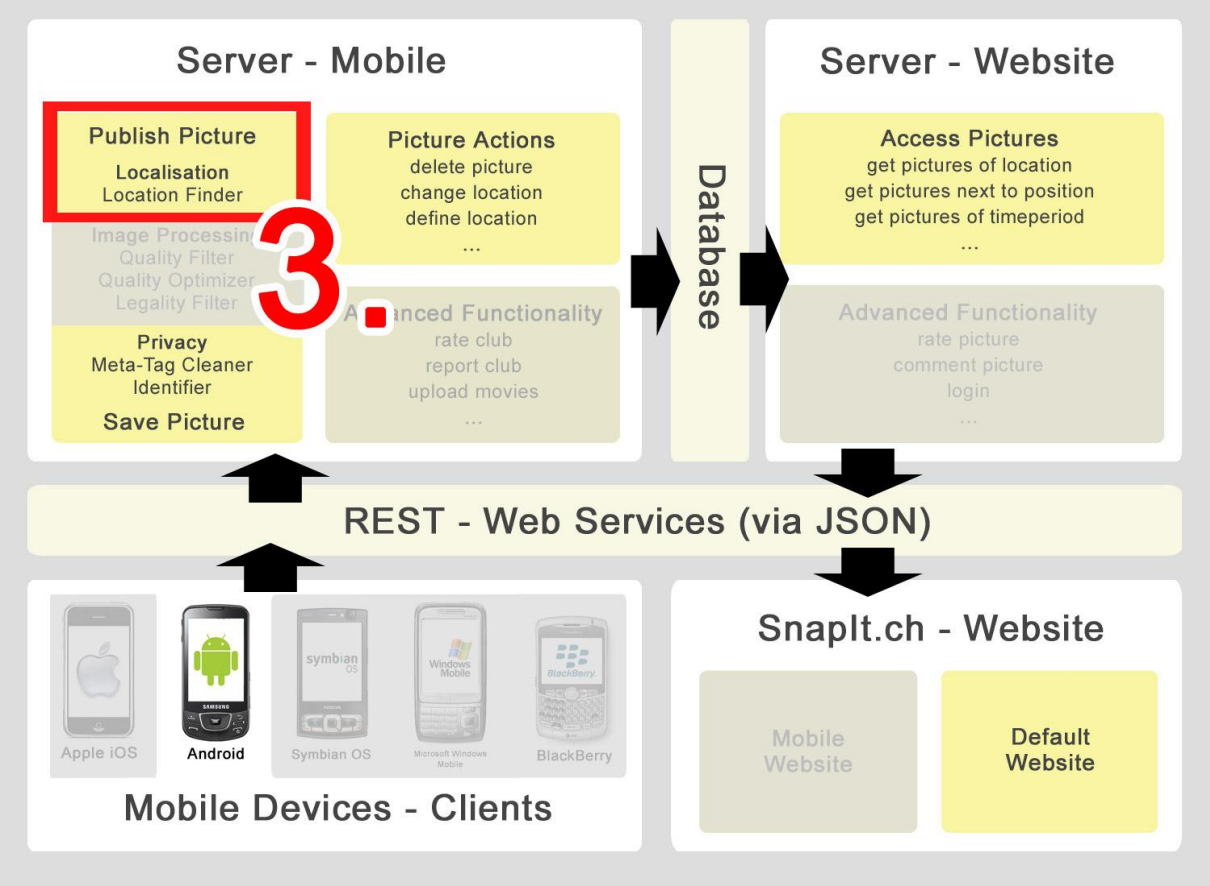

Abbildung 5-4: Übersicht Sprint 3

Beschreibung

- **User Story:** Foto schiessen - Vergiss die Bar!
- **Artefact Story:** Foto ohne Lokation
- **Iterationszeitraum:** 7. Woche (25.10. - 01.11.)
- **Abnahme:** 09.12.2010

SnapIt besteht aus zwei Kernfeatures, dem Heraufladen des Fotos und der Lokalisierung. Die zweite Iteration kümmert sich um die Lokalisierung. Hierbei geht es darum auf dem Handy die nötigen Daten zu sammeln, diese an den Server zu versenden und diese auf dem Server zu verarbeiten, so dass jedes Foto im positiven Fall einer Lokation zugeteilt wird. Auch der negative Fall wird behandelt und entsprechend darauf reagiert. Das Ziel ist es möglich viele Fotos einer Lokalität zuzuordnen können. Zum Endprodukt gehört auch die virtuelle HSR-Partymeile, welche als Testinfrastruktur aufgebaut wird.

Abnahme-Test

Der Test wird anhand der virtuellen Testumgebung HSR durchgeführt. Der empirische Test enthält sämtliche Testfälle zu allen Situationen. Dabei wird vor allem der Aspekt der Lokalisierung getestet. Das Ziel ist nachzuweisen, dass alle Bilder der korrekten Lokation zugeordnet werden können.

Resultierende Dokumente

- Domain Model (Initial Dokument)
- Analyse Lokalisierung
- Dokumentation Virtuelle Partymeile HSR
- Abnahmetest
- Testplan Lokalisierung

Software: Lokalisierung der geknipsten Fotos, sowie Lösung für Lokalisierungsprobleme

Sprint 4

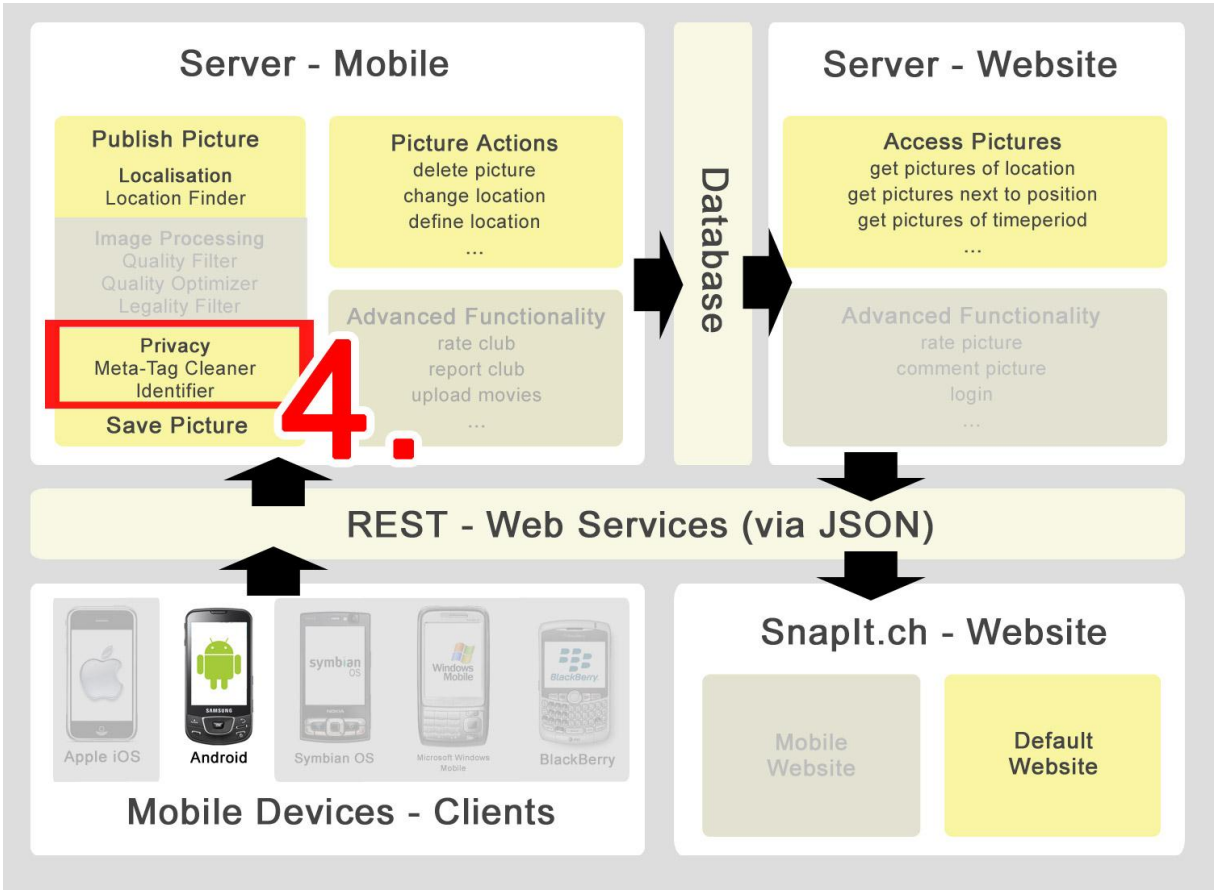

Abbildung 5-5: Übersicht Sprint 4

Beschreibung

- **User Story:** Foto schiessen - Mister Anonymous!
- **Artefact Story:** Foto mit Lokation
- **Iterationszeitraum:** 8. Woche (01.11. - 08.11.)
- **Abnahme:** 16.12.2010

SnapIt erlaubt es nicht die gespeicherten Daten einer Person zuzuteilen. Dieser Teil der Applikation wird während der vierten Iteration behandelt. Sämtliche Daten die Rückschlüsse erlauben, wie Angaben über den Benutzer, das Geräte oder das Foto werden gefiltert und gelöscht.

Abnahme-Test

Heuristischer Test anhand von Testpersonen, die über eine bestimmte Zeitspanne in der Testlandschaft Fotos knipsen. Die gesammelten Daten werden überprüft und die Testpersonen befragt.

Resultierende Dokumente

- Anforderungsspezifikationen
- Analyse Anonymisierung
- Abnahmetest

Software: Daten und Fotos sind gegen Aussen, aber auch im System nicht mehr eindeutig einer Person oder einem Gerät zuzuordnen.

Sprint 5

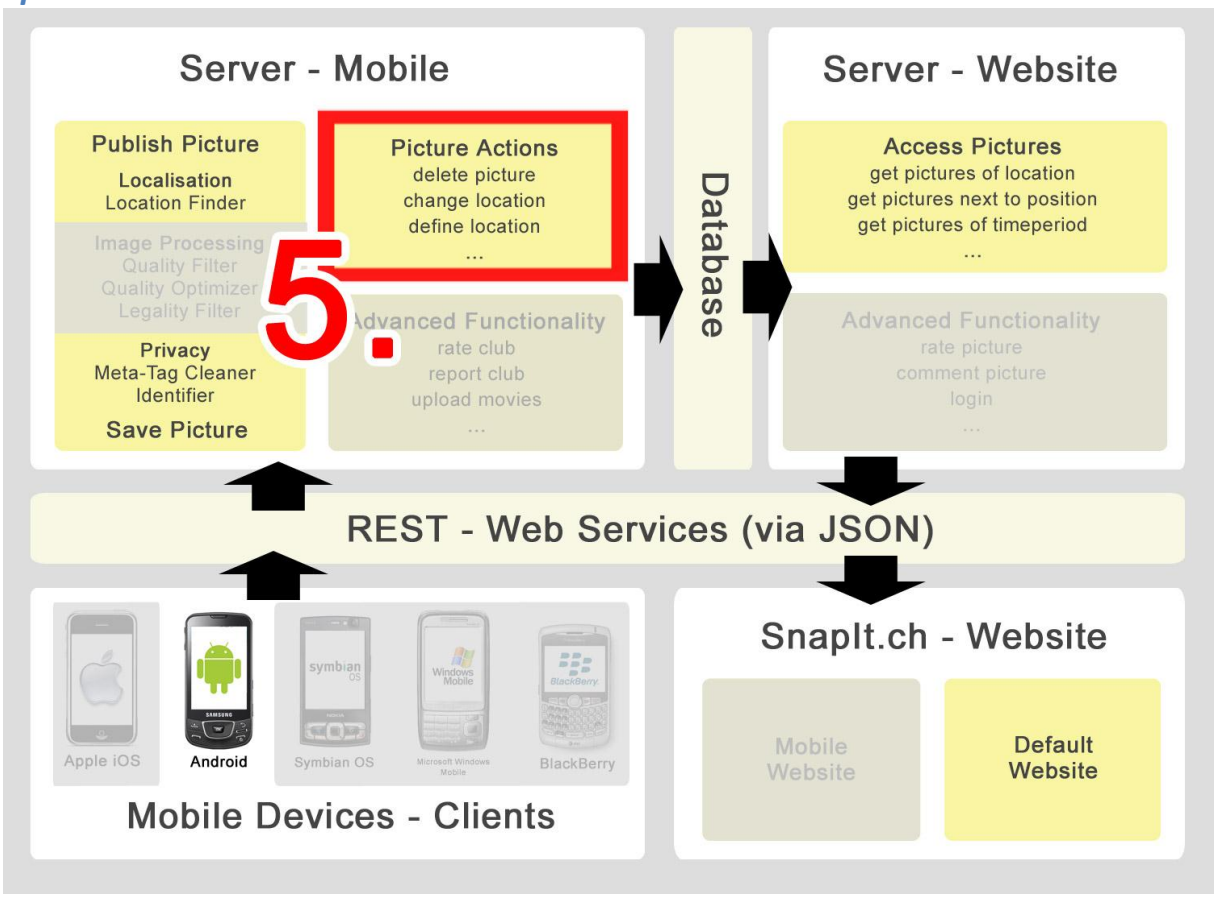

Abbildung 5-6: Übersicht Sprint 5

Beschreibung

- **User Story:** Foto schiessen, Foto löschen - Über die Stränge geschlagen!
- **Iterationszeitraum:** 9.-11. Woche (08.11. - 29.11.)
- **Abnahme:** 16.12.2010

Während der fünften Iteration wird vor allem die Android Applikation erweitert. Die geknipsten Fotos sollten nun abrufbar sein und es sollte möglich sein bestimmte Aktionen auf die Fotos anzuwenden (z.B. Foto löschen). Da hierfür ein grosser Teil des GUIs der Smartphone Applikation implementiert wird, werden hier auch die Paperprototypes erstellt und Usability-Tests durchgeführt.

Abnahme-Test

- Usability-Test
- Heuristischer Test

Resultierende Dokumente

- Paperprototypes Android GUI
- Abnahmetest
- Dokumentation Usability-Tests
- Testplan - Client komplett

Software: Implementieren der Features "Foto löschen", "Location ändern" und "Location bestimmen"

Sprint 6

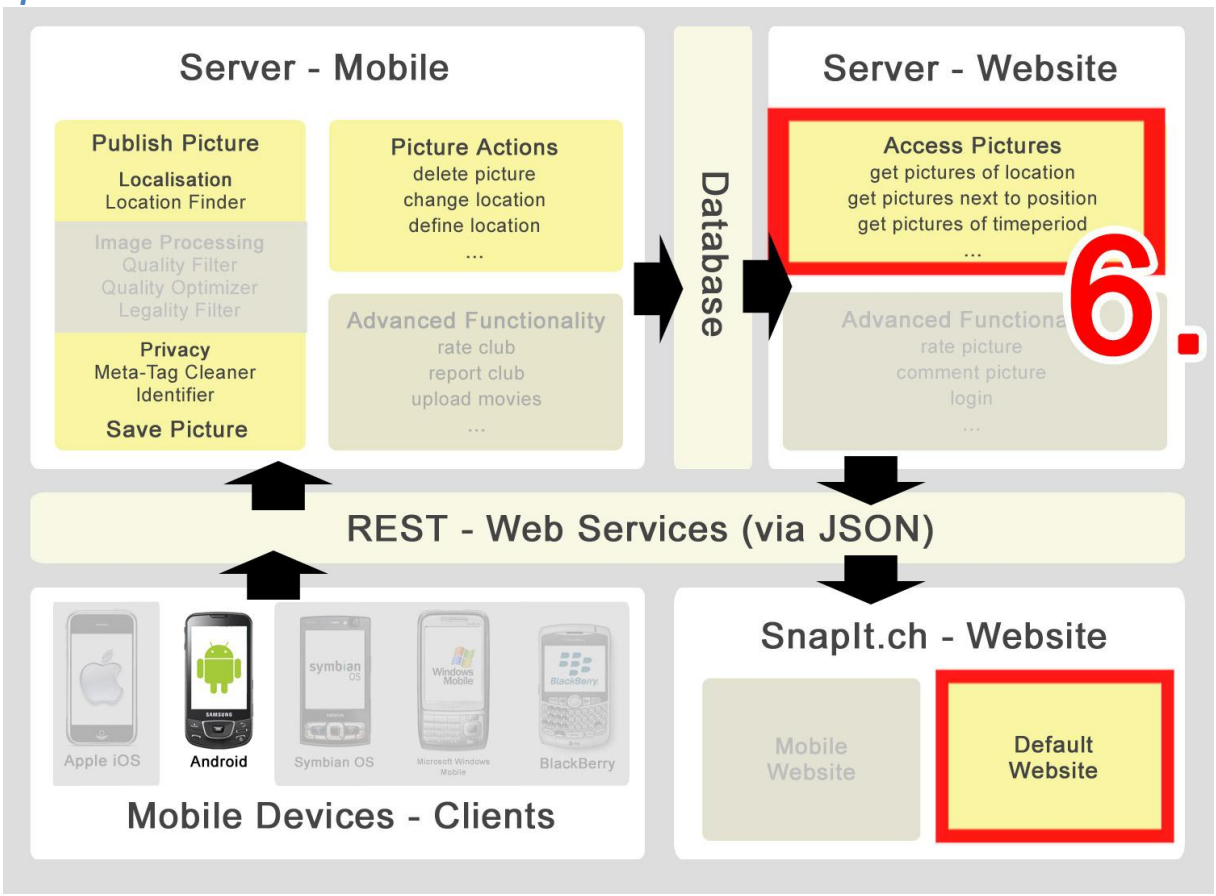

Abbildung 5-7: Übersicht Sprint 6

Beschreibung

- **User Story:** Fotos auf der Webseite betrachten - Dabei sein ist alles!
- **Iterationszeitraum:** 12. Woche (29.11. - 06.12)
- **Abnahme:** 16.12.2010

Die 6. Iteration kümmert sich um das Abrufen der Fotos. Der Fokus liegt auf dem Erstellen von Webservices die Fotos aufgrund bestimmter Parameter liefern. Eine simple Webseite stellt die Resultate dar und dient als Prototyp für eine zukünftige interaktive Webseite.

Abnahme-Test

Die Webservices werden anhand von JUnit-Tests auf die Korrektheit überprüft.

Resultierende Dokumente

-

Software: Webservices ermöglichen es Fotos auf Basis von Parametern abzurufen. Fotos werden auf einer Webseite simple dargestellt.

Sprint 7

Beschreibung

- **User Story:** Keine User-Story
- **Iterationszeitraum:** 13.-14. Woche (06.12. - 20.12)

Die letzte Iteration dient nur noch dem Aufbereiten der Dokumente und der Vorbereitung der Präsentation. Hier sind keine Änderungen an der Applikation geplant.

Abnahme-Test

Als Abnahmetest fungiert in erster Linie die Projektbewertung, welche die Benotung für die Software und die Dokumentation liefert.

Resultierende Dokumente

- Plakat
- Abstract
- Youtube-Video
- Benutzerhandbuch
- Komplette Softwaredokumentation

5.5 Kostenvorschlag

Das Projekt startet am 20. September 2010 und dauert bis am 23.12.2010. Es wird pro Woche im Durchschnitt ca. 17 Stunden pro Projektmitarbeiter daran gearbeitet. Die Stundenanzahl kann stark variieren und stellt nur ein ungefährer Bezugspunkt dar.

Gegen Ende des Projekts rechnen wir mit Überzeit, da wir dann alle Texte aus dem Wikipedia in ein Dokument überführen müssen. Zudem können erst dann Artefakte, wie zum Beispiel Plakate und Videos erstellt werden.

Das Team arbeitet während 14 Wochen ca. zwei Arbeitstage. In der Wirtschaft kostet ein Arbeitstag den Arbeitsgeber ca. 1000 CHF pro Mitarbeiter. Die Projektkosten würden sich also auf ungefähr 56'000 CHF belaufen (14*2*2*1'000).

5.6 Risikomanagement

Durch die iterative- und agile Vorgehensweise haben wir das Risikomanagement auf die Sprints aufgeteilt. Die folgenden Tabellen legen die Risiken für die verschiedenen Sprints dar.

Allgemein Risiken sind am Anfang unter dem Abschnitt Projektrisiken zu finden.

Risiko-Nr.	Risikotitel	Risikobeschreibung	Max. Schaden [h]	Eintrittswahrscheinlichkeit [%]	Massnahmen zur Vermeidung	Vorgehen bei Eintreffen
Projektrisiken:						
R1	Ausfall Arbeitsstation	Entwicklerhardware mit lokaler Testumgebung fällt aus	6	2	Wiki (Backup), SVN (Dokumente sichern)	Vorübergehend an den HSR-Rechnern weiterarbeiten
R2	Ausfall SVN-Server	SVN-Server der HSR fällt aus	10	3	Kopie auf Arbeitsrechner aktuell halten	Teammitglieder benachrichtigen und auf das Backup warten
R3	Streit im Team	Grosse Meinungsverschiedenheiten können zu Projektunterbrüchen führen	30	7	Konstruktive Kritik und aussenstehende Meinungen einholen	Aussprache mit Hilfe einer (technisch) kompetenten Person
R4	Ausfall eines Projektmitglieds	Ausfall aufgrund Krankheit, Unfall	99	1	Verantwortlichkeiten klar definieren	Andere Person stellt die Arbeit im ursprünglichen Sinn fertig evt. mit kleinerem Umfang
R5	Zeitknappheit	Belastung durch andere Module nimmt stark zu	32	5	Arbeitspakete von der Grösse her realistisch halten	Tasks werden nochmals priorisiert und nur an den wichtigsten gearbeitet
R6	Technologieprobleme	Einarbeitung in die unbekannten Technologien ist anspruchsvoller als angenommen	35	3	Analysen werden zum Teil vorgezogen	Hilfe holen von kompetenten Kollegen
Risiken im 2ten Sprint:						
R7	PostgreSQL Datenbank	Datenbank erfüllt Anforderungen nicht	1	1	Ausführliche Analyse	Auswechseln der Datenbank auf z.B. Oracle – muss aber dann noch genauer analysiert werden
R8	REST	Integration	3	1	Ausführliche	Service austauschen und

	Jersey	gestaltet sich schwierig und gewünschte Funktionalität ist nicht vorhanden			Analyse und Machbarkeit mit einem Prototyp belegen	z.B. auf SOAP wechseln – vorgängige Analyse
R9	JSON	Das JSON-Format erfüllt unsere Anforderungen nicht, problematisch sind binäre Dateien	1	2	Ausführliche Analyse und Machbarkeit mit einem Prototyp belegen	Format austauschen und z.B. auf XML wechseln
R10	Hibernate	Zu komplex und schwierig zu handhaben	10	4	Ausführliche Analyse und Machbarkeit mit einem Prototyp belegen	Direkt mit JDCB-Treiber auf der Datenbank arbeiten, anderen OR-Mapper nehmen
R11	JUnit	Unsere Tests sind nicht umzusetzen, gewisse Funktionalitäten werden nicht unterstützt	30	5	Ausführliche Analyse und Machbarkeit mit einem Prototyp belegen	Anderes Test-Framework evaluieren – z. B. TestNG
R12	Tomcat - Applikationsserver	Konfiguration gestaltet sich schwierig	10	6	Ausführliche Analyse und Machbarkeit mit einem Prototyp belegen	Umstellung auf einen anderen Applikationsserver – z.B. JBoss
R13	ANT	Einarbeitung dauert lange und gewisse Funktionalitäten werden nicht unterstützt	20	1	Ausführliche Analyse und Machbarkeit mit einem Prototyp belegen	Projekte zu Maven umkonfigurieren, Lösung mit dem Betreuer finden
R14	Performance der Datenbank	Probleme mit der Geschwindigkeit von Abfragen aufgrund von in den Tabellen gespeicherten BLOB-Daten (Bilder)	30	1	Periodische automatische Lasttests erstellen und durchführen	Bilder auf dem Dateisystem speichern und in der Datenbank nur einen Link zur Datei abspeichern
Risiken im 3te Sprint:						
R15	Lokalisierung	Android kann die gewünschte Geo-Position nicht so exakt liefern, wie wir sie gerne	5	9	Ausführliche Analyse und praktische Tests	Alternative, aber für den Benutzer annehmbare Lösung suchen und implementieren

		hätten				
Risiken im 4ten Sprint:						
R16	Anonymisierung	Die Anforderungen von anonymen Bildern und Bilder personenbezogen zu verwalten stören einander	6	7	Ausführliche Analyse und praktische Tests	Alternative Lösung suchen
Risiken im 5ten Sprint:						
R17	GUI: Papierprototyp	Das gezeichnet GUI des Papierprototypen kann nicht genau so entwickelt werden	10	9	Machbarkeit bei anderen Applikationen überprüfen	Gewisse Kompromisse eingehen für das fertige GUI
Risiken im 6ten Sprint:						
R18	Webseite	Einarbeitung in jQuery ist aufwendig	5	4	Beispielwebseite so einfach wie möglich halten	Für die Einarbeitung in jQuery mehr Zeit einplanen, da die Webseite als einziges ansatzweise die Komplexität des Systems visualisieren kann
Risiken im 7ten Sprint:						
R18	Abgabe-Dokument	Übernahme der geschriebenen Artikel vom Wikipedia ins Abgabe-Dokument ist zeitaufwändiger als geplant	50	9	So früh wie möglich anfangen mit dem Dokument	Überstunden!
R19	Video	Testgerät steht für die Aufnahmen nicht zur Verfügung	10	3	Sich frühzeitig mit den anderen Teams absprechen	Planung umstellen und an anderen Artefakten arbeiten

5.7 Infrastruktur

In diesem Kapitel wird beschrieben, unter welcher Umgebung gearbeitet wird.

5.7.1 Entwicklungsumgebung

Die lokale Entwicklungsumgebung setzt sich aus folgenden Komponenten zusammen:

Was	Wozu
Eclipse	Entwicklungsumgebung für SnapItClient, SnapItServer und Beispielwebseite
Android Emulator SDK	SnapItClient
Tomcat Webserver	SnapItServer
PostgreSQL	Datenbank

Von jedem Teammitglied wurde dies individuell auf dem eigenen Arbeitsrechner eingerichtet. Eine genaue Installationsanleitung ist im Kapitel Benutzerhandbuch zu finden.

5.7.2 Projektumgebung

Während der Projektdurchführung wurden alle Texte und Dokumentationen im eigenen Wikipedia erfasst. Das Wikipedia wurde von Christoph Süess kostenlos zur Verfügung gestellt.

Grafiken und andere Dokumente wie zum Beispiel Enterprise Architekt Dateien wurden im SVN-Verzeichnis abgelegt. Der Pfad zum SnapIt-SVN-Repository lautet: svn://svns.hsr.ch/SnapIt.

Um Ordnung zu halten wurde folgende Verzeichnisstruktur angelegt:

- /
  - branches – Fertige Applikationen
  - docs – Alle Dokumente, Bilder, usw.
  - tags - Meilensteine
  - trunk – Arbeitsverzeichnis der Projekte

Am Schluss dieser Arbeit wurden alle erstellten Artefakte in dieses Dokument überführt. Es soll nochmals erwähnt werden, dass die Wikipedia-Einträge nur für die Entwicklung gebraucht wurden und somit mit der Fertigstellung dieses Dokumentes obsolet sind.

5.7.3 Testumgebung

Als Testumgebung diente uns ein virtueller Windows Server in der Version 2003, auf dem ein Apache Tomcat 6 mit einer PostgreSQL-Datenbank installiert ist. Des Weiteren wurde uns ein Android-Mobiltelefon vom IFS zur Verfügung gestellt.

5.8 Qualitätsmassnahmen

Knowledge

Das Projekt befasst sich mit verschiedenen Technologien, die teilweise noch erlernt werden müssen. Für jedes grössere Themengebiet wird eine Person als Verantwortlicher definiert, welche sich in die Thematik einliesst und die

Implementierung zum grössten Teil vornimmt. Damit auch die zweite Person den Durchblick behält, ist auch sie darauf angewiesen die Materie zu erlernen, um bei der Implementation behilflich zu sein. Somit wird eine Arbeitsteilung erreicht und die Qualität der Implementation verbessert.

Gerade im Umfeld der Android Applikation wird mit einer Technologie gearbeitet, die sich noch in rasanter Entwicklung befindet. Empfohlene Open-Source Apps, der Android Entwickler Blog sowie Unterlagen zu den aktuellen Google-IOs sind behilflich um der Entwicklung zu folgen und dadurch qualitativ hochwertige Software zu schreiben.

UI Guides und Usability-Tests

Google bemüht sich mit UI Guides und Vorgaben (http://developer.android.com/guide/practices/ui_guidelines/index.html) alle von Drittanbietern entwickelten Applikationen einheitliche erscheinen zu lassen. Dadurch wird erreicht, dass Benutzer eines Android-Gerätes einfach mit neuen Applikationen zu Recht kommen und sich ein übersichtliches Gesamtbild ergibt. Durch das Einhalten dieser Regeln und regelmässigen Usability-Tests, die auch mit Einbezug von Drittpersonen erfolgen, wird die Qualität der Benutzeroberfläche und die Benutzerfreundlichkeit auf einem hohen Niveau garantiert.

Zu beachten gilt, dass wir beim Prototypen auf Benutzerführung, Positionierung der Bildschirmelemente und andere Elemente der Usability achten. Das Design (Farben, Bilder, Icons, usw.) hingegen wird nur provisorisch und mit tiefer Priorität erarbeitet (in einem späteren Projekt wird das Design festgelegt und für Corporate Identity über alle Systemelemente bzw. Applikationen gesorgt).

Codierrichtlinien

Einheitlich formatierter Sourcecode sorgt für rasches Verständnis und übersichtlichen Aufbau. Bei komplexen Codeblöcken wird vereinzelt auch mit normalen Kommentaren die Entscheidungen und der Ablauf beschrieben. Verwendete Codierrichtlinien:

- Die Mobile-Applikation wird nach dem Android Code Style Guide programmiert (Android Code Style Guide).
- Der Serverteil stützt sich auf die Sun-Richtlinien (Sun Code Conventions).

Review

Da einige Dokumentationen und der Sourcecode teilweise nicht in Teamarbeit entstehen, werden die Resultate regelmässig gegenseitig gelesen und kontrolliert. Der Sourcecode wird zusätzlich mit der Applikation PMD unter Anwendung eines speziellen Android-Checkstyles geprüft. PMD zeigt Schwachstellen und optimierungsbedürftige Stellen im Sourcecode an.

Google stellt zusätzlich diverse Tools zur Erhöhung der Qualität zur Verfügung die zusammen mit dem Android SDK installiert werden. Sie erlauben das Testen und Verbessern der Benutzeroberfläche und der Anwendungsperformance.

Tests

Da unsere Applikation aus verschiedenen Komponenten besteht und mehrere Technologien verwendet, ist die Testbarkeit grundsätzlich eher schlecht. Dabei gibt es diverse Probleme:

- Das Android Framework unterstützt zwar JUnit-Test, die sind aber sehr mühsam zu implementieren. Die Test können nicht ohne einen Emulator oder ein physisches Smartphone ausgeführt werden und sind darum sehr inperformant.
- Die wahrscheinlichen Hauptprobleme der App sind das Handling von Empfangsstörungen und -unterbrüchen, die Lokalisierung und Fehlfunktionen aufgrund der Android spezifischen Verwaltung von Prozessen. Diese Dinge zu simulieren ist sehr schwer und aufwendig, die Realität ist in diesen Fällen kaum nachstellbar.

Diese Gründe erklären warum beim Testen öfters ein heuristisches Verfahren zum Einsatz kommt. Dort wo die Implementation von JUnit-Test den Aufwand rechtfertigen wird darauf zurückgegriffen. Als Ergänzung werden empirische Testpläne ausgearbeitet, welche die Use Cases abdecken. Die Benutzerführung wird anhand heuristischer Usability-Tests geprüft.

JUnit

Für das Testen der SnapIt-Server Applikation wird JUnit 4 verwendet. Da es einfach ist Testfälle zu schreiben. Eine Alternative wäre TestNG, da aber die Einarbeitung viel Zeit in Anspruch nehmen würde, verzichten wir in diesem Projekt darauf.

Indirekt werden auf der Serverseite alle Schichten getestet. Für die Zwischenschichten werden keine eignen Tests geschrieben, ausser bei sehr speziellen Fällen, die dies rechtfertigen würden. Da dies einem automatischen Systemtest gleicht ist beim Ausführen der Tests darauf zu achten, eine saubere Testumgebung herzustellen und diese wieder im Ursprungszustand zu verlassen. Das Schreiben von Tests für jede Schicht ist zu aufwendig und somit wäre der Zeitaufwand zu hoch.

Dokumentation

Im Code werden nur schwer verständliche Algorithmen und ähnliches kommentiert. Ausser einer JavaDoc-Beschreibung der Klassen wird weitgehend auf Kommentare im Code verzichtet, da diese grundsätzlich unnötig sind. Jede Klasse wird mit einem JavaDoc-Kommentar versehen und die wichtigen Schnittstellenmethoden ebenfalls beschrieben. Die Analyse- und Architektur-Dokumente erklären ansonsten alles was nicht direkt aus der Implementierung entnommen werden können.

5.9 Einrichten der Entwicklungsumgebung

Das Einrichten der Entwicklungsumgebung benötigt aus sieben Schritten.

1. **Eclipse** Helios für Java EE Entwickler herunterladen und in ein Verzeichnis entpacken.
2. Android **Emulator** herunterladen und den SDK Manager ausführen und die gewünschte Versionen und Plattformen wählen.
3. **Android Development Tools** (ADT) in Eclipse als Plug-In installieren.
4. **JBoss Tools** in Eclipse als Plug-In installieren. Dieser Schritt ist optional und kann weggelassen werden. Diese Tools bieten jedoch einen guten Editor an, um Hibernate Konfigurationsdateien einfach und übersichtlich zu editieren.
5. **PMD** in Eclipse als Plug-In installieren. Wie der vorhergehende Schritt ist dies auch optional. Dieses Tool kann für automatischen Code-Reviews eingesetzt werden.
6. **Subclipse** – Für den Zugriff auf das SVN-Repository von SnapIt.
7. **Eclipse Metrics-Plugin** installieren.

Damit die Einrichtung, wie oben beschrieben ist, erfolgreich durchgeführt werden kann, ist das vorhanden sein eines JDK ab der Version 6 und einem Apache Tomcat 6 erforderlich.

6 Business Analyse

6.1 Markt-Analyse

Umfrage

Zur Markt-Analyse haben wir eine Umfrage durchgeführt. Die Empfänger der Mitteilung sind verschiedene Kollegen, darunter sind Studenten und Lehrabsolventen. Die meisten Befragten sind zwischen 21 – 24. Folgende Nachricht haben wir aunseren Kollegen über Facebook zukommen lassen:

Stell dir vor du kannst auf dein I-Phone oder Android-Phone eine gratis Applikation installieren, mit der du Fotos im Ausgang machen kannst und das geschossene Bild direkt im Internet landet inkl. Zeit und Ortsangabe, voll automatisch (z. B. 10.11.2011, 24:14, Downtown SG). Dieses Foto kannst du dann direkt an Kollegen schicken oder auf deinem Facebook-Profil veröffentlichen.

Stell dir vor das machen hunderte oder tausende Leute und du kannst jeden Abend auf einer Webseite live Bilder von den besten Partys und Clubs der Stadt anschauen. Du kannst dir die Fotos im Umkreis deines aktuellen Standorts anzeigen lassen oder auch einfach wild durchblättern.

Fragen

1. Was ist deine Meinung zu dieser Idee?
2. Würdest du die Webseite besuchen um diese Fotos zu betrachten?
3. Würdest du die App installieren und selber Fotos knipsen?

Antworten

Reto K.

1. So etwas gibt es schon. Es gibt Apps um Bilder live in das Internet zu laden.
2. Ja, aus purer Neugierde. +
3. Ich habe nicht einmal ein Profilbild im Facebook. Ich würde keine Bilder machen. -

Ruedi O.

1. Ich habe kein solches Smartphone, aber ich könnte mir vorstellen, dass dies eine grosse Nachfrage wecken könnte.
2. –
3. Ich werde kein Bild knipsen, aber es gibt genügend Leute die „geil“ darauf sind das jeder hinterletzte weiss, was sie im Moment machen. -

Patrick F.

1. -
2. Ich würde die Webseite ansehen und die App installieren +
3. Ja oder nein, je nachdem.

Sandro V.

1. Super Idee, das fehlt noch!
2. Ich würde die Seite von Zuhause aus besuchen. +

3. Ja, ich würde Bilder schiessen. +

Severin F.

1. Das finde ich eine coole Idee. Vor allem wenn man eine coole, bereits am laufende Party/Club sucht. Vielleicht kann damit bestehenden Partyfoto-Plattformen zusammengearbeitet werden, damit die bestehenden Strukturen/Usern schnell erreicht werden können. (Lautundspitz/Tillate) Das wäre für diese Unternehmen sicher eine attraktive Neuheit. Unter Umständen braucht die Applikation ohne Verbindung mit bestehenden Strukturen etwas viel Zeit, um in die Gänge zu kommen, bzw. müsste sie bei vielen Usern installiert sein. Im App Store von Apple werden meines Wissens Apps mit zu wenig Downloadzahlen vom Store entfernt. Diese Gefahr würde bestehen ohne einen bekannten Namen dahinter.
2. Ich würde diese Website besuchen um die Fotos anzusehen und die aktuellen Partys zu suchen. +
3. Ja ich würde die App installieren und selber Fotos knipsen. (Auch ohne Lautundspitz oder Tillate als "Brand") +

Bruno S.

1. Interessant, aber wozu noch mehr Bilder von Besoffenen-Leuten
2. Nein -
3. Nein, ich habe weder ein I-Phone noch ein Android-Phone -

Andreas Z.

1. Gute Idee, aber alleine für diesen Zweck eine eigene App find ich etwas "mager"
2. Sicher, immer dann wenn Langeweile herrscht +
3. Warum nicht +

Patrick G.

1. Ich finde die Idee super!
2. Ja, klar +
3. Ja, ich mache ja auch sonst Fotos wenn ich im Ausgang bin. Privatsphäreneinstellungen wären aber angebracht. Damit ich Fotos von mir und meinen Kollegen evt. Nur für uns knipsen kann +

Christian M.

1. Super Idee!
2. Ja +
3. Ja +

Lukas S.

1. Ich denke dieses Projekt hat Potenzial. Zumal heute eine solche App nicht vorhanden ist und eine solche App durch die Einfachheit durchaus brillieren könnte. Die Schwierigkeit liegt wahrscheinlich darin, die App zum Gebrauchsobjekt zu machen. Also solange nur wenige Personen die App nutzen werden wahrscheinlich einige die App nie wieder verwendet, was aber nicht an der Nützlichkeit der App, sondern am Mangel der User liegt.
2. Ja +

3. Würde ich, habe aber kein iPhone/Android. Ich sehe aber ein anderes Problem: hast du mal versucht in einer Disco oder einer Bar mit dem iPhone ein Foto zu machen? Es könnte nämlich sein, dass es verschwommen wird...

Person A

Ich habe gerade einen Kurs (IMT) der sich mit Web 2.0 beschäftigt. Ich glaube dieses Programm könnte tatsächlich Zukunft haben, da es die Partys in der Umgebung transparenter machen würde und somit leichter die geilste Party gefunden werden können.

1. Gute Idee, sogar umsetzungspotenzial vorhanden, aber hohe Benutzerzahl benötigt und somit nicht leicht umzusetzen --> Strategie gefragt. erst nur auf SG beschränken etc.
2. Homepage angucken, vor allem mit Facebook-Verlinkung wär möglich +
3. Ich habe kein iPhone -

Person B

1. Eine sehr innovative Idee, die wirklich funktionieren könnte.
2. Ja, eine solche Applikation würde ich durchaus nutzen. Jedoch sehe ich ein grosses Problem. Und das ist die Qualität der Bilder. Einerseits da nur von Mobiltelefonen aus fotografiert wird (was besonders bei den Lichtverhältnissen in den Clubs recht katastrophal rauskommt) und andererseits da jeder einfach alles Hochladen kann was er gerade knipst. Hinzukommt die rechtliche Frage, da die Bilder ja dann veröffentlicht werden und man dafür die Erlaubnis der fotografierten Person benötigt. +

Person C

- Ich finde das eine mega coole Idee. Würde die Seite anschauen und die App herunterladen. +

Person D

1. Brauch ich nicht. Manchmal ist es auch besser, wenn man die Fotos zuerst am nächsten Tag anschauen kann, bevor man sie veröffentlicht.
2. Nein. Schau ich mir schon heut nicht an. Partyfotos sind nicht so spannend. -
3. Nein. Erstens kein iPhone, Android-Handy. Zweitens: Brauch ich nicht. -

Person E

- Du hast sehr geringe Investitionskosten. Und bei der Entwicklung der Software lernst du auch etwas. Ob du damit Geld verdienst, bezweifle ich. Ich persönlich würde dein Angebot nicht nutzen, weil ich ein Portal habe (Facebook) und das reicht mir. Ich sehe deine Verkaufsargumente, wäre für mich aber kein Benefiz. Auch aufgrund der Privatsphäre. -

Person F

1. Finde ich keine gute Idee, es gibt schon viel zu viele Fotos im Internet und ehrlich gesagt möchte ich nicht auf einem dieser Fotos sein. ich lasse mich auch nicht von Partyfotografen fotografieren, weil nicht jeder wissen muss, wo ich im Ausgang war oder mit dieser Anwendung, wo ich im Ausgang bin.

2. Nein ich würde die Website nicht besuchen -
3. Schlussfolgerung ;-) nein würde ich beides nicht -

Person G

1. Coole Idee, aber du brauchst schnell viele Leute, damit es etwas wird
2. Ich denke schon +
3. Keine Ahnung

Person H

- Es gibt schon sehr viele Portale für Fotos, etc.

Person I

1. Es gibt bereits was ähnliches (http://www.facebook.com/l/8b712;foursquare.com/). Wenn man http://www.facebook.com/l/8b712;TechCrunch.com glauben kann wird Facebook diese Funktion nun einbinden. In vielen Clubs besteht keine Internet Verbindung
2. Warum sollte ich Fotos von der Party machen, anstatt die Party zu genießen?
3. Bye, Bye Privatsphäre -

Auswertung

- Anzahl Rückmeldungen: 18

Name	+ positiv	enthalten	- negativ
Fotos bzw. Webseite ansehen	11	2	5
Fotos knipsen	6	5	7

Fotos bzw. Webseite ansehen

Ein grosser Teil der Befragten findet unsere Idee gut und würde gerne die Fotos über das Phone oder über den PC auf unserer Webseite ansehen. Zwei Personen haben sich nicht klar zu dieser Frage geäussert. Fünf der Befragten werden sich nach ihrer Aussage die Seite nicht anschauen. Wir könnten uns vorstellen, dass bei hoher Popularität unserer Seite, die Webseite auch von denjenigen hin und wieder besucht werden würde, welche eine negative Bewertung abgegeben haben.

Fotos knipsen

Sechs der Befragten würden Ihr Phone zum Knipsen von Fotos verwenden. Unter den vier Enthaltungen befinden sich Leute die kein passendes Mobilephone besitzen oder sich über die Sache noch nicht im Klaren sind. Wie erwartet sind viele Leute nicht bereit im Ausgang Fotos zu schiessen. Einigen ist es schlichtwegs zu dumm im Ausgang sich um Fotos zu kümmern, andere sorgen sich um ihre Privatsphäre. Der Reiz die Mobileapplikation zu nutzen, ist im Moment noch zu gering. Durch weitere Ideen und der Verbesserung der Privatsphäre könnten wir bestimmt mehr Leute erreichen.

Ideen und Meinungen aus der Umfrage

- Zusammenarbeit mit bestehenden Partyportalen

    - Grundsätzlich nein. Denkbare Alternative falls eigene Webseite nicht in die Gänge kommt.
- Privatsphäre
- Bildqualität
- Einfachheit
    - Sehr wichtig!
- Facebook-Verlinkung
- Hohe Nutzerzahl
- Rechtliche Frage
    - Muss natürlich noch genauer abgeklärt werden, ist aber grundsätzlich nichts anderes als bei Facebook.

6.2 Konkurrenz-Analyse

Konkurrenten

Lokalisierungsdienste

- facebook.com/places
    - Facebook Places verknüpft die aktuelle Lokation der Person mit Statusmeldungen und mit der Position der Facebook-Freunde. Dadurch entsteht die Möglichkeit die Standorte der Kollegen einzusehen und die Informationsflut auf Facebook auch geografisch zu ordnen und zu filtern.
- foursquare.com
    - Foursquare ist eine Applikation für Mobiltelefone über die man seinen Freunden die aktuelle Position mitteilen kann. Foursquare wird teilweise auch als Vorreiter von Places gesehen, da es die Möglichkeit den aktuellen Standort auf Facebook zu publizieren bereits vorher ermöglichte. Neben der Verknüpfung mit Social Networks agiert Foursquare auch als Reality-Spiel. Durch das "Einchecken" (publizieren des aktuellen Standorts) sammelt der Benutzer Punkte, anhand deren er gewisse Dienste im realen Leben beanspruchen kann (z.B. gratis Kaffee im Starbucks).

Partyportale

- usgang.ch
- lautundspitz.ch
- tillate.com
- usw.

Die aufgelisteten Partyportale bauen auf derselben Grundidee auf. Rekrutierte Fotografen knipsen Fotos auf Partys und Events. Diese Fotos werden dann im Verlauf des nächsten Tages vom Fotograf über das Internet auf dem Partyportal veröffentlicht. Die Fotos können dann von den Website-Besuchern eingesehen und kommentiert werden. Im Laufe der Zeit entwickelten sich die Partyportale weiter zu Social Communities, Eventkalendern und Magazinen die rund um die Thematiken Events, Nachtleben und Party publizieren.

Die Partyportale setzen beim Sammeln der Bilder vor allem auf Qualität und auf bekannte Events und Partys.

Beurteilung der Marktposition

Die beiden Konkurrenztypen unterscheiden sich in mehreren Punkten von unserem Produkt.

Die Kernfunktion der Lokalisierungs-Dienste ist das Veröffentlichen der aktuellen Position des Benutzers. Indirekt wird die geografische Position mit anderen persönlichen Daten verknüpft. Die Facebook-App erlaubt das Uploaden von Fotos. Das direkte verknüpfen mit einer geografischen Position ist jedoch nicht möglich. Foursquare unterstützt keine Bilder, hier wird nur Text und Koordinaten verwendet.

Die Partyportale gleichen auf den ersten Blick sehr unserem Produkt. Die Ziele sind jedoch absolute widersprüchlich:

- Die Partyportale setzen auf Bilder in hoher Qualität, wobei bei uns die Qualität Sekundär ist.
- Die Fotos der Partyportale sind normalerweise ein bis zwei Tage nach dem Event abrufbar, bei uns dauert der Prozess zwischen Aufnahme und Publikation im Normalfall nicht mehr als 60 Sekunden.
- Die Partyportale, die sich inzwischen zu Communities weiterentwickelt haben, versuchen die Fotos zunehmend mit persönlichen Daten der Fotomotive zu verbinden. Dies geschieht über Möglichkeiten, sich an einem Event anzumelden, Personen auf Fotos zu markieren, usw. SnapIt setzt auf Anonymität und Privatsphäre.
- Die Fotografen der Portale befinden sich meistens in populären Lokationen und an namhaften Partys. Die Fotos von SnapIt stammen aus diversen Lokationen, die mehr oder weniger berühmt sind.

Durch die unterschiedlichen Ziele, haben die Benutzer auch einen anderen Nutzen:

- Was kann SnapIt im Gegensatz zu den Partyportalen?
  - SnapIt liefert Live-Bilder. Diese Bilder können dem Benutzer einerseits als Entscheidungsgrundlage dienen und andererseits müssen die Benutzer nie auf ihre Aufnahmen warten, wie das bei den Partyportalen der Fall ist.
  - SnapIt ist Anonym. Privatsphäre wird bei unserem Produkt gross geschrieben. Unser Fokus liegt auf der Lokation und nicht auf den Besuchern, wobei natürlich das Publikum eine Lokation zu einem grossen Teil ausmacht.
  - Unsere Fotos können nicht nur nach Lokation geordnet werden, sondern besitzen eindeutige Koordinaten. Somit können auch Bilder in einem gewissen Umkreis einer geografischen Position gefiltert werden.
- Was können die Partyportale besser?
  - SnapIt wird nicht dieselbe Qualität der Bilder erreichen.

Fazit

Viele andere Produkte verwenden dieselben Möglichkeiten wie SnapIt. SnapIt ist aber in seiner Art, soweit bekannt, einzigartig. Das Verknüpfen der beiden Hauptelemente, geografische Position und Fotos, im Kontext von Events und Lokationen im Bereich des Nachtlebens ist ein neuer Ansatz. Vor allem mit dem Echtzeitverhalten und der Anonymität der Benutzer werden Faktoren beachtet, die in der heutigen Gesellschaft einen hohen Stellenwert haben.

SnapIt kann als eine Mischung zwischen Partyportal und Lokalisierungsdienst gesehen werden, der aber nicht die Personen, sondern die Lokationen im Mittelpunkt stellt.

Nicht die Partyportale und die Lokalisierungsdienst sind unsere wahren Konkurrenten, sondern das grosse Angebot an Anwendungen, welches die potentiellen Kunden von der Verwendung von SnapIt abhalten könnte.

6.3 User Stories

Stakeholder

Folgende Stakeholder sind bei der Verwendung von SnapIt betroffen:

- **Fotograf**: Ein Partybesucher, der Fotos von der Party schiesst.
- **Anseher**: Person, welche die Bilder auf der Webseite betrachtet.
- **Fotomotiv**: Person, die fotografiert wurde.
- **Lokationsinhaber / Personal**: Person, die für die Lokation arbeitet und zu Werbezwecken Fotos knippst.
- **Informierer**: Person, welche die Webseite nutzt, um passende Partylokationen zu finden.

User Stories

Foto Schiessen - 3,2,1...online

Stakeholder: Fotograf, Anseher

Franz Foto knippst um 23 Uhr im Restaurant Bierkeller ein Foto und betrachtet es sogleich über sein Handy auf snapit.ch.

Foto schiessen - Hier feiern wir!

Stakeholder: Fotograf

Frank Famous geht sehr gerne in den Ausgang und kennt dort auch öfters viele Freunde. Nach einer langen und anstrengenden Arbeitswoche ist endlich Freitagabend. Frank beschliesst, dass sich er und drei weitere Freunde heute Abend gegen 21:00 Uhr am Bahnhof in St. Gallen treffen, um dann gemeinsam in einen Club zu gehen. Wohin sie genau gehen wollen wissen sie noch nicht. Nach einem kurzen Gespräch wohin es gehen soll, wurde einstimmig beschlossen in den Club namens "Backstage" zu gehen. Als sie dort angekommen sind und auch schon einige Drinks getrunken hatten, möchte Frank den tollen Moment auf einem Foto festhalten, damit er es seinem besten Freund, der leider krank ist und zuhause bleiben musste, zeigen kann. Er nimmt sein Android-Mobiltelefon hervor und startet die Applikation "SnapIt". Kurz darauf wurde ein passendes Foto geschossen, das automatisch im Internet landete.

Foto schiessen - Vergiss die Bar!

Stakeholder: Fotograf, Informierer

Peter Party ist angetan von der Idee von SnapIt. Heute hat er sich mit einigen Kollegen verabredet um etwas durch die Stadt zu ziehen. Geduscht, gestylt und mit einer frischen SnapIt-Installation auf seinem HTC Desire trifft er sich mit seinen Kollegen in der Galleria-Bar. Leider läuft nicht viel und die Jungs entscheiden sich ihren Standort zu wechseln. Kurz bevor sie sich auf den Weg machen, schnappt sich Peter sein

Smartphone und knippst zwei Fotos von der verlassenen Bar. Mit der Genugtuung, dass sich heute keiner mehr in die lahme Location verirrt, verlässt er die Bar.

Foto schiessen - Werbung in eigener Sache!

Stakeholder: Lokationsinhaber / Personal

Barbara Barli ist stolze Besitzerin einer eigenen kleinen Bar. Ihr Geschäft läuft gut aber sie sucht schon lange eine effektive und kostengünstige Methode um Werbung für ihre Bar zu machen. Sie beobachtete schon seit einiger Zeit, dass vermehrt Gäste in ihrer Bar mit dem Handy Fotos schiessen. Eines Abends beschliesst sie einen dieser Gäste zu fragen, was er denn da genau macht. Dieser erklärt ihr sehr ausführlich, was es mit der Applikation SnapIt auf sich hat. Zufrieden spendiert sie dem Gast einen gratis Drink für die tolle Auskunft. Bis zum nächsten Wochenende installiert sie sich auch SnapIt auf ihrem Mobiltelefon. Überraschenderweise kommt gegen 23:00 Uhr die vorheriges Wochenende neue gewählte Miss Schweiz in ihre Bar um sich einige Drinks zu genehmigen. Ihre Freude ist riesig, aber sonst sind leider noch nicht viele Leute anwesend. Dann erinnert sie sich an die neu installierte Applikation SnapIt. Sie überlegt nicht lange, startet SnapIt und nutzt den Augenblick um ein Foto von der neuen Miss Schweiz zu schiessen. Kurz darauf stürmen die Gäste nur so in die Bar, zur Freude von Barbara.

Foto schiessen - Mister Anonymous

Stakeholder: Fotograf

Paul Privacy findet die neuen Social Networks zwar eine coole Sache und würde sich dort auch gerne beteiligen, doch dafür legt er zu viel Wert auf Anonymität im Internet. Darum landen die von ihm heute im Ausgang geknipsten Fotos nicht auf Facebook, Twitter und co., sondern auf snapit.ch, total anonym, selbstverständlich!

Foto schiessen, Foto löschen - über die Stränge geschlagen!

Stakeholder: Fotograf, Fotomotiv

Wow, was für ein Abend denkt sich Verona Vollgas. Gestern wurde es etwas zu spät und das eine oder andere Bier weniger hätte auch nicht geschadet. "Das wäre ja alles nicht so schlimm, wäre da nicht das lästige Kopfweh und die vielen Fotos auf snapit.ch, die ich Gestern knipste" denkt sich Verona. Von ihrem Bett aus greift sie zum Smartphone und löscht die Fotos von snapit.ch. Mit nur noch halb so starken Schmerzen schläft Verona weiter.

Fotos auf der Webseite betrachten - Dabei sein ist alles!

Stakeholder: Anseher

Igor Ill geht für sein Leben gerne mit seinen Kollegen in Club "Magic Mountain" um zu tanzen und zu feiern. Aber leider ist er gerade krank und kann sein Bett nicht verlassen. Er weiss aber, dass seine Freunde heute Abend ohne ihn weggehen. Um trotzdem einen Hauch des Feelings zu erleben, ruft er die Webseite von SnapIt auf. Er weiss ja wo sie sich aufhalten. Auf der Webseite sieht er die Bilder, die seine Kollegen vor wenigen Minuten geschossen hatten und nimmt gerade noch einen Schluck von seinem Tee, denn sein Drang wieder gesund zu werden und auch wieder in den Ausgang zu gehen, steigt mit jedem weiteren Foto, dass er betrachtet.

Fotos auf der Webseite betrachten - Neues entdecken!

Stakeholder: Informierer

Walter Whats-Up kennt die berüchtigten Clubs der Stadt und weiss auch dass das "Weekend" kaum jemanden rein lässt, das "Watergate" stets überfüllt ist und sich im "Matrix" nur die Minderjährigen rumtreiben. Da er sich an diesem lang ersehnten Wochenende nicht auf einer dieser Clubs einlassen will, entscheidet er sich am schon etwas späteren Abend eine neue Location zu suchen. Auf der Webseite von SnapIt betrachtet er die Fotos die aus der ganzen Stadt hereinkommen. Besonders viele Fotos stammen aus dem Klub "Goldengate". Auch als Insider war Walter dieser Klub noch nicht bekannt, der grosse Frauenanteil verleiten ihn aber dazu dieser Location an diesem Abend die Ehre zu erweisen.

Fotos mit dem Handy betrachten - Die Party danach!

Stakeholder: Fotomotiv, Anseher

Sabrina Schön lässt sich gerne auf Fotos ablichten, aber es dürfen nur die Besten sein, denn sie ist sehr auf ihr Äusseres bedacht. Ein Kollege von ihr hatte am Vorabend mit der Applikation SnapIt einige Fotos von ihr geschossen, da sie sich nicht mehr genau erinnern kann, ob diese gut geworden sind, nimmt sie ihr Android-Handy und startet die vorhin erwähnte Applikation. Nach kurzer Eingabe vom gestrigen Clubstandort wird sie fündig und kann die Bilder, welche übrigens alle sehr schön geworden sind, ansehen.

Fotos mit dem Handy betrachten - Was geht ab?

Stakeholder: Fotograf, Informierer

Tabea Travel hat es endlich geschafft, Sie steht zusammen mit ihrer besten Freundin am Piccadilly Circus. Die Reise nach London war schon immer ein grosser Traum der beiden Studentinnen. Voller Eifer und Freude entscheiden sich die beiden den Abend nicht an der von Touristen überschwemmten Kreuzung zu vergeuden. Auch von den vielen zugesteckten Flyern und Eintritten lassen Sie sich nicht beeinflussen. Tabea schnappt sich ihr Smartphone und sucht mit Hilfe von SnapIt nach einer Location. In einer unscheinbaren Seitenstrasse scheint London die beste Seite seiner berüchtigten Nightlife-Szene zu zeigen. Und schon eine halbe Stunde später haben die beiden den Tanzfloor des Clubs erobert. "Ohne SnapIt hätten wir das nie gefunden!"...denkt sich Tabea und bedankt sich mit einem weiteren Foto der Location bei SnapIt und all denen die mitmachen!

Geknippst werden - Lieber nicht!

Stakeholder: Fotomotiv

Louis Lüge hat sich gerade bei seiner Freundin Luisa krankgeschrieben. Er hat keine Lust mit ihr ins Kino zu gehen und verbringt den Abend lieber mit seinen Kollegen. Das muss Luisa aber nicht wissen. Die Jungs amüsieren sich prächtig, das findet auch John, ein Kollege von Louis. John zückt daher immer wieder sein Smartphone und knippst die Jungs auf der Tanzfläche ab. Louis fühlt sich dabei nicht so gut und hat Angst Luisa könnte die Fotos entdecken. Unbemerkt weicht er auf die Seite, sobald John zur Kamera greift.

Artefact-Story

Foto ohne Lokation

Der Benutzer schiesst mit der Applikation SnapIt ein Bild, welches er an einer unbestimmten Lokation aufnimmt. Das Foto wird erstellt und wird zum SnapIt-Server hochgeladen. Dort angekommen merkt das System, dass die mitgegebenen Lokalisierungspunkte mit keinem Klub in der Datenbank übereinstimmen. Jedoch wird es trotzdem gespeichert, aber mit dem Vermerk, dass kein passender Klub gefunden wurde. Das Foto erscheint trotzdem auf der Webseite, aber ohne dazugehörenden Klub. Falls der Benutzer die Lokation kennt, kann er dies mit Hilfe von SnapIt mitteilen und das Foto wird mit der neuen Lokation (Klub / Bar) verbunden. Nun kann das Foto, wie alle anderen Bilder, mit der passenden Lokation auf der Webseite angezeigt werden.

Foto mit Lokation

Das Foto, die Position und weitere Meta-Daten werden von der Kamera erfasst und über eine Schnittstelle an den Server geschickt. Der Server nimmt das Foto entgegen und entfernt sämtliche Meta-Daten. Das anonymisierte Foto wird lokalisiert und auf dem Server abgelegt. Ein Webservice liest das Foto aus und liefert es der Webseite, wo es der Benutzer einsehen kann.

6.4 Use Cases

UC01 Foto knipsen

Der Benutzer knipst ein Foto mit Hilfe der Applikation SnapIt und sendet dieses ab.

User-Stories	• **Einfaches Foto schiessen** • **Foto schiessen um Stimmung aufzunehmen**
Primary Actor	SnapIt - Benutzer
Scope	SnapIt - Applikation
Level	Benutzerziel
Overview	Der Benutzer macht ein neues Fotos mit Hilfe der Applikation SnapIt und sendet dieses an unseren Server.
Stakeholders & Interests	**Fotograf**: Einfaches knipsen des Fotos und unkompliziertes Hochladen ohne zusätzliche Benutzereingaben. **Anseher**: - **Fotomotiv**: - Lokationsinhaber / Personal: - **Informierer**: - **SnapIt**: Qualität und Quantität muss stimmen, zudem sollte immer die passende Lokation zugeordnet werden können (gegebenenfalls mit Benutzerhilfe). Der Server muss die vielen Requests verarbeiten können und stabil laufen.
Preconditions	Der Benutzer hat die Applikation gestartet.
Postconditions	Das hochgeladene Bild wird vom System akzeptiert und auf der Webseite zur Verfügung gestellt.
Motivation	Der Benutzer möchte zeigen wie er und seine Freunde feiern und dies der Welt mitteilen.
Main Success Scenario	1. Der Benutzer schiesst ein Foto. 2. Das Foto und die aktuellen Koordinaten werden an den Server geschickt.

	3. Der Benutzer wird über den Erfolg / Fehlerfall informiert. 4. Das Foto wird zusammen mit der Lokation auf der Webseite angezeigt.
Extensions (Alternative Flows)	2a. Keine Internetverbindung vorhanden 4a. Das Foto kann nicht mit einer Lokation verknüpft werden.
Special Requirements	Die Übertragung der Daten sollte in einem angemessenen Zeitraum stattfinden. (unter 1 Minute)
Technology & Data Variations List	Mobiltelefon unterstützt GPS oder alternative Lokalisierungstechnologie
Frequency of Occurrence	ca. 4'000 Bilder pro Abend
Open Issues	• Wie wird der Benutzer informiert, dass keine passende Lokation dem Foto zugeordnet werden konnte? • Wie merkt die Applikation, dass sie wieder eine Internetverbindung hat? • Werden die Lokations-Vorschläge zuerst geprüft oder direkt dem Bild zugeordnet?

UC02 CRUD Fotos

Der Benutzer verwaltet seine geschossenen Fotos.

User-Stories	• **Foto löschen** • **Fotolokation bearbeiten**
Primary Actor	SnapIt - Benutzer
Scope	SnapIt - Applikation
Level	Benutzerziel
Overview	Der Benutzer kann seine eigenen Fotos verwalten.
Stakeholders & Interests	**Fotograf**: Einfaches verwalten von seinen geknipsten Fotos **Anseher**: - **Fotomotiv**: - **Lokationsinhaber / Personal**: - **Informierer**: - **SnapIt**: Stabiles System. Korrektheit der Informationen. Möglichst geringer Verwaltungsaufwand.
Preconditions	Benutzer hat mindestens ein Foto gemacht.
Postconditions	Die gewünschte Verwaltungsoperation konnte erfolgreich durchgeführt werden.
Motivation	Der Benutzer kann im Nachhinein seine Fotos verwalten.
Main Success Scenario	1. Benutzer wählt Foto aus 2. Benutzer führt gewünschte Aktion auf dem Foto aus 3. System verarbeitet die Aktion 4. Applikation informiert den Benutzer über den Erfolg seiner Aktion
Extensions (Alternative Flows)	-
Special Requirements	-
Technology & Data	-

Variations List	
Frequency of Occurrence	ca. 400 Verwaltungszugriffe pro Abend
Open Issues	-

UC03 Fotos ansehen

Der Benutzer sieht sich Fotos von einem bestimmten Club an.

User-Stories	• **Aktuelle Stimmung der Öffentlichkeit mitteilen** • **Eigene Fotos auf der Webseite betrachten** • **Alle Fotos ansehen**
Primary Actor	SnapIt - Benutzer
Scope	SnapIt - Applikation
Level	Benutzerziel
Overview	Ein Besucher betrachtet die nach verschiedenen Kriterien geordneten Fotos im Browser
Stakeholders & Interests	**Fotograf**: - **Anseher**: Alle geknipsten Fotos jederzeit auf der Webseite abrufen **Fotomotiv**: Fotos des Motivs auf der Webseite abrufbar **Lokationsinhaber / Personal**: Präsentationsplattform für die in seiner Lokation geknippsten Fotos (Werbung) **Informierer**: Aktuelle Fotos nach bestimmten Kriterien betrachten **SnapIt**: Die Infrastruktur kann die Masse an Zugriffen handeln
Preconditions	Es wurde mindestens ein Foto publiziert
Postconditions	-
Motivation	Der Benutzer kann die geknipsten Fotos betrachten
Main Success Scenario	1. Benutzer öffnet die Webseite im Browser 2. Benutzer bestimmt die Anzeigekriterien 3. Fotos werden auf der Webseite angezeigt
Extensions (Alternative Flows)	Zu den gewählten Kriterien existiert kein Foto
Special Requirements	-
Technology & Data Variations List	-
Frequency of Occurrence	10'000'000 Seitenzugriffe im Monat
Open Issues	• Wie wird die Seite in Echtzeit mit neuen Fotos versorgt?

7 Technische Analyse

7.1 Lokalisierung

7.1.1 Ausgangslage

Die Server-Applikation ordnet vom User geknipste Fotos einer Lokation (Club/Bar/..) zu. Ein Punkt auf der Erde wird mittels Längen- und Breitengrad bestimmt. Um ein Foto einer Lokation zuzuordnen muss der Client mittels GPS (oder einer alternativen Technologie) seine momentane Position (Punkt) so exakt wie möglich bestimmen und diese mit dem Foto dem Server mitteilen. Dann sollte herausgefunden werden, an welcher Lokation sich der User zum Zeitpunkt des Fotoknipsens befunden hatte.

Probleme

Folgende Probleme wurden bei der Analyse identifiziert:

- Genauigkeit der Lokationsbestimmung auf dem Handy

Das Mobiltelefon liefert, je nach Telefoneinstellung, sehr ungenaue Positionsangaben. Es kann sein, dass der Benutzer Lokationsbestimmung mit GPS ausgeschaltet hat und dann kennt das Handy nur die letzte bekannte Position oder die Position des Sendemasten in der Zelle in der es sich befindet. Anhand dieser Angaben, lässt sich ein Foto nicht genau einer Lokation zuordnen.

- Genauigkeit bei der Erfassung der Lokationen

Eine Lokation kann nur dann exakt erfasst werden, wenn viele Koordinatenpunkte zu einem geschlossenen Polygon geformt werden. Dann kann exakt festgestellt werden, ob sich der Benutzer innerhalb dieser Lokation befindet. Jedoch mit der Einschränkung, dass die Koordinaten vom Standort des Benutzers auch exakt sein müssen.

- Zuordnung von Foto und Lokation

Die korrekte Zuordnung von einem Foto zu einer bestimmten Lokation kann nur erfolgen, wenn die Lokation komplett erfasst wurde und die Koordinaten des Fotos exakt sind. Kleinste Abweichungen können dazu führen, dass die Zuordnung nicht mit absoluter Sicherheit erfolgen kann.

- GPS-Signal in geschlossenen Räumen

Die meisten Lokationen befinden sich in einem Gebäude und die Lokationsbestimmung mittels GPS ist dann fast unmöglich. Somit kann keine korrekte Position ermittelt werden.

Lösungsmöglichkeiten

- Der Benutzer wählt seine Lokation selber

Der Client schickt seine genauste mögliche Position an den Server. Dieser ermittelt alle Lokationen in einem bestimmten Umkreis und schickt diese dann dem User zurück. Der Benutzer wählt dann die korrekte Lokation aus.

- Benutzer bestimmt seine Lokation selber

Der User bestimmt seine Lokation selber, ohne irgendeinen Einfluss vom Server. Das heisst, er gibt den Namen der Lokation manuell ein.

7.1.2 Lösung

Eine geeignete Lösung, welche nicht viel Aufwand mit sich bringt, ist, dass Lokationsstandorte nur mit einem Koordinatenpunkt erfasst werden und der User dann aus einer Liste von möglichen Lokationen seinen genauen Standort auswählen kann. Diese Lösung ist in ihrer Umsetzung einfacher und somit weniger fehleranfällig. Der Aufwand für das Erfassen von Lokationen reduziert sich auch auf ein Minimum.

Die folgende Grafik illustriert die Grundidee.

Abbildung 7-1: Club erfassen

Erklärung der Abbildung 7-1: Club erfassen

- Der Umriss stellt eine Lokation (Club) dar
- Die Position der Lokation wird mittels einem Koordinatenpunkt (rotes Kreuz) erfasst

Falls keine Koordinaten bestimmt werden können wird der Benutzer aufgefordert den Namen der Lokationen einzugeben. Diese wird auf der Serverseite verglichen und wenn kein passender Eintrag gefunden wurde wird das Foto keiner Lokation zugeordnet. Der Benutzer kann jederzeit die richtige Lokation hinzufügen. Ein weiterer Fall ist, wenn der Benutzer die Lokation wechselt. Hier wird beim Upload des Fotos eine falsche Lokation mitgeschickt. Wenn dies auf der Serverseite erkannt wird, erhält der Benutzer einen entsprechenden Hinweis, der ihn auffordert die Lokation neu einzugeben.

7.2 Client-Server-Kommunikation

Die Analyse der Client-Serverarchitektur bzw. des REST-Frameworks basiert auf dem Ansatz von Google, der an der Google I/O 2010 präsentiert wurde: http://www.youtube.com/watch?v=xHXn3Kg2IQE.

7.3 Anonymisierung

7.3.1 Ausgangslage

Die Hauptanforderung ist es, dass Bilder vollkommen anonym auf den Server geladen werden könne. Den Begriff anonym definieren wir wie folgt:

Alle auf dem Server gespeicherten Daten lasen keine Rückschlüsse auf den Benutzer, der das Bild heraufgeladen hat, zu.

7.3.2 Problem

Die Anforderung der anonymen Daten auf dem Server und der Anforderung, dass Benutzer trotzdem ihre Fotos löschen und relokalisieren können konkurrenzieren sich. Um Benutzer für eine der vorhin genannten Aktionen authentifizieren zu können braucht es ein eindeutiges gemeinsames Geheimnis. Dies wiederum verletzt die Anforderung eines anonymen Bilderuploads.

7.3.3 Lösung

Der Benutzer erhält einmalig eine vom Server generierte Identifikation.

Nach Buchmann ist diese Identifikation um die Zugangsberechtigung, die an eine bestimmte Identität gebunden ist, zu überprüfen. Dieses Verfahren wird Identifikationsprotokoll genannt und in diesem demonstriert der Beweiser dem Verifizierer, dass der Verifizierer gerade mit dem Beweiser kommuniziert oder kurz vorher mit ihm kommuniziert hat (vgl. [BUC04], S.227).

Diese Identifikation, nach Buchmann, schickt der Server dem Benutzer und speichert gleichzeitig im System den SHA-256-Hash der Identifikation persistent ab. Somit kennt der Server die ursprüngliche Identifikation nicht mehr. Der Client kann sich mit der Identifikationsnummer bei jeder Anfrage indentifizieren. Der Server kann dann einen weiteren Hash generieren und überprüfen, ob dieser mit dem im System übereinstimmt (vgl. [BUC04], S.228).

7.4 Userinterface

Navigation Map

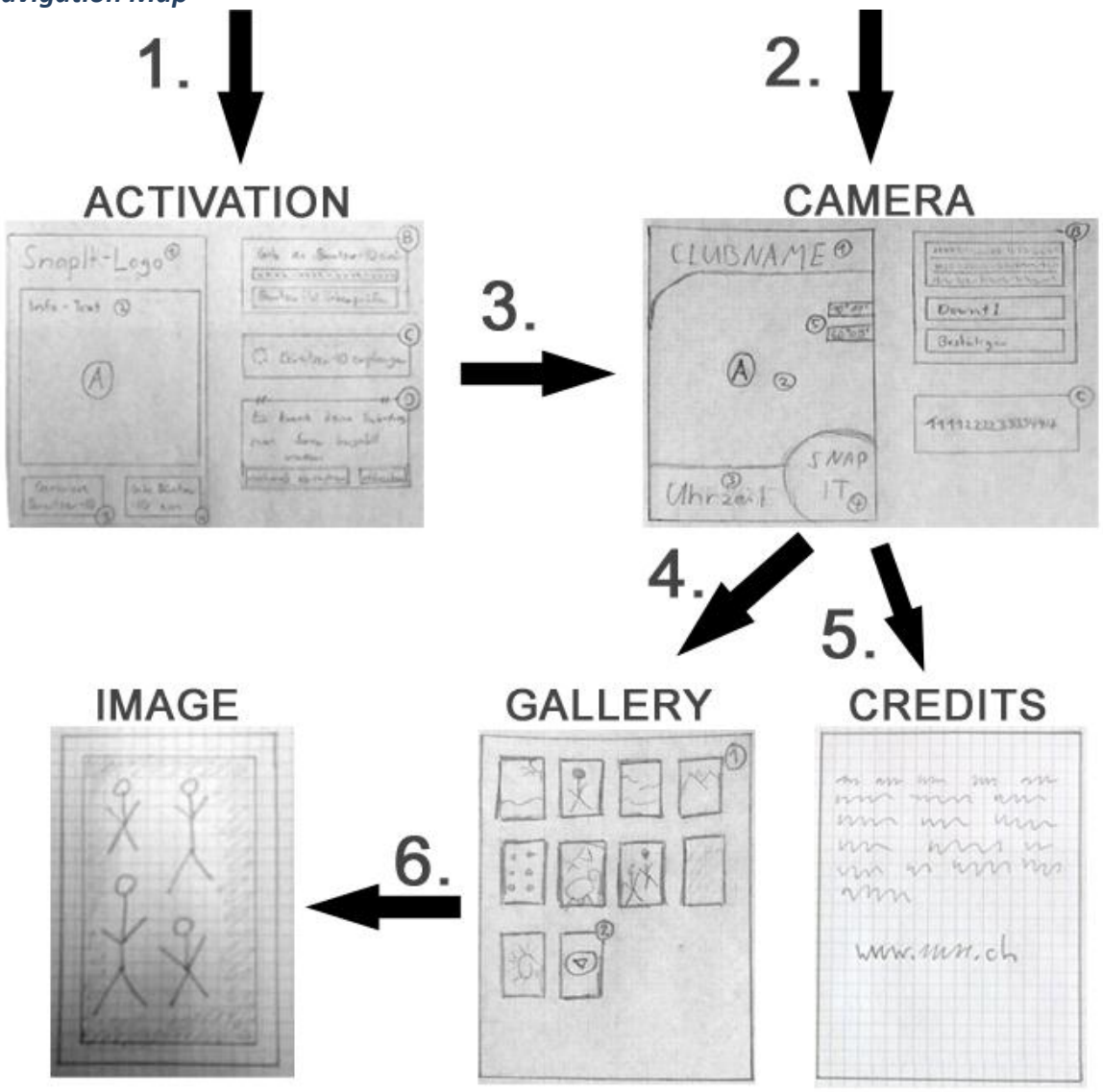

Abbildung 7-2: GUI-Navigation-Map

1. Beim ersten Starten der App ist der Activation-Screen der Einstiegspunkt. Da es nicht möglich ist, die App ohne eine Benutzer-ID zu nutzen und die Benutzer-ID nach der Aktivierung auf dem Smartphone gespeichert bleibt, wird dieser Screen genau einmal dargestellt.
2. Für jeden weiteren Einstieg wird direkt die Camera-Ansicht angezeigt. Somit gelangt der Benutzer mit nur einem Klick zum Kernfeature der App. Die Camera-View dient zudem als Ausgangspunkt um weitere Screens zu erreichen.
3. Nach der einmaligen Aktivierung der App, gelangt man vom Activation-Screen direkt auf die Camera-Ansicht.
4. Der Gallery-Screen ist über das Menü erreichbar. Auch ein Klick auf die Upload-Meldung führt zu diesem Ansichtswechsel.
5. Die Credits sind über das Menü des Camera-Screens erreichbar. Somit fungiert die Camera-Ansicht als zentraler Haupt-Screen.

6. Mit einem Klick auf ein Miniatur-Foto wird das Bild gross auf dem Screen angezeigt.

Paperprototyps

Activation

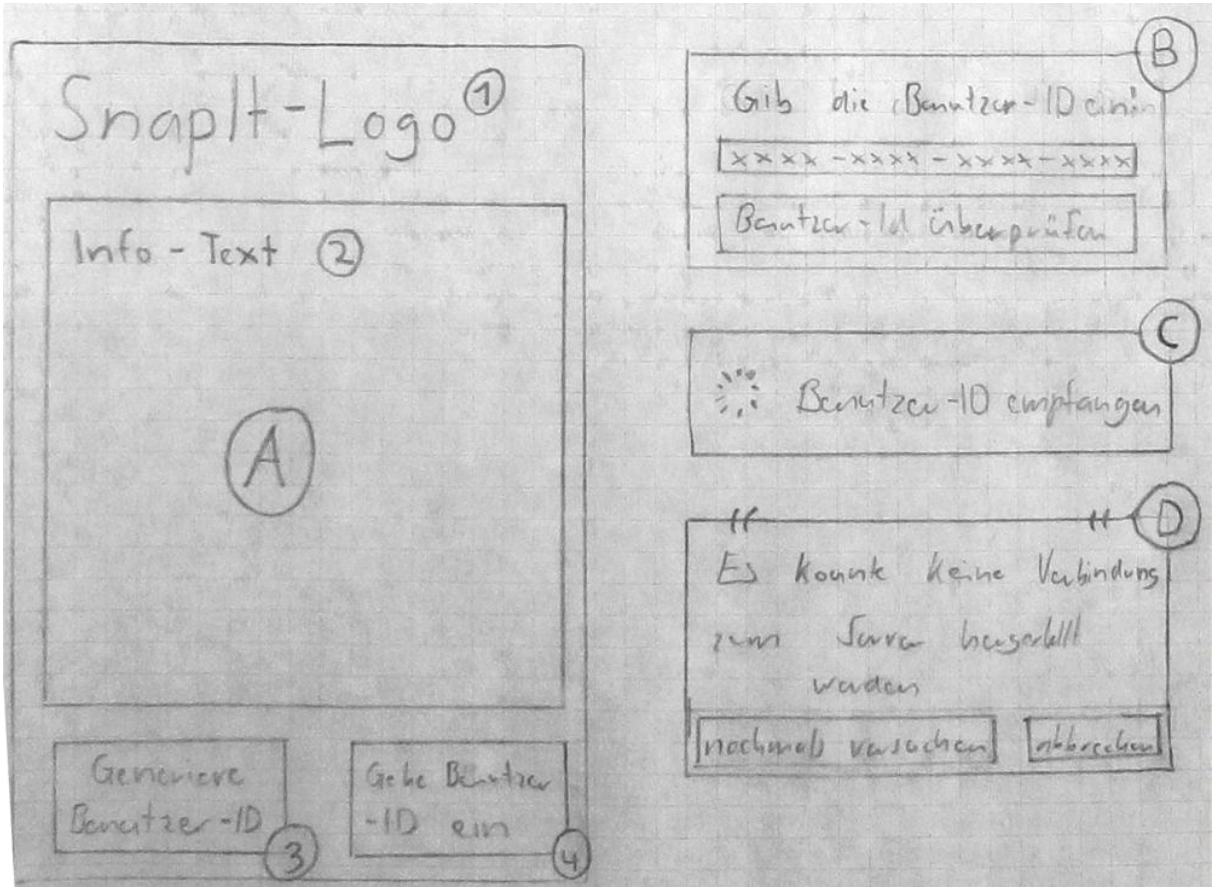

Abbildung 7-3: GUI-Activation

A. Screen: Activation
   1. SnapIt-Logo
   2. Info-Text
   3. Button: Generiere Benutzer-ID
   4. Button: Gebe Benutzer-ID ein
B. Dialog: Eingabe Benutzer-ID
C. Dialog: Benutzer-ID empfangen
D. Dialog: Keine Verbindung

Camera

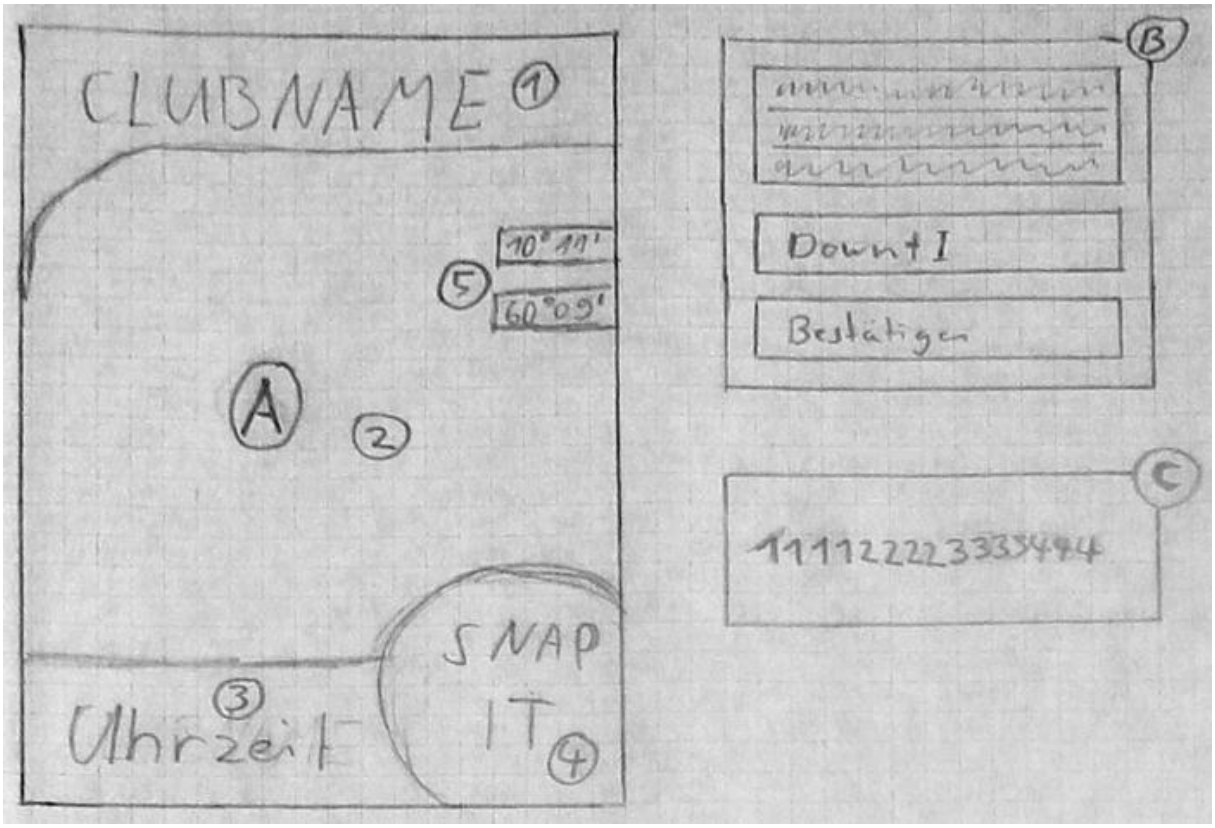

Abbildung 7-4: GUI-Camera

A. Screen: Camera
    1. Lokations-Name
    2. Kamera
    3. Uhrzeit
    4. Button: Foto knippsen „Snap it“
    5. Geo-Koordinaten
B. Dialog: Lokation bestimmen
C. Dialog: Benutzer-ID anzeigen

Gallery

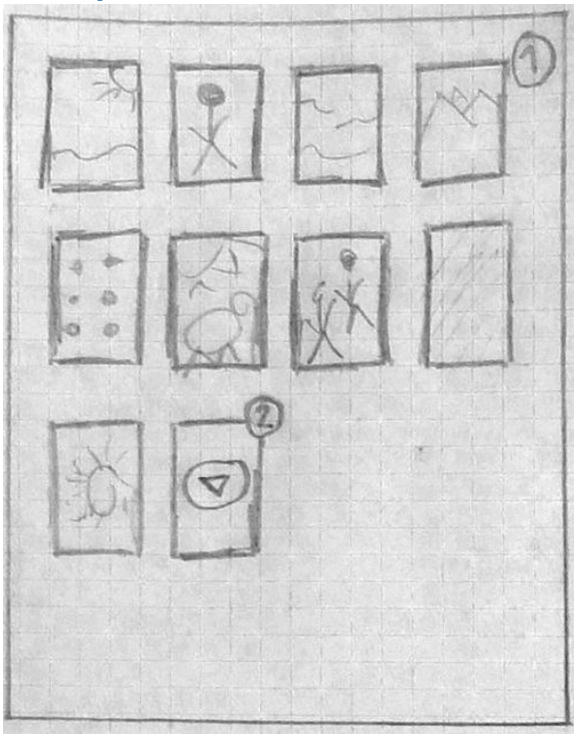

Abbildung 7-5: GUI-Gallery

A. Screen: Gallery
   1. Fotos
   2. Button: Weitere Fotos anzeigen

Credits

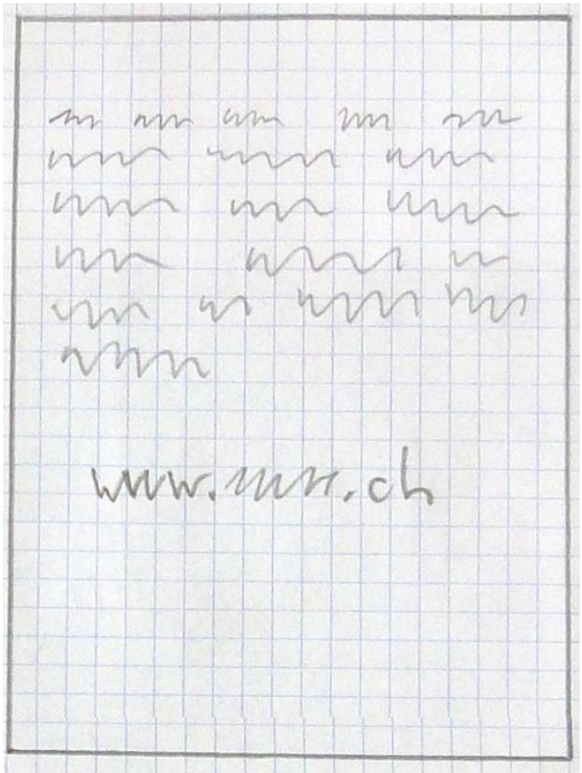

Abbildung 7-6: GUI-Credits

Image

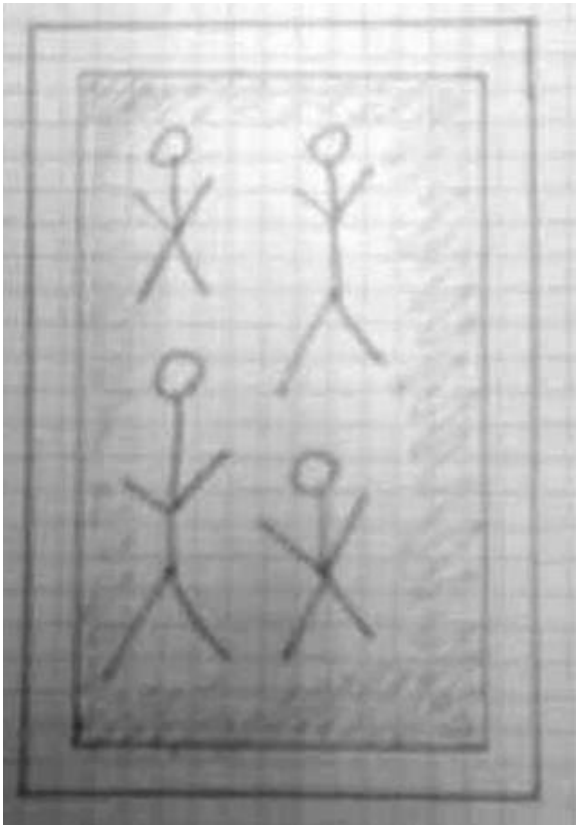

Abbildung 7-7: GUI-Image

Accessibility

Die App wurde nicht für Personen mit Einschränkungen optimiert.

7.4.1 Screens

Die folgenden Texte und Printscreens zeigen ein Walkthrough durch die finale App. Usability-Entscheidungen die auf Grund der Usability-Tests entstanden sind, wurden ebenfalls hier notiert.

Activation

Abbildung 7-8: GUI-Activation-Screen

Beschreibung

- Der Aktivierungsscreen wird beim ersten Starten der Applikation angezeigt.
- Die beiden Textabschnitte informieren den Benutzer einerseits kurz über den Nutzen der Applikation und andererseits über den Sinn der Benutzer-ID.
- Der Benutzer kann von diesem Screen aus eine Benutzer-ID generieren oder eine bestehende eingeben.

Usability- Entscheide

- Nutzen der Benutzer-ID kurz beschrieben, mit Hinweis auf die Anonymität.

Beschreibung

- Nach dem Klicken auf „ID generieren“ wird eine ID vom Server empfangen.
- Nach dem Vorgang wird die Kamera-Ansicht angezeigt.
- Sollte ein unerwarteter Fehler auftreten oder die Verbindung nicht vorhanden sein, wird der Benutzer informiert.

Abbildung 7-9: GUI-Activation-Screen: Benutzer-ID generieren

Abbildung 7-10: GUI-Activation-Screen: Benutzer-ID eingeben

Beschreibung

- Nach dem Klicken auf den Button "ID eingeben" erscheint diese Maske.
- Der Benutzer wird über die Eingabe einer ungültigen Benutzer-ID informiert.
- Wurde eine gültige ID eingeben erscheint die Kamera-Ansicht.
- Sollte ein unerwarteter Fehler auftreten oder die Verbindung nicht vorhanden sein, wird der Benutzer informiert.

Usability- Entscheide

- Anzeigen des Eingabeformats als Hint, damit Benutzer erkennt um welche Eingabe erwartet wird.

Camera

Abbildung 7-11: GUI-Camera-Screen: Lokationen laden

Beschreibung

- Sobald die Kamera-Ansicht angezeigt wird, lädt das Programm die nahegelegenen Lokationen vom Server.

Usability- Entscheide

- Der Ladevorgang ist jederzeit abbrechbar.

Beschreibung

- Die ermittelten Lokationen können aus einer Liste ausgewählt werden. Ist die Lokation nicht vorhanden, kann auch eine neue bzw. eine nicht aufgelistete Lokation eingetragen werden

Usability- Entscheide

- Die Eingabe eines Namens bewirkt, dass nur die Lokationen angezeigt werden, die den eingegeben String enthalten. Dies verbessert die Übersicht.

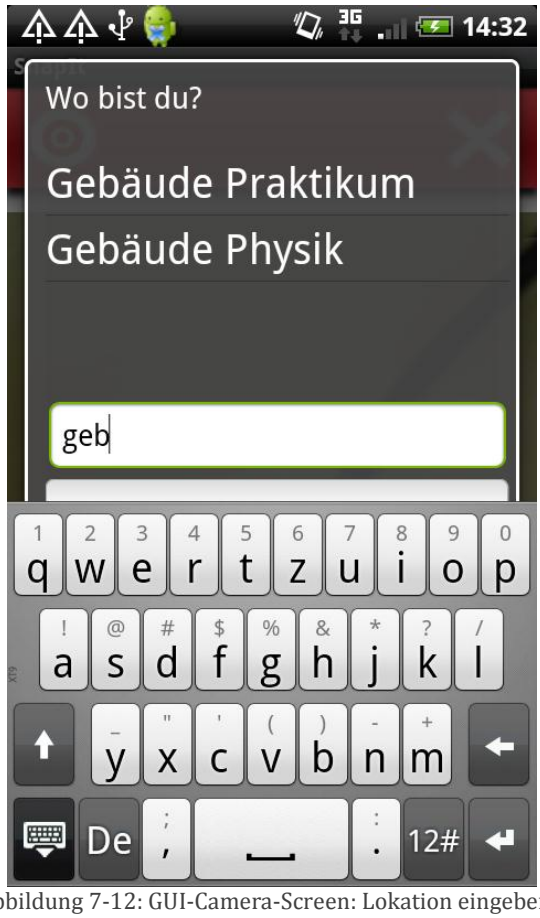

Abbildung 7-12: GUI-Camera-Screen: Lokation eingeben

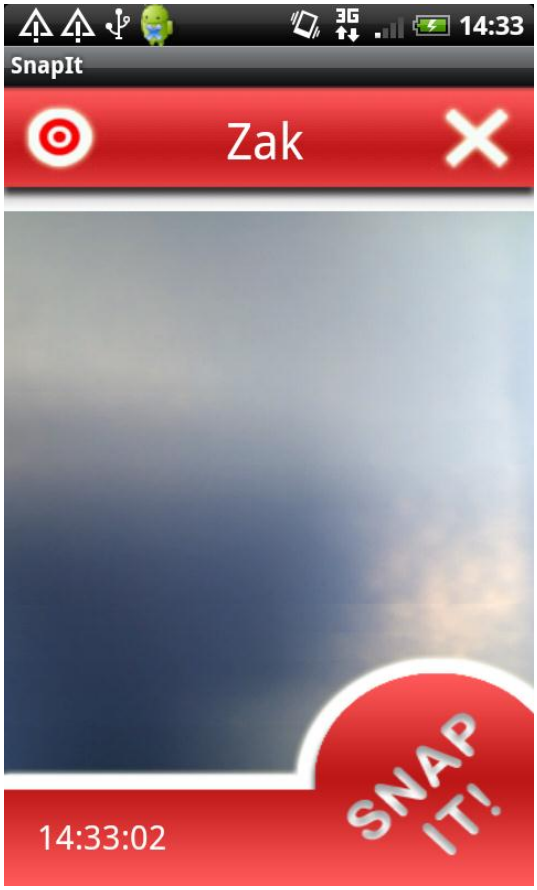

Abbildung 7-13: GUI-Camera-Screen: Foto knipsen

Beschreibung

- Die gewählte Lokation wird im obereren Teil des Screens angezeigt.
- Mit einem Klick auf den Namen oder das linke Icon kann die Lokation jederzeit neu bestimmt werden.

Usability- Entscheide

- Das Kreuz in der oberen rechten Ecke bewirk, dass die Lokation als aktueller Auftenhaltsort entfernt wird. Ein Benutzer kann so jederzeit die Bilder ohne Lokationsangaben uploaden.
- Auf die Angabe der Geo-Koordinaten, wie sie in den Paperprototypes geplant war, wurde verzichtet. Die Geokoordinaten helfen dem Benutzer nur sehr wenig und der Screen wirkt sonst zu überladen.

Beschreibung

- Das Menü, welches über einen Klick auf die Standart-Menü-Taste angezeigt wird, enthält zwei Icons: Das “Benutzer ID anzeigen” und das Icon “Bilderübersicht”.

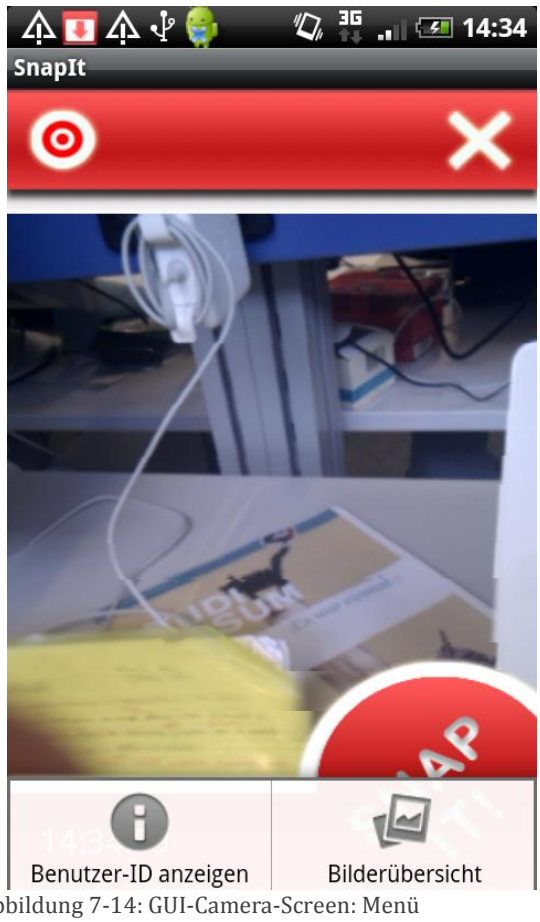

Abbildung 7-14: GUI-Camera-Screen: Menü

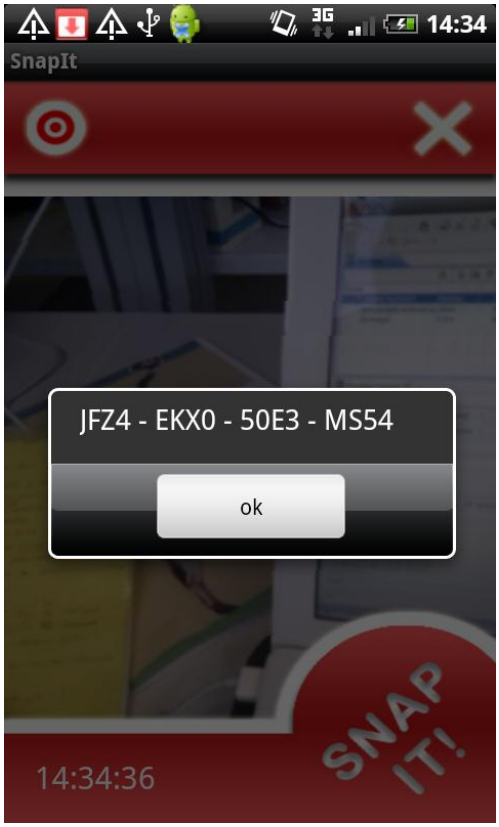

Abbildung 7-15: GUI-Camera-Screen: Benutzer-ID anzeigen

Beschreibung

- Die Benutzer-ID wird nach dem Klick auf das Icon “Benutzer-ID anzeigen” in einem Popup dargestellt.

Usability-Entscheide

- Die Benutzer-ID wird mit Bindestrichen getrennt dargestellt. Dies erhöht die Lesbarkeit.

Beschreibung

- Die Anzahl der übermittelten und ausstehenden Fotos (also Fotos die auf Grund von fehlender Verbindung nicht heraufgeladen werden können) wird in der Notification-Leiste angezeigt.
- Die Notification verschwindet sobald man auf sie klickt. Ein Klickt führt zu einem Wechsel auf die “Gallery” Ansicht.

Usability-Entscheide

- Der Benutzer enthält sonst kein Feedback, da er sich während der Party nicht um das Uploaden der Fotos kümmern will.

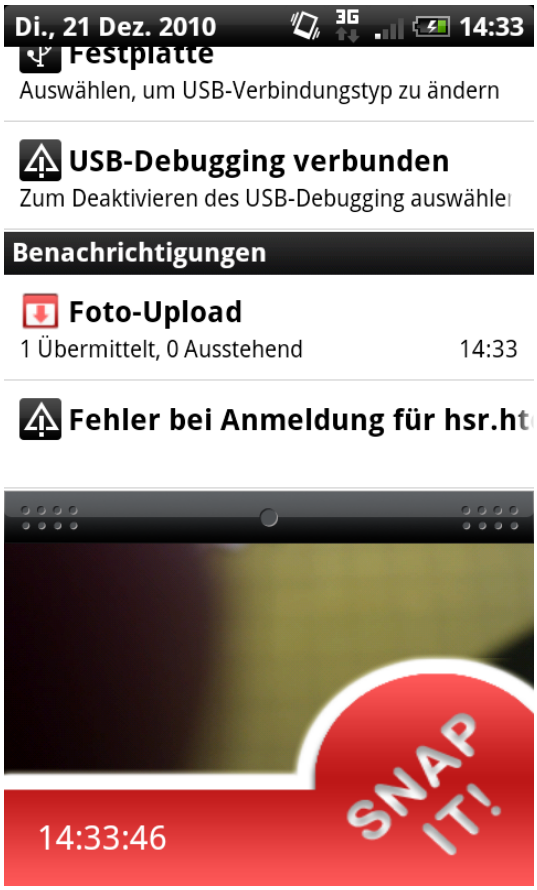

Abbildung 7-16: GUI-Camera-Screen: Benachrichtigung

Gallery

Abbildung 7-17: GUI-Gallery-Screen: Fotos laden

Beschreibung

- Beim Öffnen der Galerie wird ein Ladebalken angezeigt.
- Sollte ein unerwarteter Fehler auftreten oder die Verbindung nicht vorhanden sein, wird der Benutzer informiert.

Beschreibung

- Nach dem Laden werden die Bilder als Miniaturen angezeigt.
- Die kleinen grünen und gelben Icons zeigen an, ob die Fotos einer korrekten Lokation zugeordnet werden konnten.
- Sind mehr als elf Fotos vorhanden erscheint ein Icon, das beim Anklicken weitere elf Fotos nachlädt.

Usability-Entscheide

- Das Design passt sich beim Drehen des Displays an. Dies wurde von den Benutzern begrüsst, da sich das als Standard bei solchen Ansichten etabliert hat.

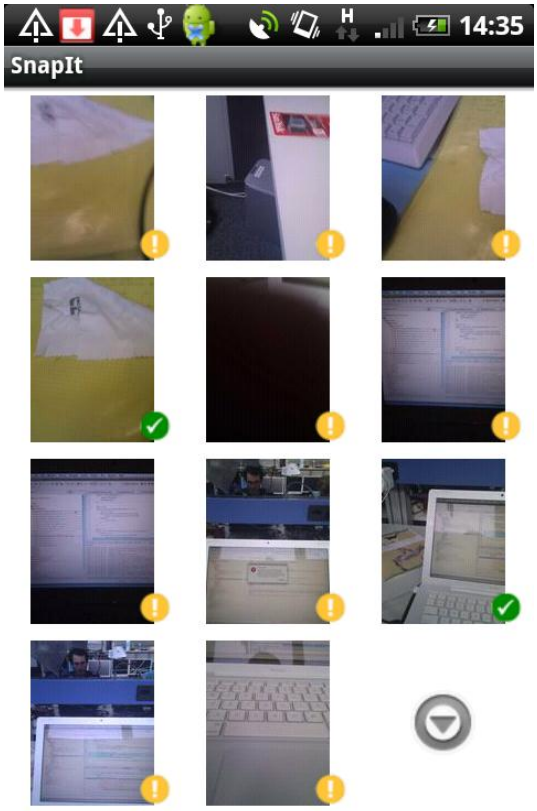

Abbildung 7-18: GUI-Gallery-Screen

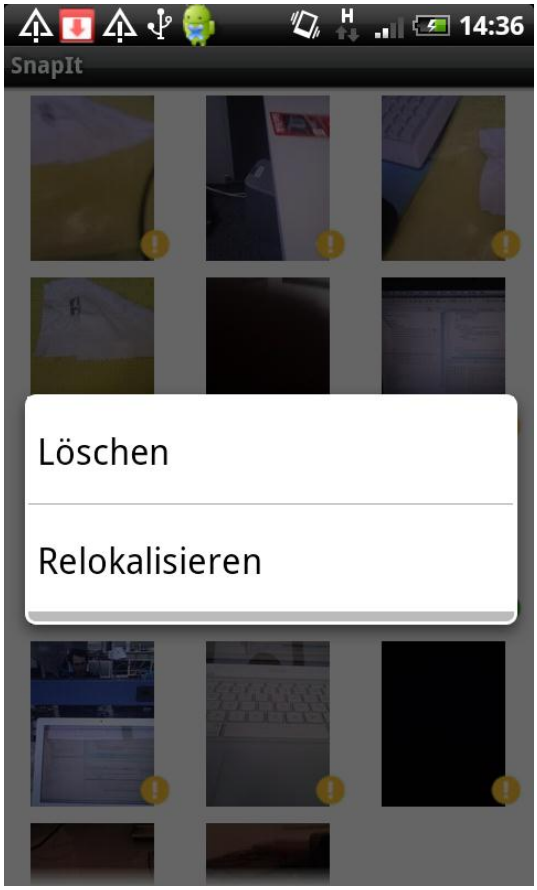

Abbildung 7-19: GUI-Gallery-Screen: Kontext-Menü

Beschreibung

- Ein Long-Klick auf ein Foto ruft das Kontext-Menü auf, welches die zwei Features “Löschen” und “Relokalisieren” anbietet.
- Beim Löschen wird das Bild nach dem Vorgang aus der Ansicht entfernt.
- Beim Relokalisieren erhält der Benutzer ein Feedback in Form einer Meldung. Ausserdem wird gegebenfalls die Farbe des Icons angepasst.
- Sollte ein unerwarteter Fehler auftreten oder die Verbindung nicht vorhanden sein, wird der Benutzer informiert.

Beschreibung

- Das Menü beinhaltet drei Icons: Das Kamera-Icon bringt den Benutzer zurück zur Kamera, das Credits-Icon führt auf die Credits-Seite und der Aktualisieren Button lädt alle Bilder erneut

Usability-Entscheide

- As Kamera- und Aktualisieren-Icon wurde nachträglich hinzugefügt:
  - Obwohl die Kamera über den Backbutton erreichbar ist, wurde dieser Menüeintrag als Ergänzung von vielen Benutzern erwünscht. Speziell Leute die mit dem Umgang mit Android-Smartphones nicht vertraut sind, äusserten diesen Wunsch.
  - Der Aktualisieren-Button gibt dem Benutzer die Möglichkeit zu jederzeit die neusten Fotos anzuzeigen,

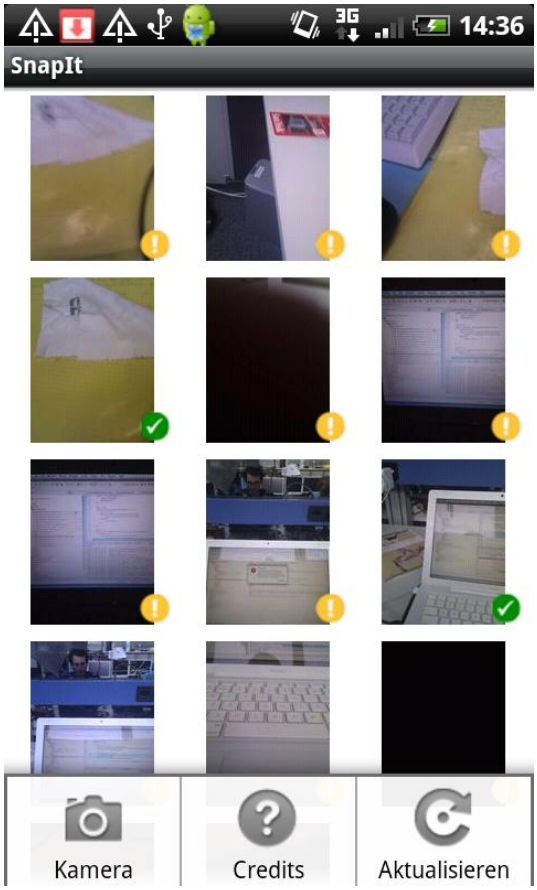

Abbildung 7-20: GUI-Gallery-Screen: Menü

auch solche die während dem Verweilen in der Galerie im Hintergrund komplett heraufgeladen wurden.

Credits

Beschreibung

- Die Credits-Seite zeigt Infos über die Entwickler an. Ein Link führt zur Webseite, wo die Fotos in Echtzeit angezeigt werden.

Abbildung 7-21: GUI-Credits-Screen

Image

Beschreibung

- Die Grossansicht eines Fotos erfordert erneut einen Server Zugriff. Auch diese Wartezeit wird durch das Einblenden einer Fortschritts-Anzeige überbrückt.
- Neben dem Foto werden der Aufnahmezeitpunkt und die Lokation, zu welcher das Foto zugeteilt wurde, angezeigt.
- Sollte ein unerwarteter Fehler auftreten oder die Verbindung nicht vorhanden sein, wird der Benutzer informiert.

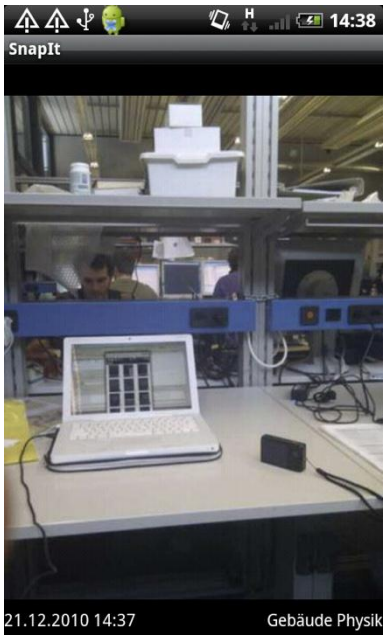

Abbildung 7-22: GUI-Image-Screen

8 Architektur

8.1 Client-Server Architektur

Dieser Abschnitt beschreibt die Architektur von der Kommunikation zwischen Client und dem REST-Server-Programm. Es wird nicht auf das Protokoll (Meldungsaustausch) eingegangen. Dies wird im Kapitel „Webservice Extension Developer Handbuch“ beschrieben.

Die folgende Grafik zeigt den Ablauf einer Anfrage.

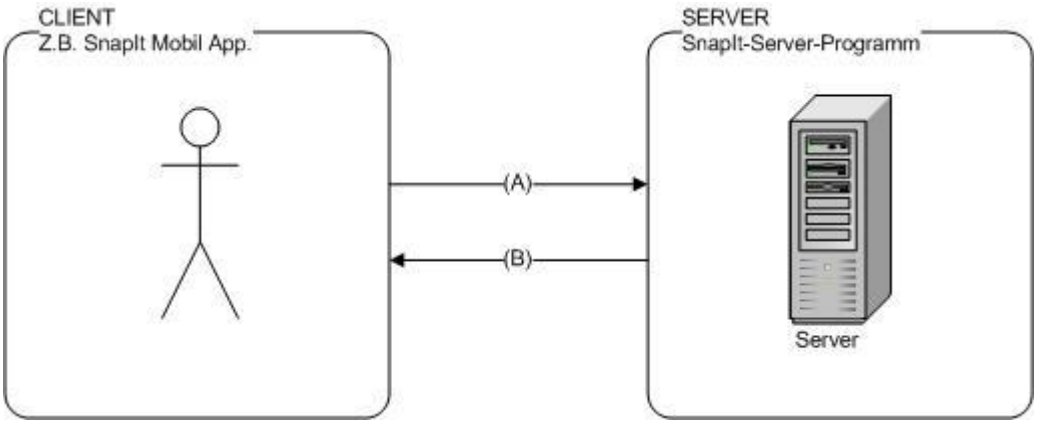

Abbildung 8-1: REST, Request – Response

Der Client startet eine Anfrage, diese wird dann über das HTTP-Protokoll, welches auf dem IP-Protokoll aufsetzt abgeschickt. Diese Anfrage wird vom SnapIt-Server-Programm entgegen genommen und verarbeitet. Danach schickt dieser dem Client eine entsprechende Antwort. Detailliertere Informationen über die HTTP Spezifikation findet man auf IETF HTTP 1.1.

Beschreibung zur Grafik:

(A)

- Der Client führt einen der folgenden Anfragen aus: GET, POST, PUT, DELETE.
- Diese Anfragen beziehen sich immer auf eine bestimmte Web-Ressource z.B. ein Bild. Diese basieren auf dem REST-Ansatz von Roy Fielding.
- Die Anfragetypen unterscheiden sich nur ein wenig. Folgende Aufzählung soll zum Verständnis beitragen:
  - GET
    - Als Query-Parameter sollte eine Transaktions-ID (Parameter-Name: transID) vom Typ Integer32 mitgegeben werden. Diese wird in der Antwort des Servers mitgeschickt.
    - Die Anfrage sollte kein Body haben, dieser wird ignoriert.
  - POST
    - Hier wird, wie bei der GET-Anfrage, eine Transaktions-ID benötigt (siehe GET). Auch hier wird diese in der URL als Parameter mitgeschickt.
    - Zusätzlich sollte ein String im Body der Anfrage als JSON-String mitgeschickt werden. Dieser kann verschiedene Informationen enthalten.
  - PUT
    - Hier wird, wie bei der GET-Anfrage, eine Transaktions-ID benötigt (siehe GET). Auch hier wird diese in der URL als Parameter mitgeschickt.
    - Zusätzlich enthält der Body ein JSON-String mit verschiedenen Informationen.
  - DELETE
    - Gleich wie GET, nur das eine Ressource gelöscht wird.

(B)

- Die Antwort auf eine Anfrage.
- Diese bezieht sich immer auf die vorherige Anfrage.
- Jede Antwort enthält im HTTP-Body ein JSON-String mit spezifischen Informationen,. Es sollte immer eine Transactions-ID mitgeschickt werden, die sich auf die Anfrage bezieht.

8.1.1 Client

Die implementierte REST-Architektur stützt sich grösstenteils auf dem Vortrag von der Google I/O 2010 (Google I/O 2010 - Android REST client applications). Der grosse Unterschied ist, dass diese Architektur mit dem Ziel der Wiederverwendbarkeit implementiert wurde.

Das Pseudo-Package-Diagramm zeigt die Bestandteile der Client-REST Architektur:

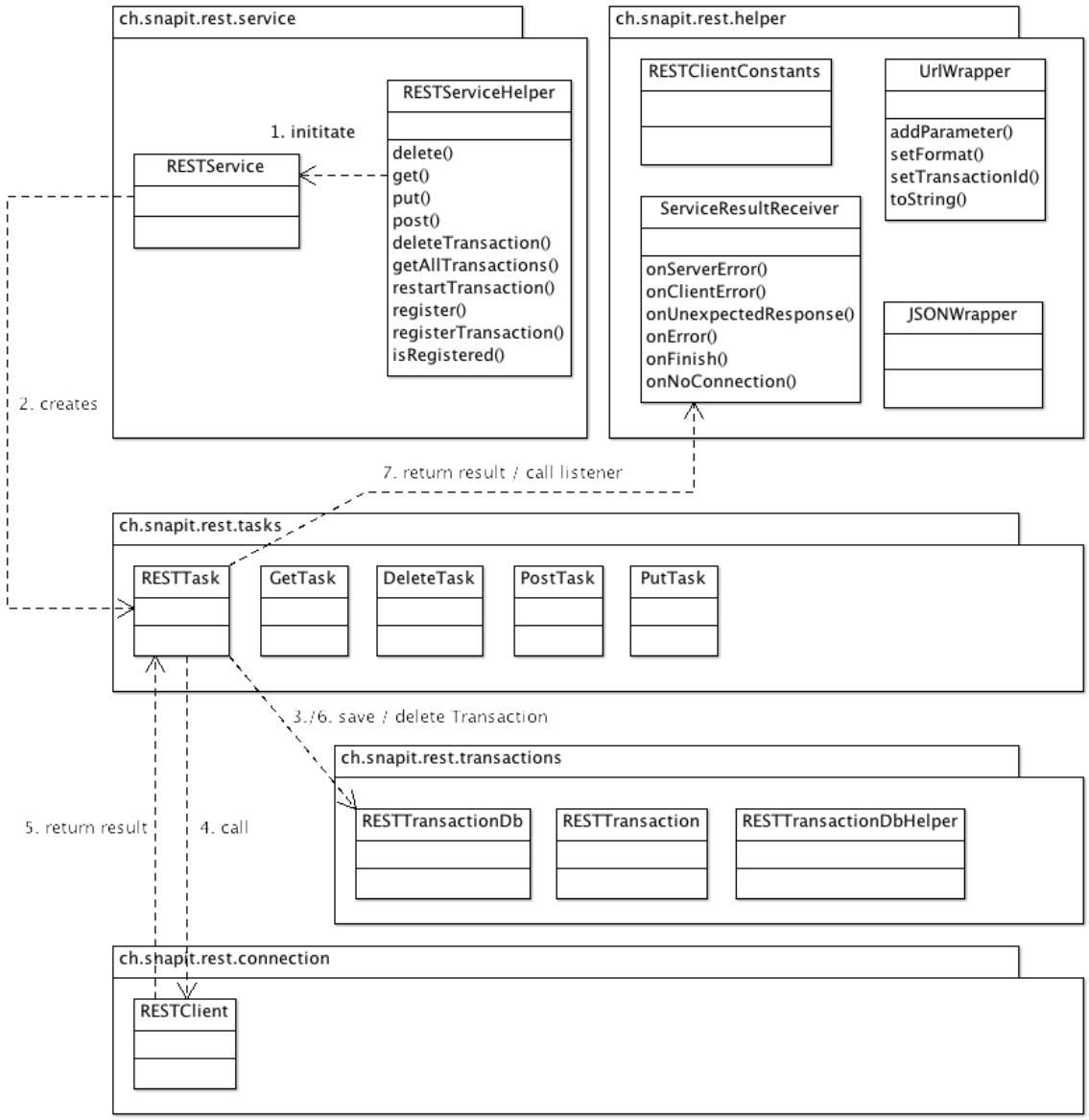

Abbildung 8-2: Package Diagramm - REST - Client

Ablauf

Die folgende Aufzählung beschreibt den Ablauf der serverseitigen Kommunikation:

1. Will eine Activity auf Daten vom Server zugreifen oder Daten an den Zielserver übermitteln ruft sie die gewünschte REST-Methode auf einer RESTServiceHelper-Instanz auf. Dem Helper übergibt sie sämtliche essentiellen Informationen, dazu gehören ein ResultReceiver und die Ziel-URL. Bei POST und PUT kann zusätzlich ein JSON-Wrapper übergeben werden, der die zu übermittelnden Daten enthält.
2. Im RESTServiceHelper werden die Informationen zerlegt und daraus ein Intent erstellt. Zur Überwachung der Transaktion über sämtliche Schichten (auch auf dem Server) wird eine eindeutige Transaktions-Nummer erstellet und ebenfalls dem Intent übergeben. Der Intent wird dann an den RESTService gesendet, sofern eine Datenverbindung besteht. Ansonsten wird der Vorgang frühzeitig beendet und der Aufrufer über den ResultReceiver informiert.
3. Ist der Service gestoppt, wird er gestartet, ansonsten bleibt er aktiv. Der Service ruft nun abhängig von der REST-Methode einen asynchronen Task auf.

4. In REST-Methoden spezifischen asynchronen Task wird die Activity über das Starten informiert und kann dadurch darauf reagieren. Die Transaktion wird dann im RestTransactionDb-ContentProvider persistent gespeichert. Somit kann die Transaktion überwacht und der Status abgefragt werden. Der Task ruft nun die entsprechende RESTClient-Methode auf.
5. Die Klasse RESTClient sendet nun den Request über HTTP an den Server und empfängt dann das Resultat, welches dann wieder an den aufrufenden Task zurückgeben wird.
6. Der Task empfängt die Antwort und löscht die Transaktion aus der Datenbank wenn es sich um einen GET- oder DELETE-Request handelte. POST- und PUT-Request müssen manuell entfernt werden (das Framework stellt eine solche Methode zur Verfügung), somit besteht die Möglichkeit diese Transaktionen erneut zu starten, sollte das Resultat fehlerhaft sein. Zum Schluss wird der aufrufenden Activity das Resultat übergeben und der definierte ResultReceiver aufgerufen.
7. Die Activity kann nun das empfangene JSON interpretieren und die Daten verarbeiten.

Treten auf der Client-Seite Fehler auf, wird die Activity informiert. Diese kann dann entsprechend reagieren.

Komponenten

ch.snapit.rest.service

Die ganze REST-Kommunikation läuft über das Service-Package um das Ganze von der Businesslogik zu trennen.

- **RESTService**
  - Der RESTService trennt die Datenkommunikations-Logik von der Businesslogik. Der Service wird von Android speziell behandelt: Sollten Applikationselemente aufgrund von fehlender CPU Leistung oder Memory vom BS gestoppt werden, wird eine Activity mit laufendem Service nur in extremen Fällen gestoppt (siehe API Service). Für jeden Request wird dem Service ein Intent gesendet. Der Intent wird dann verarbeitet und der asynchrone Task gestartet, welcher zu der entsprechenden REST-Methode passt. Sobald keine Transaktionen mehr im Gange sind, wird der Service wieder gestoppt und somit seine priorisierte Stellung gegenüber dem BS aufgehoben.
- **RESTServiceHelper**
  - Die Klasse RESTServiceHelper vereinfacht den Aufruf des Services. Für jede REST-Methode existiert eine Java-Methode welche aus der Activity aufgerufen werden kann um einen Request zu starten. Im RESTServiceHelper werden aus den übermittelten Informationen Intents erstellt und diese zum Starten des Services versendet. Diese Methoden liefern der Activity die Transaction-Id, anhand welcher der Status einer Transaction jederzeit abgefragt werden kann.

ch.snapit.rest.tasks

Dieses Package stellt alle Funktionalität zur Verfügung um die REST-Operationen parallel auszuführen.

- **RESTTask**

  - Der RESTTask ist verantwortlich dafür, dass jeder Request in einem eigenen Thread läuft. Damit werden längere Verzögerungen im GUI verhindert. Er übernimmt auch das Loggen der Transaktionen im ContentProvider (siehe RestTransactionDb) und informiert den "ServiceResultReceiver" über die erfolgreiche oder fehlerhafte Ausführung einer Anfrage.
- **Delete-, Put-, Get- & PostTask**
  - Diese Klassen rufen die korrekten REST-Methoden des RESTClients auf. Es sind alles Subklassen von RESTTask.

ch.snapit.rest.transaction

Das Transaction-Package stellt Klassen zum Speichern der Transaktionen auf dem Smartphone zur Verfügung. Dadurch werden die losen Requests zentral verwaltet. Es können Statusabfragen und -änderungen vorgenommen werden.

- **RESTTransaction**
  - Die RestTransaction ist eine Container-Klasse, die alle Informationen hält die einen Request eindeutig identifizieren. Die Klasse wird als Transferobjekt zwischen dem Service und dem Contentprovider verwendet. Im ContentProvider werden Datensätze von diesem Typ gespeichert.
- **RESTTransactionDb**
  - Die RESTTransactionDb ist ein ContentProvider der auf eine Datenbank zugreift. Hier werden alle Transaktionen registriert und nach erfolgreichem Ausführen wieder gelöscht. An dieser zentralen Stelle, kann der Verkehr zwischen Server und Client jederzeit abgefragt werden.
- **RESTTransactionDbHelper**
  - Der RESTTransactionDbHelper beinhaltet Hilfsmethoden zur Erstellung der Datenbank.

ch.snapit.rest.RESTClient

Der RESTClient führt die REST-Anfragen über das HTTP-Protokoll aus und Empfängt die Bytes vom Server. Der Rückgabewert aller Methoden ist ein JSON-String der später in ein JSON-Objekt geparst werden kann.

ch.snapit.rest.helper

Das Helperpackage enthält diverse Klassen, die einerseits das verwenden der Kommunikationsschnittstelle vereinfachen und andererseits mehrfach verwendete Hilfsmethoden bündeln. Folgende Klassen werden verwendet:

- **JSONWrapper**
  - Die Android JSON-Implementierung wirft bei vielen Methoden Exceptions. Damit diese Exceptions nicht jedes Mal beim Erstellen und Empfangen von JSON-Informationen erneut behandelt werden müssen, übernimmt das die Wrapper-Klasse. Die Wrapperklasse setzt bei fehlendem Inhalt Standartwerte und loggt unerwartete Fehler. Die Klasse erleichtert die Arbeit mit JSON und sorgt für besser lesbaren Quellcode.
- **RESTClientConstants**
  - Über die primitiven Schnittstellen werden die Daten anhand von Bundles übertragen. Ein Bundle ist ein spezieller Key-Value-Container der serialisierbar ist. Die RESTClientConstants-Klasse enthält Konstanten die

als Keys eingesetzt werden. Somit wird die Fehlerwahrscheinlichkeit vermindert und das Programmieren vereinfacht.

- **ServiceResultReceiver**
  - Nach dem abarbeiten eines REST-Request wird die Activity über den ServiceResultReceiver informiert. Diese abstrakte Standard Implementation fungiert als Template-Funktion. Durch das Überschreiben der Klasse, können Methoden für den Fehler- und Erfolgsfall konkret implementiert werden. Weitere Events können ohne grossen Aufwand hinzugefügt werden.
- **UrlWrapper**
  - Der UrlWrapper dient als Baukasten für die Request-URLs. Die verschiedenen Methoden erlauben es die Domain, den Pfad, die Attribute und auch den Rückgabetype zu definieren. Der Vorteile ist, dass kein Vorwissen über den Aufbau der URLs vorhanden sein muss, um die Zieladresse für einen Request zu definieren. Ausserdem werden so die String-Operationen an einem Ort performant ausgeführt.

Schnittstelle

Die REST-Architektur kann grundsätzlich von einer beliebigen Applikation genutzt werden um via REST mit einem Server zu kommunizieren. Die Schnittstelle zur Kommunikation bildet die Klasse RESTServiceHelper, über deren öffentliche Methoden die ganze Funktionalität zur Verfügung gestellt wird (siehe JavaDoc).

Funktionalität

Folgende Funktionalität stellt die REST-Client-Architektur in der Version 1.0 zur Verfügung:

- Senden und Empfangen von GET-, POST-, PUT- und DELETE-Requests/Reponses
- Senden und Empfangen von Text und Bildern im JSON-Format
- Alle Transaktionen können jederzeit abgefragt werden
- Neustaren und Löschen von Transaktionen
- Datenverkehr wird in einem eigenen Service und in separaten Threads ausgeführt
- Es können für jede Transaktion folgende Listener implementiert werden
  - **onServerError** - Wird von onError aufgerufen, wenn serverseitig ein unerwarteter Fehler auftrat
  - **onClientError** - Wird von onError aufgerufen, wenn clientseitig ein unerwarteter Fehler auftrat
  - **onUnexpectedResponse** - Wird normalerweise nicht implementiert. Wird aufgerufen, wenn Transaktion bereits verarbeitet wurde, wenn Antwort eintrifft (Tritt auf, wenn die selbe Transaktionen mehrmals aufgerufen wurde)
  - **onNoConnection** - Wird aufgerufen wenn keine Verbindung zum Internet besteht oder der Server nicht erreichbar ist
  - **onError** - siehe onClientError und onServerError
  - **onFinish** - Wird aufgerufen, wenn die Daten erfolgreiche versendet und empfangen wurden

Folgende Funktionalitäten sind noch nicht implementiert, aber für zukünftige Versionen denkbar:

- **Caching**
  - Daten können und werden gewöhnlich von der Applikation nur abgefragt, wenn Sie noch nicht vorhanden sind. Das Caching würde dies vereinheitlichen und in gewissen Fällen nur neue Daten laden.
- **Abfragen des Fortschritts**
  - Die Apache Implementierung der REST-Methoden erlaubt in der neusten Version nicht mehr das Abfragen des Fortschritts. Die Lösung wäre das verwenden einer älteren Version, was aber die Dateigrösse immens erhöhen würde (da die Library seperat eingebunden werden müsste) und auf die Features und Stabilität der neuen Version verzichtet werden müsste.
- **Sicherheit**
  - Ist für unsere Applikation nicht sehr wichtig, könnte aber für die Wiederverwendung der REST-Architektur implementiert werden (niedrige Priorität)

8.1.2 Server

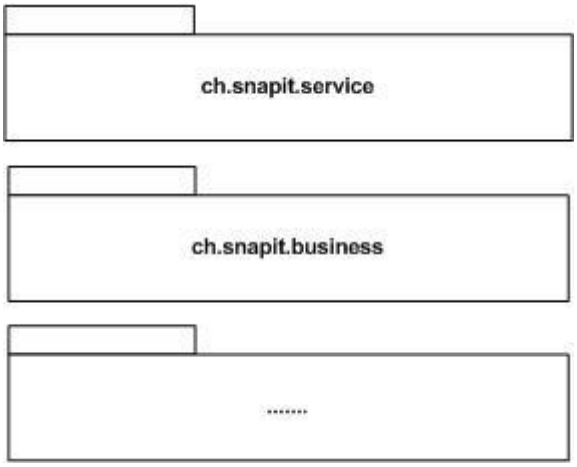

Abbildung 8-3: Package Übersicht - Server

Diese Grafik zeigt die zwei interessantesten Schichten der REST-Server-Architektur. (Service- und Businesslayer)

Das folgende Sequenzdiagramm zeigt den Ablauf bei einer Anfrage vom Client.

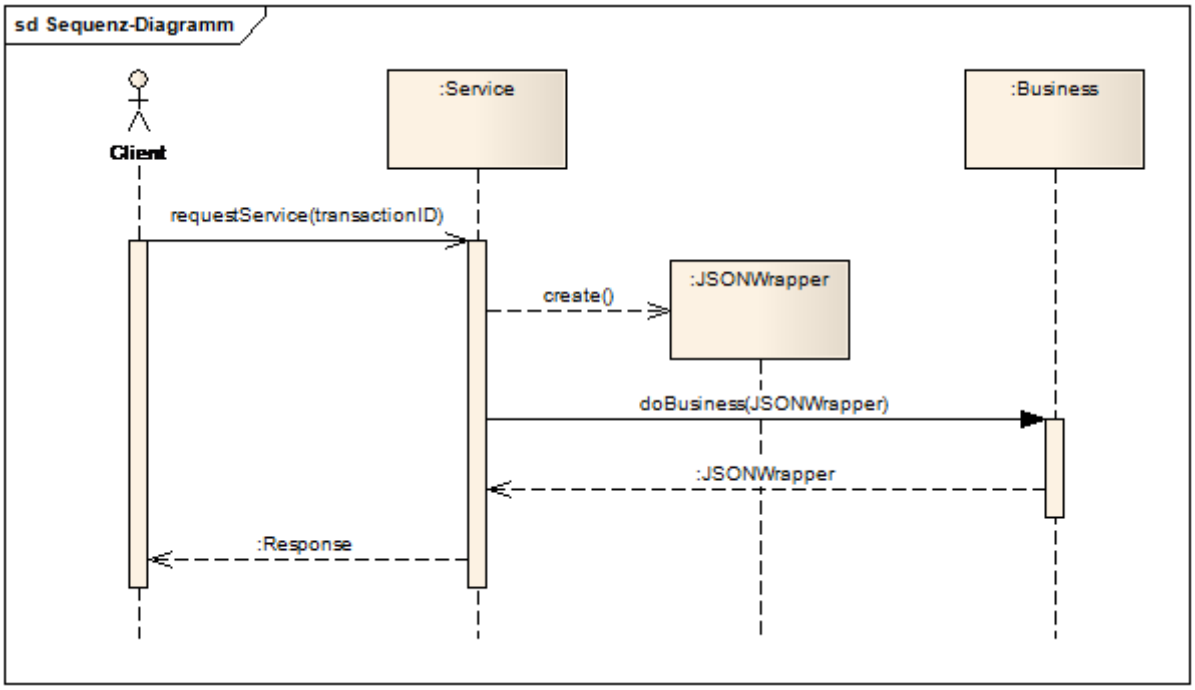

Abbildung 8-4: REST-Kommunikation

1. Eine REST-Service-Methode nimmt die Anfrage entgegen.
2. Die Transaktions-ID wird gelesen.
3. Die übergebenen Query-Parameter werden in ein JSON-Wrapper-Objekt verpackt.
4. Das Wrapper-Objekt wird einer Business-Methode übergeben und als Rückgabe dieser Business-Methode wird ebenfalls ein JSON-Wrapper Antwort Objekt erwartet.
5. Zu diesem Antwort Objekt wird die Transaktions-ID hinzugefügt.
6. Das Antwort Objekt wird in das entsprechende Ausgabeformat konvertiert.
7. Die Antwort wird abgeschickt.

Komponenten

ch.snapit.rs.service - REST

Das Service Package ist die REST-Service-Schicht. Darin enthalten sind alle Klassen die direkt von aussen (Web) angesprochen werden können. Jede REST-Anfrage wird in speziellen Klassen entgegen genommen und verarbeitet. Pro Ressource ist eine konkrete Klasse zuständig. Diese werden in einem späteren Architekturdokument detaillierter beschrieben. Diese Klassen delegieren dann die entsprechenden Aufgaben an die Business-Schicht, um dort weiter verarbeitet zu werden und zusätzliche eine entsprechende Antwort zu generieren. In der Service-Ebene wird die Anfrage entgegen genommen und die Antwort zurückgeschickt. Bevor dem Abschicken wird die vom Client übergebene Transaktions-ID wieder zur Antwort hinzugefügt. Falls ein Fehler auf der Serverseite auftritt oder die Anfrage unvollständig sein sollte wird trotzdem eine Antwort generiert, aber mit einem Property namens "status" welches dann denn Fehlerwert "1" zugewiesen wird. In Erfolgsfall wird dem Status immer den Wert "0" zugewiesen. Jeder andere Wert über "0" bedeutet einen Fehler.

ch.snapit.business - Business

In dieser Schicht wird die Anfrage auf ihre Vollständigkeit geprüft und entsprechend den gewünschten Operationen an weiter unterliegende Schichten gereicht.

Für die Kapselung der vielen Parameter wird ein JSON-Wrapper-Objekt erstellt und durch die Schichten gereicht. Dieses Objekt verhält sich sehr ähnlich zu einem Anything-Objekt. (siehe APF - Vorlesung - 04_05_Reflection.pdf)

8.2 Desing- und Technologieentscheide

Die folgende Tabelle zeigt den Einfluss der Architekturentscheide auf die nichtfunktionalen Anforderungen (siehe ISO 9126):

	Funktionalität	Zuverlässigkeit	Benutzbarkeit	Effizienz	Änderbarkeit	Übertragbarkeit
JSON				X	X	X
REST						X
Jersey		X			X	
Tomcat				X		X
SQLite				X		X
Google Service API - REST-Architektur		X		X		
PostgreSQL		x	x	x	x	
Hibernate		x	x	x	x	

JSON

Beschreibung

JSON ist ein kompaktes textbasiertes Datenaustauschformat.

- JSON Wikipedia

Einsatz

Sämtliche Daten die nicht in der URL Platz finden, werden im JSON-File übertragen. Wir nutzen das JSON-Format einerseits, um komplexe Anfragen und Antworten in einem Objekt zu kapseln und andererseits, um zwischen den Softwareschichten lange Parameterlisten zu vermeiden.

Alternativen

- XML
- ProtoBuf - Binäres Datenaustauschformat von Google

Vorteile

- Textbasiert und somit leserlich - Änderbarkeit
- Plattformunabhängig - Übertragbarkeit
- Schneller als XML (Parsen) - Effizienz
- Geringe Dateigrösse (schlanke Notation) - Effizienz

Nachteile

- Zur Übertragen von Binären-Daten braucht es BASE64-Encoding

REST

Beschreibung

REST ist eine zustandslose, HTTP-basierte Service-Architektur, die definiert wie Daten zwischen den beteiligten ausgetauscht werden.

- REST auf Wikipedia.org

Einsatz

Die gesamte Client-Server-Architektur von SnapIt beruht auf einer REST-Architektur

Alternativen

- SOAP - Webprotokoll zum Austauschen von Daten und für "Remote Procedure Calls" (XML als Datenaustauschformat)
- RMI - Javaspezifische Realisierung von "Remote Procedure Calls"
- Eigenes Protokoll

Vorteile

- Plattformunabhängigkeit (Anbindung von Fremdsystemen) - Übertragbarkeit
- Skalierbarkeit - Effizienz
- Unabhängiges Datenaustauschformat - Übertragbarkeit

Nachteile

- Fehlende Standardisierung

Jersey

Beschreibung

Jersey ist die Referenzimplementation um RESTful Webservices zu erstellen. Es setzt das Prinzip von HTTP-Ressourcen um und bietet dem Anwender die Möglichkeit einfache RESTful-Webservices zu erstellen.

- Jersey Home Page.

Einsatz

Jersey wird verwendet, um Nachrichten oder Anfragen von einem Client auf dem Server entgegen zu nehmen.

Alternativen

- Eigene Implementationen: Eigenes Servlet schreiben um REST-Anfragen entgegen zu nehmen.

Vorteile

- Standardisiert - Änderbarkeit
- Ausgereifte Technik - Zuverlässigkeit
- Schon oft erfolgreich in grossen Projekten eingesetzt - Zuverlässigkeit
- Relativ einfach verständlich - Änderbarkeit

Nachteile

- Zustandslos, d.h. es muss alles für eine erfolgreiche Transaktion in einem Request mitgeschickt werden.
- Die verschiedenen URIs müssen dynamisch generiert werden.

Tomcat

Beschreibung

Tomcat ist ein Open-Source Applikationsserver, der Java-Servlet und Java-Server-Pages interpretieren und ausführen kann.

- Tomcat

Einsatz

Tomcat ist der Container, in dem die SnapIt-Server Applikation läuft.

Alternativen

- Jetty - einfacherer Server als Tomcat. jetty
- JBoss - Komplexerer Applikationsserver

Vorteile

- Kostenlos
- Einfache Konfiguration - Änderbarkeit
- Läuft auch auf Linux-Plattformen - Übertragbarkeit

Nachteile

- Kaum skalierbar
- Aufwändige Konfiguration für Load-Balancing

SQLite

Beschreibung

SQLite ist ein DBMS für schmale Datenbankanwendungen. Im Gegensatz zu anderen DBMS hat SQLite einen kleineren Fuktionsumfang, ist jedoch performanter und kleiner

in der Datenmenge und somit effektiver für den Einsatz auf leistungsschwächeren Endgeräten. SQLite eignet sich nur, wenn kleine Datenmengen verwaltet werden.

- SQLite Home Page

Einsatz

SQLite wird auf dem Android-Smartphone eingesetzt um die Client-Server-Transaktionen zu verwalten.

Alternativen

db4objects (DBMS mit ORM) ist das einzig bekannte DBMS-System, dass als Alternative auf dem Android-Smartphone eingesetzt wird. Da wird das DBMS aber nur für wenige und einfache Transaktionen verwenden, lohnt es sich nicht (auch weil db4objects nicht so performant wie SQLite)

Vorteile

- Für Android optimierte API - Effizienz
- Performant - Effizienz
- Unterstütz alle von uns benötigten Funktionen - Effizienz
- Kein Android spezifisches DBMS - Übertragbarkeit

Nachteile

- Kein ORM

Google Service API - REST-Architektur

Beschreibung

Die "Google Service API - REST-Architektur" ist eine von drei auf der Google I/O 2010 besprochenen REST-Architekturen für Android Apps. Im Moment existiert noch keine konkrete opensource Implementierung, sondern nur die textuelle Beschreibung (siehe Präsentation). Die Archtitektur löst Smartphonetypische Probleme, wie Verbindungsunterbrüche, betriebssystemspezifische Eigenheiten und Performanceanforderungen.

- RESTful Android Apps

Einsatz

Die "Google Service API - REST-Architektur" wurde nicht eins zu eins implementiert. Es wurden vorallem Lösungsansätze übernommen.

Alternativen

- "Google ContentProvider API - REST-Architektur" (siehe Präsentation)
- "Google ContentProvider API with SyncAdapter - REST-Architektur" (siehe Präsentation)
- Eigene Ansätze möglich

Vorteile

- Empfehlung vom Hersteller - Zuverlässigkeit
- Stabil - Zuverlässigkeit
- Im Vergleich zu anderen Google-Varianten einfach zu implementieren - Änderbarkeit

Nachteile

- Komplexität

PostgreSQL:

Beschreibung

Weit verbreitete und stabile Datenbank.

Einsatz

Für die persistente Speicherung der Domain-Objekte

Alternativen

- MySQL
- Oracle DB

Vorteile

- Sehr weit verbreitete Datenbank - Benutzbarkeit
- Kostenlos - Änderbarkeit
- Wird ständig weiterentwickelt - Zuverlässigkeit
- Gute Performance - Effizienz

Nachteil

- Komplexe Konfigurierbarkeit
- Schwieriger als MySQL

Hibernate:

Beschreibung: OR-Mapper für Java

Einsatz: Für die einfache Verbindung zur Speicherung der Domain-Objekte in eine Datenbank.

Alternativen:

- JPA
- MyBatis

Vorteile

- Sehr weit verbreitete - Benutzbarkeit
- Kostenlos - Änderbarkeit
- Wird ständig weiterentwickelt - Zuverlässigkeit
- Relativ einfache Nutzung - Effizienz

Nachteil

- Möglichkeit zur komplexen Konfiguration
- Viele zusätzliche Libraries (jars) werden benötigt

8.3 SnapIt Architektur

Auf dem Grundstein, der REST-Architektur wurde die eigentliche Business-Anwendung entwickelt. Durch die REST-Architektur war die Schnittstelle zwischen Client und Server schon vorgegeben. Darauf aufbauen wurden die konkreten Webservices und die Benutzeroberfläche implementiert.

8.4 Client

8.4.1 Übersicht

Die Clientapplikation ist nach dem MVC-Prinzip aufgebaut. Das Model, sowie der Control & Data-Layer bestehen aus reinen Java-Klassen. Die View hingegen wurde zum grössten Teil deklarativ, als XML-Dateien, definiert.

Klassendiagram

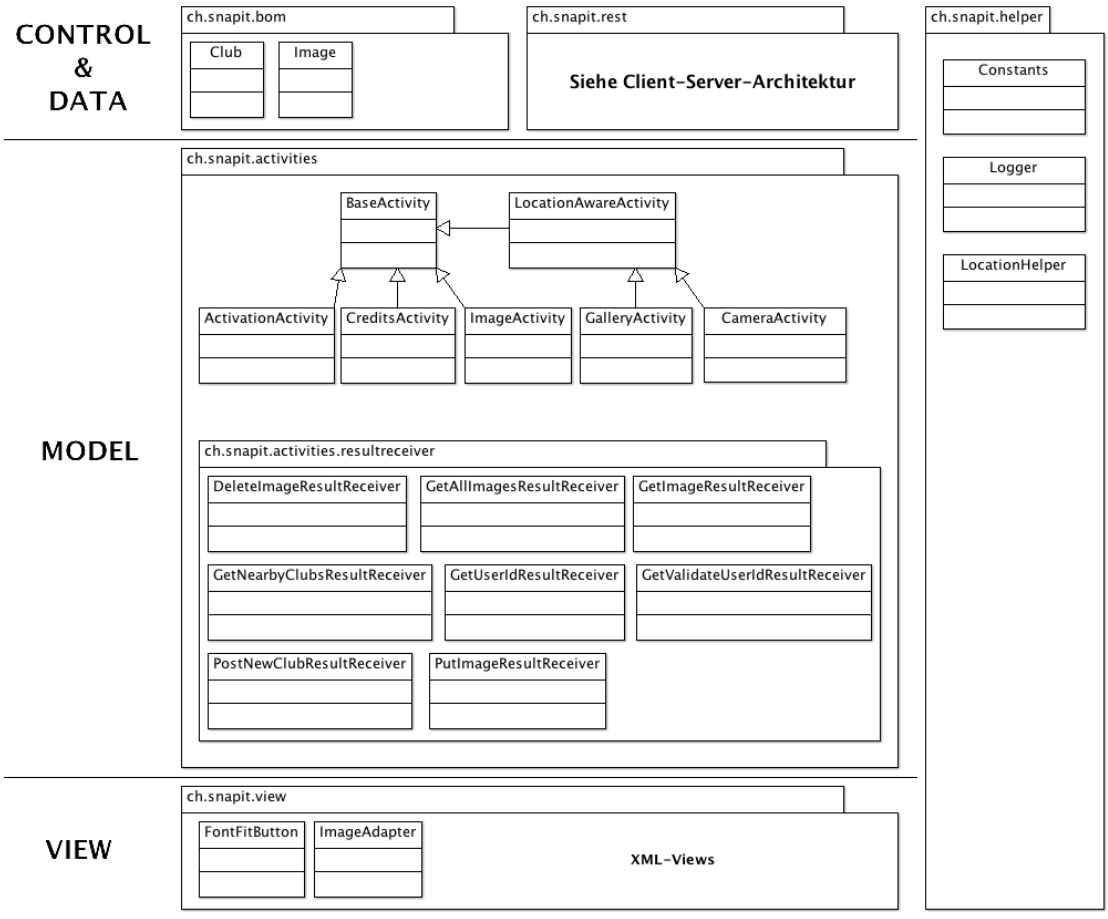

Abbildung 8-5: Klassendiagramm - Client

Control & Data

Der Control & Data-Layer wurde in der obigen Grafik als ein Layer dargestellt. Man kann diese Schichten aber auch als zwei einzelne Layers betrachten.

Das REST-Package wurde im Abschnitt „Architektur Client-Server-Kommunikation“ genau beschrieben.

Das bom-Package (Business Object Model) wurde ziemlich klein gehalten. Dies lässt sich folgendermassen begründen: Die Daten mit denen SnapIt arbeitet stammen einerseits vom Server und andererseits von der Benutzereingabe. Diese Daten werden zwischen Client und Server transportiert. Der Transport dieser Daten ist das Kerngeschäft von SnapIt. SnapIt hat also nicht viel Business-Logik. Der Aufwand die zu sendenden und empfangenen Daten in Objekte zu parsen ist in diesem Fall nicht gerechtfertigt. Dies würde nur Performance und somit auch Akkulaufzeit vergeuden. Anstelle die Daten aufwendig zu parsen, werden sie direkt dargestellt oder in den Datenaustauschformat JSON verpackt.

Model

Das Model besteht aus dem Package „activities“ und dem Subpackage „resultreceiver“.

Die Activites füllen die Views ab und kontrollieren sie, in dem sie Elemente dynamisch ein- und ausblenden, de-/aktivieren oder erzeugen. Sie agieren also als klassische Controller. Die Daten dazu beziehen sie über REST-Request die sie an den Server senden. Je nach Situation werden die Daten vom Server direkt angezeigt oder zuerst noch in Businessobjekte geparst.

Für jeden Request der an den Server geschickt wird, existiert ein ResultReceiver im Subpackage „resultreceiver“. Die Klassen in diesem Package sind Callback-Klassen, welche die asynchrone Antwort des Servers abfangen und verarbeiten. Die Verarbeitung geschieht dann wieder im Zusammenspiel mit der Activity, welche den Request abgesetzt hat.

View

Das View-Package erscheint in der Diagramm sehr leicht, da die XLM-Dateien, welche das GUI definieren nicht aufgelistet sind. Es sind nur zwei Klassen ersichtlich:

- Die Klasse FontFitButton definiert ein GUI-Element. Dieser Button hat die Eigenschaft, dass sich die Schriftgrösse des Labels der Textmenge anpasst. So wird das Label immer möglich gross dargestellt, ohne das Text abgeschnitten wird.
- ImageAdapter dient als GUI-Abstraktion der Business-Klasse Image. Hier wird das Mapping der Objekt-Felder mit der View definiert. Beim Darstellen mehrerer Fotos wird das Mapping bei jedem Objekt durchgeführt. Die Klasse erbt von einer Basisklasse aus dem Android-Framework.

Mehr zur Implementierung des GUIs ist im Abschnitt „Userinterface“ ersichtlich.

8.5 Server

8.5.1 Einführung

Die folgenden Kapitel befassen sich mit der Architektur des Servers. Zuerst werden Komponenten oder Technologien genannt, mit der die SnapIt-Server-Applikation

arbeitet. Danach werden einzelne Sonderheiten des Programms beschrieben und zuletzt gibt es Beschreibungen zu den einzelnen Packages und Klassen, dort wo es für das Verständnis hilfreich ist.

8.5.2 Referenzen

Die Applikation verwendet verschiedene externe Komponenten. Es ist deshalb sinnvoll sich in die folgenden Themen einzulesen.

- Hibernate: http://www.hibernate.org
- Tomcat Server: http://tomcat.apache.org
- Ant: http://ant.apache.org
- PostgreSQL: http://www.postgresql.org
- Apache log4j: http://logging.apache.org/log4j
- Google Gson: http://code.google.com/p/google-gson
- Jersey (REST): http://jersey.java.net

Für das Erstellen der Webseite wurde mit den folgenden Technologien gearbeitet. Da die Webseite relativ einfach gehalten wurde, wird diese nicht dokumentiert. Es wird auf den Quellcode verwiesen.

- CSS: http://de.wikipedia.org/wiki/Cascading_Style_Sheets
- JavaScript: http://de.selfhtml.org/javascript
- HTML: http://de.selfhtml.org
- jQuery: http://jquery.com

8.5.3 Information zum Stil der weiteren Kapitel

Die Dokumentation bezieht sich hauptsächlich auf die SnapIt-Server-Anwendung (SnapItServer-Projekt).

8.5.4 Software-Systemarchitektur

Systemübersicht

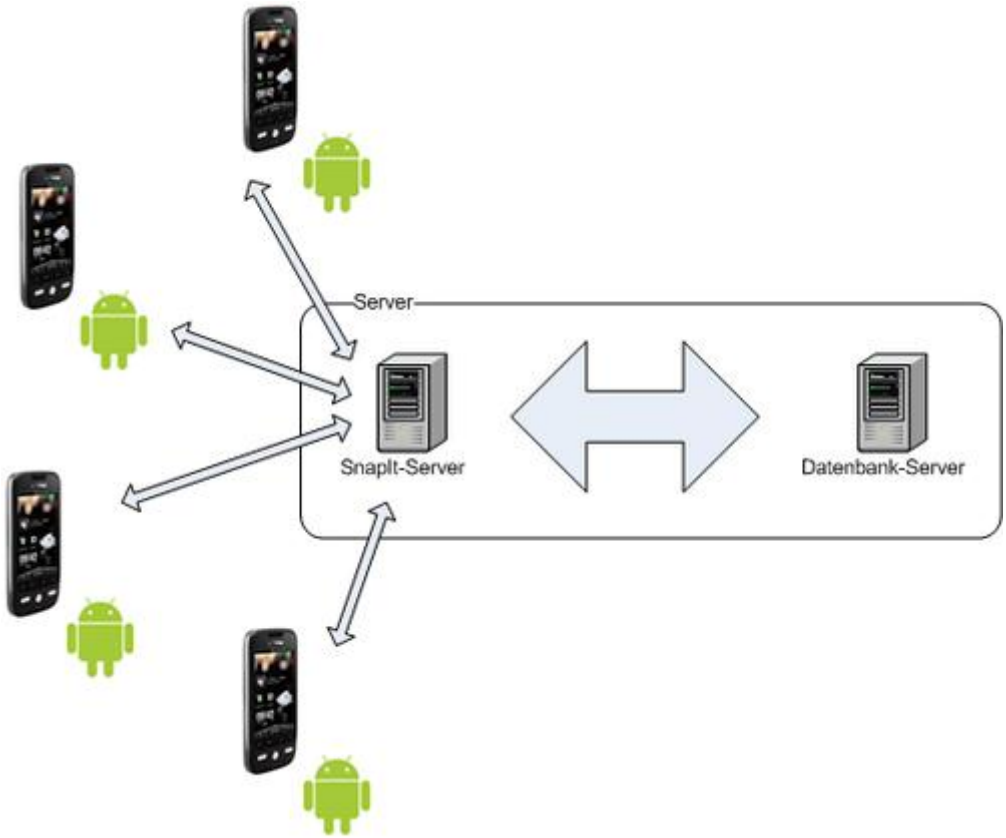

Abbildung 8-6: Systemübersicht

Architekturübersicht

SnapIt ist eine Client/Server-Anwendung. Beide Teile der Anwendung werden als separate Applikationen angesehen. Da es ein Zukunftsziel ist, die Client-Anwendung auf verschiedenen Handy-Plattformen zu portieren. Es besteht daher keine Kopplung zwischen den Projekten. Da aber für den Austausch von Nachrichten über die REST-Schnittstelle Schlüsselwörter definiert werden mussten, ist es möglich einzelne Klassen zu teilen (beispielsweise ch.snapit.util.JSONConstants). Jedoch wird darauf hingewiesen, dass auch ohne diese gemeinsam genutzte Klasse funktionieren würde. Weitere Details sind im entsprechenden Kapitel aufgeführt.

Systemstruktur

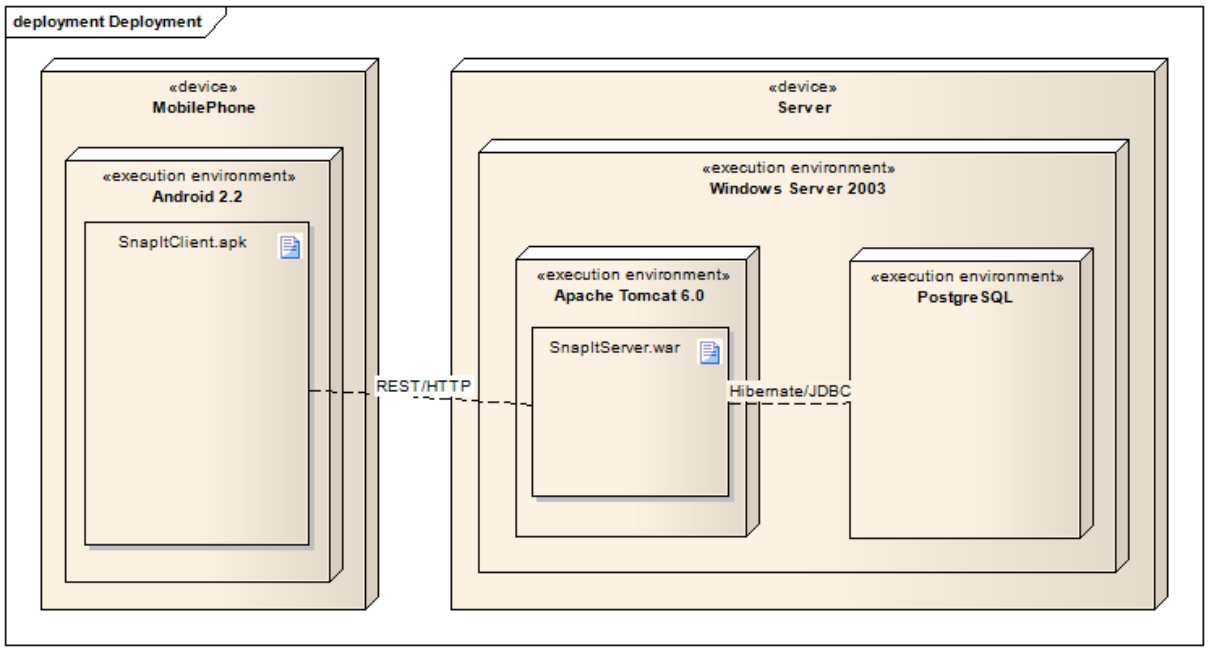

Abbildung 8-7: Deployment

Die Applikation SnapIt basiert auf einer 2-Tier Client-Server Architektur. Der Webserver sowie die dahinterliegende PostgreSQL-Datenbank sind auf demselben physikalischen Server abgelegt. Diese Architektur wurde gewählt, weil das Mobildevice lediglich zur Darstellung dient, während der Webservice die eigentliche Logik beinhaltet.

Implementationskonzepte

Bild-Upload

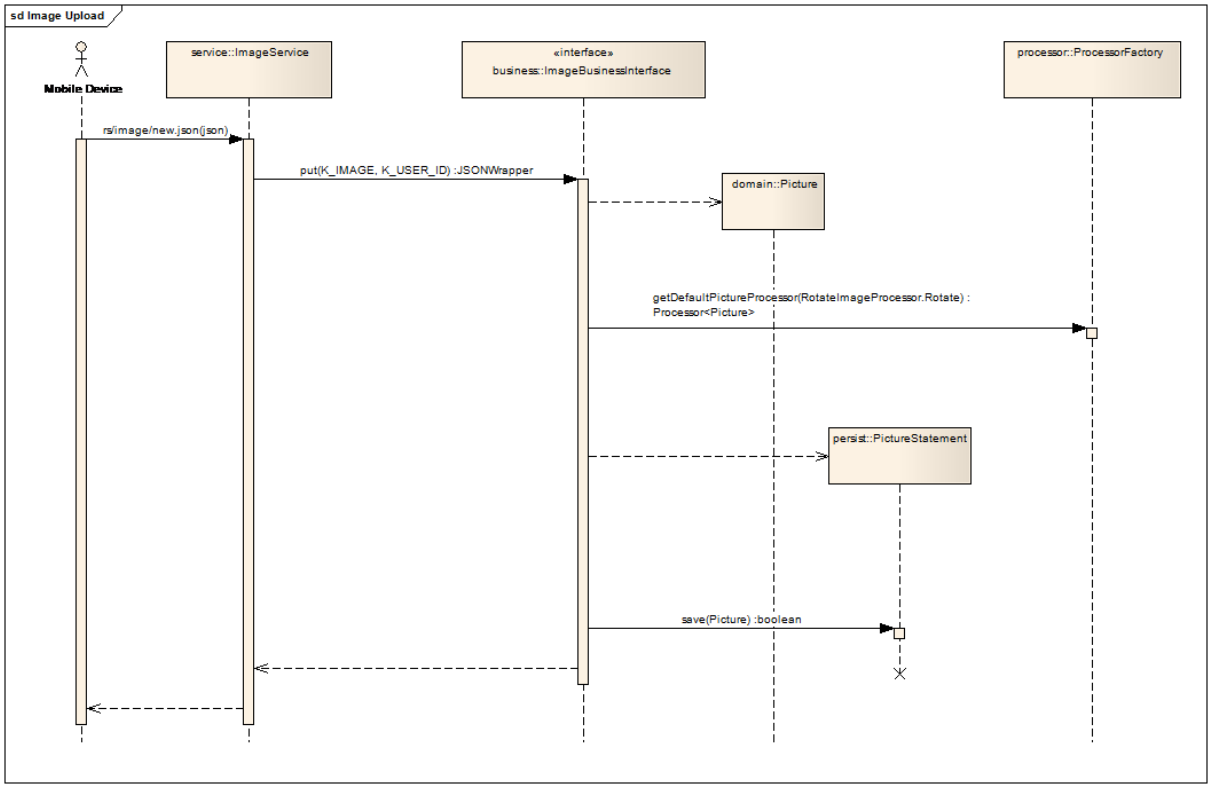

Abbildung 8-8: Sequenzdiagramm Bild-Upload

Beschreibung

Der Anwender ruft die mittels HTTP-PUT die Ressource „rs/image/new.json“. Als Inhalt dieser Anfrage muss das Bild als BASE64 enkodierter String im JSON-String unter dem Schlüssel K_IMAGE abgelegt werden. Zudem muss eine gültige Benutzer ID K_USER_ID mitgegeben werden.

Optional kann die Club ID (K_CLUB_ID) mitgesendet werden. Als Alternative kann der Name des Clubs mitgegeben werden unter K_CLUB_NAME.

Des Weiteren kann die genaue Position (K_LOCATION_LATITUDE und K_LOCATION_LONGITUDE) hinzugefügt werden.

Diese Eigenschaften werden dann vom Jersey-Servlet entgegengenommen und an die entsprechende Image-Service-Methode übergeben. Darin wird von der Business-Factory eine Instanz des ImageBusinessInterfaces erstellt und den die ganzen Parameter als JSONWrapper-Objekt übergeben.

Innerhalb der Business-Methode "put" wird ein Bild-Objekt (Picture) erstellt und dieses wird mit dem Standard-Bild-Prozessor bearbeitet.

Danach wird die Datenbank-Schicht mit dem PictureStatement-Objekt aufgerufen. Diesem wird das bearbeitete Bild übergeben und gespeichert.

Wenn das Bild erfolgreich gespeichert wurde, dann wird die Bild-ID mit dem erfolgreichen Status (K_STATUS), welcher den Wert 0 (Zahlenwert) hat zurückgeschickt.

Exception-Handling

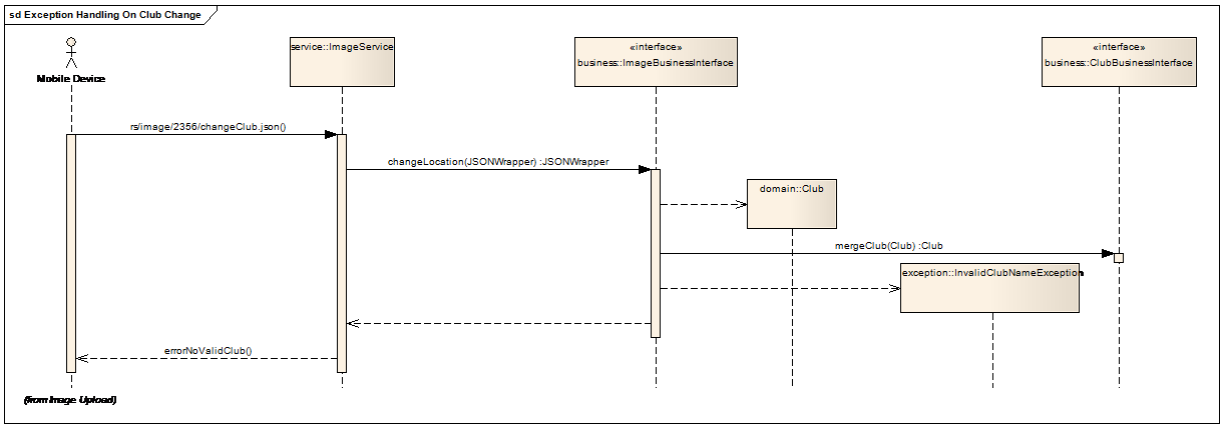

Abbildung 8-9: Exception-Handling

Beschreibung

Das Ziel der Ausnahmebehandlung war, dass der Benutzer möglichst eine Fehlermeldung sieht, und diese vom System versteckt werden. Somit ist es auch eine Art Sicherheitsmechanismus. Es werden an den Anwender nur relevante Fehler weitergeleitet.

Das hier beschriebene Beispiel dient als Referenz für alle Ausnahmen und kann somit auch auf diese angewendet werden.

Wenn ein Benutzer den Club eines Bildes wechseln möchte ruft er dazu z.B. die Ressource „image/2356/changeClub.json“ auf und übergibt den neuen Club, entweder mit K_CLUB_ID oder mit dem Namen K_CLUB_NAME.

Diese Anfrage wird dann durchgereicht bis zum behandelnden Business-Methode. Dort wird festgestellt, ob eine Club-ID oder ein gültiger Club Name mitgegeben wurde. Es wird eine InvalidClubNameException geworfen, welche eine SnapItException ist, sollte dies nicht der Fall sein. Weitere Details über die Ausnahmen werden in den nächsten Abschnitten erklärt.

Diese Exception wird danach in der „returnJSON“-Methode der ImageService- bzw. Service-Klasse abgefangen und entsprechend behandelt. In dieser Service-Klasse werden auch andere Ausnahmen, wie zum Beispiel HibernateExceptions abgefangen und auf die entsprechenden SnapItExceptions umgewandelt.

Da in diesem Beispiel, die Ausnahme an den Benutzer weitergeleitet werden soll, wird der K_STATUS auf V_STATUS_ERROR_INVALIDCLUBNAME (3) gesetzt und zurückgeschickt.

Wenn der Status (K_STATUS) grösser als eins ist, war der Request fehlerhaft.

Werte für K_STATUS:

- 0 -> alles OK
- 1 -> Allgemeiner Fehler in der Applikation
- > 1 -> Fehler, welcher durch fehlende Eingaben des Benutzers verursacht worden sind

Bildverarbeitung

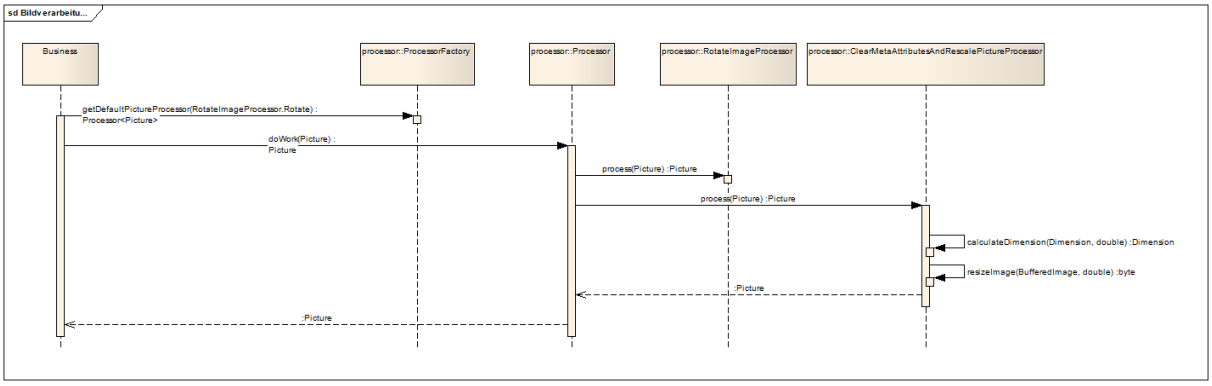

Abbildung 8-10: Bildverarbeitung

Beschreibung

Aus der Business-Methode wird ein gewünschter Prozessor geholt. In diesem Fall der Standard-Prozessor. Danach kann darauf die Methode „doWork“ aufgerufen werden, mit dem zu bearbeitenden Bild. Dieser führt dann alle Prozessoren auf diesem Bild aus und gibt das bearbeitete Bild wieder zurück.

Dieser komplexe Mechanismus wurde eingebaut, um unabhängig noch weitere Prozessoren einfach hinzufügen zu können.

Dafür wird das Prozessor-Interface implementiert und in der process-Methode die gewünschte Funktionalität implementiert.

Aufgrund der Factory können dann einfach verschiedene Prozessorabläufe definiert werden.

Datenbank-Problematik

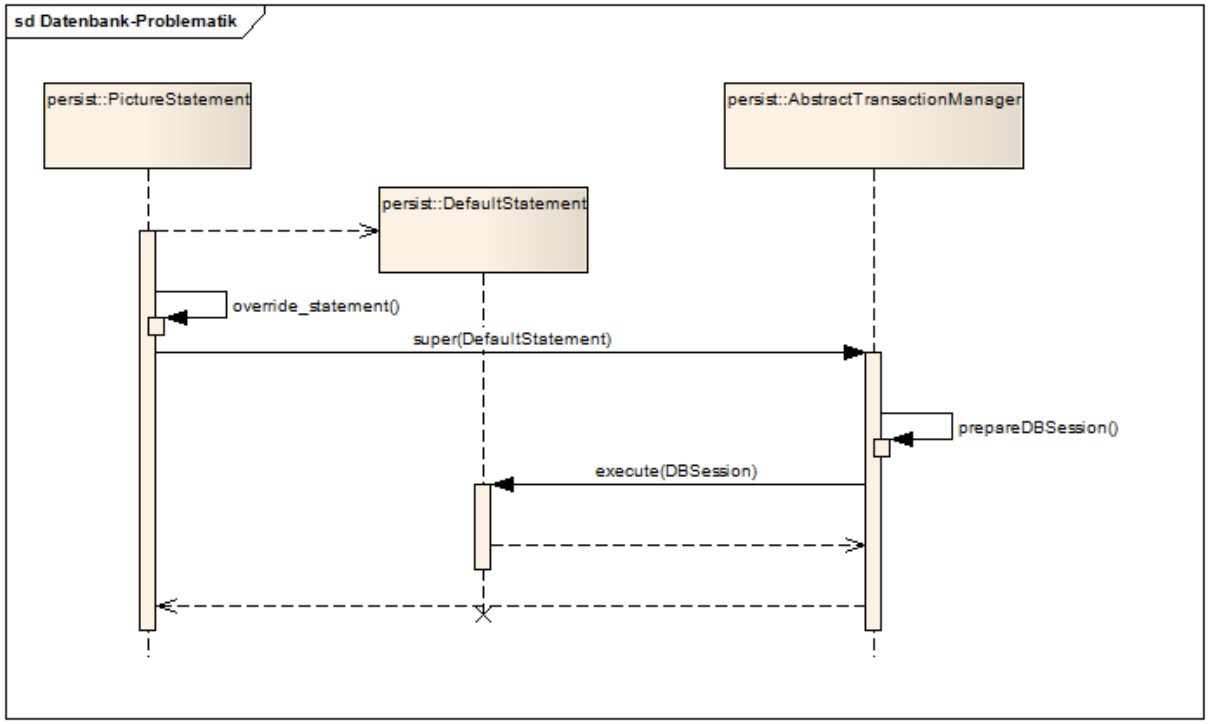

Abbildung 8-11: Datenbank-Problematik

Beschreibung

Für Datenbankabfragen muss jedes Mal eine Verbindung vom Datenbank-Verbindungs-Pool besorgt werden. Dann eine Transaktion gestartet werden. Erst danach kann die eigentliche Aktion ausgeführt werden. Nach der Abfrage sollte die Transaktion wieder geschlossen werden und an den Pool zurückgegeben werden. Um diesen Teil des Vorbereitens und Nachbearbeitens der Datenbankverbindung nicht bei jedem Statement nochmals zu schreiben wurde ein abstrakter Transactions-Manager erstellt.

Der Manager besitzt drei wichtige Methoden:

- execute - führt eine Anweisung aus und gibt zurück, ob diese Erfolgreich war
- get - holt ein Objekt aus der Datenbank
- getAll - holt eine Collection von Objekten aus der Datenbank

Wenn eine Abfrage abgesetzt werden soll, bei der nur etwas in die Datenbank gespeichert wird, dann nutzt man dazu die execute-Methode. Hierfür erweitert man die AbstractTransactionManager-Klasse und erzeugt ein neues DefaultStatement-Objekt und überschreibt die execute-Methode mit der gewünschten Save-Operation. Um den Verbindungsauf und Abbau kümmert sich die entsprechende execute-Methode der Superklasse. Man kann also nurnoch die spezifische Aktion ausprogrammieren und dieses DefaultStatement-Objekt der execute-Methode der AbstractTransactionManager-Klasse zur Ausführung übergeben.

Dieser Mechanismus ist für alle oben beschriebenen Methoden implementiert und funktioniert gleich. Somit sind alle möglichen Fälle von Datenbankabfragen und Aktionen abgedeckt.

Logging

Auf der Serverseite verwenden wir Log4J. Folgende Log-Level werden unterschieden: (Die Rangordnung ist auch ersichtlich)

- DEBUG < INFO < WARN < ERROR < FATAL
  - Je näher der Log-Level bei FATAL eingestellt ist, desto schneller laufen die Logging-Methoden.

Für den produktiven Einsatz sollte der Log-Lever für die Applikation SnapIt auf FATAL gestellt werden. Aufgrund der Anonymität der Nutzer und dem Performance-Vorteil.

8.5.5 Beschreibung der Packages

ch.snapit.business

Beschreibung

Das Package ist für eigentliche Verarbeitung der REST-Anfragen zuständig. Stellt zudem eine Factory zur Verfügung, damit verschiedene Implementierungen zulassen werden können, ohne die darüber liegenden Schichten zu ändern.

Klasse

BusinessFactory

- ClubBusinessInterface getDefaultClubBusinessInterface()
- ImageBusinessInterface getDefaultImageBusinessInterface()
- SecurityBusinessInterface getDefaultSecurityBusinessInterface()
- UserBusinessInterface getDefaultUserBusinessInterface()

Interfaces

ClubBusinessInterface:

- public abstract JSONWrapper getNearest(final JSONWrapper jsonRequest);
  - Gibt alle Clubs zurück, die sich in der Nähe befinden. Im Objekt jsonRequest müssen die Konstanten K_LOCATION_LATITUDE und K_LOCATION_LONGITUDE enthalten sein. Als Rückgabe werden die ermittelten Clubs als Array mit dem Schlüssel K_CLUBS in den JSONWrapper verpackt. Dabei werden die ClubDto-Objekte ins Array mittels Gson serialisiert.
- public abstract JSONWrapper getAll(JSONWrapper jsonRequest);
  - Gibt alle Clubs zurück. Falls noch zusätzlich im jsonRequest ein K_USER_ID angegeben wurde, werden nur alle Clubs zurückgegeben in denen der Benutzer mindestens ein Bild gekinpst hat. Die Rückgabe ist gleich aufgebaut wie bei der Methode getNearest.
- public abstract Club mergeClub(Club club);
  - Gibt einen gültigen Club oder null zurück. Als Parameter kann im Club-Objekt entweder die Id gesetzt sein oder der Name. Falls der Name schon

vorhanden ist, wird dieser Club zurückgegen. Wenn es sich um einen neuen Club handelt, wird dieser neu erfasst, sofern ein gültiger Name gesetzt ist.

ImageBusinessInterface:

- public abstract JSONWrapper delete(JSONWrapper jsonRequest);
  - Löscht das Bild mit der K_IMAGE_ID, sofern der Benutzer K_USER_ID der Eigentümer des Bildes ist.
- public abstract JSONWrapper get(JSONWrapper jsonRequest);
  - Gibt das Bild mit der ID K_IMAGE_ID zurück. Es können noch zusätzliche Parameter (K_IMAGE_SIZE) mitgegeben werden, die bestimmen in welcher Grösse das Bild zurückgeschickt werden soll.
- public abstract JSONWrapper getLast(JSONWrapper jsonRequest);
  - Gibt das zu letzt heraufgeladene Bild zurück. Ansonsten verhält sich der Aufruf gleich wie die Methode get().
- public abstract JSONWrapper getAll(JSONWrapper jsonRequest);
  - Gibt alle Bilder zurück, die auf die mitgegebenen Parameter passen. Weitere Erklärungen und Beispiele sind im Kapitel Protokoll zu finden.
- public JSONWrapper belongs(JSONWrapper jsonRequest);
  - Diese Methode prüft, ob der mitgegebene Benutzer (K_USER_ID) der Besitzer des Bildes (K_IMAGE_ID) ist. Die Antwort wird als Wahrheitswert unter dem Schlüssel K_IMAGE_BELONGS zurückgeschickt.
- public JSONWrapper changeLocation(JSONWrapper jsonRequest);

Ändert den Club eines Bildes K_IMAGE_ID, sofern der mitgegebene Benutzer K_USER_ID der Besitzer des Bildes ist. Falls eine gültige Club ID K_CLUB_ID mitgegeben wurde, wird das Bild diesem Club zugeordnet. Wenn nur ein K_CLUB_NAME mitgegeben wurde, dann wird geprüft, ob sich dieser Club mit dem System verbinden lässt. Eventuell wird auch ein neuer Club angelegt.

- public JSONWrapper put(JSONWrapper jsonRequest);
  - Fügt ein neues Bild hinzu. Das Bild ist als BASE64-Codierter String unter dem Schlüssel K_IMAGE zu finden, zudem muss ein gültiger Benutzer K_USER_ID mitgegeben werden.

SecurityBusinessInterface:

- public abstract User userAuthenticate(JSONWrapper json);
  - Diese Methode prüft, ob der Benutzer (K_USER_ID) existiert und gibt ihn im Erfolgsfall zurück. Falls es keinen solcher Benutzer gibt, wird eine NonExistingUserException geworfen.

UserBusinessInterface:

- public abstract JSONWrapper getNewID();
  - Generiert eine neue Benutzer ID und gibt diese zurück. Die neue ID ist unter dem Schlüssel K_USER_ID zu finden.
- public abstract JSONWrapper isUserIdValid(JSONWrapper jsonRequest);

  - Diese Methode prüft, ob die mitgegebene Benutzer ID K_USER_ID im System vorhanden und gültig ist.
- public abstract boolean existsUserId(String userId);
  - Prüft, ob eine Benutzer ID existiert. Diese Methode wird von der Methode isUserIdValid() genutzt. Zudem können mit Hilfe dieser Methode andere Klassen diese nutzen, um gewisse Operationen nur für glütige Benutzer zugänglich zu machen.

ch.snapit.config

Beschreibung

In diesem Package sind die Konfigurationsdateien für Hibernate und SnapIt abgelegt. Für die Hibernate-Konfigurationsdatei wird auf die Dokumentation von Hibernate verwiesen.

snapit.properties

- IMAGE_SIZE_LARGE=600
- IMAGE_SIZE_MIDDLE=480
- IMAGE_SIZE_LITTLE=150
- LOCATION_SEARCH_START=50
- LOCATION_SEARCH_END=100
- LOCATION_SEARCH_STEPSIZE=25
- USER_KEY_ALLOWED_CHARACTERS=0123456789abcdefghijklmnopqrstuvwxyz
- USER_KEY_COUNT_OF_TRY=100
- USER_KEY_SIZE=16

Erklärung:

Die Bilder werden nach dem Upload in drei Grössen in der Datenbank gespeichert. IMAGE_SIZE_XXX gibt an, wie gross die grössere Seite des Bildes in Pixeln sein darf. Falls das Bild nicht dieser entspricht, wird das Bild verkleinert. Wenn das Bild aber kleiner ist als die vorgegebene Grösse, dann behält das Bild seine ursprüngliche Grösse bei.

LOCATION_SEARCH_XXX wird gebraucht für die Bestimmung der am nächsten gelegenen Clubs. Zuerst wird eine Datenbankabfrage mit dem Wert LOCATION_SEARCH_START gemacht, falls innerhalb dieses Radius keine Clubs auffindbar sind, dann wird den Wert um LOCATION_SEARCH_STEPSIZE erhöht bis die obere Grenze, also LOCATION_SEARCH_END erreicht wird oder mindestens einen Club gefunden wird. Die Angaben werden in der Masseinheit Meter behandelt.

Alle Schlüssel mit dem Präfix USER_KEY_XXX werden für die Generierung der anonymen Benutzer ID gebraucht. USER_KEY_ALLOWED_CHARACTERS gibt an, welche Zeichen die ID enthalten kann. USER_KEY_SIZE bestimmt die Länge der ID und USER_KEY_COUNT_OF_TRY definiert, wie oft Versucht wird eine eindeutige ID zu generieren, bis eine SnapItException geworfen wird.

ch.snapit.domain

Beschreibung

Dieses Package enthält zwei Gruppen von Klassen. Nämlich Datenbank- und Datentransfer-Klassen.

Die Datenbank-Klassen werden hauptsächlich von Hibernate gebraucht. Die Datentranfer-Klassen sind für die Kommunikation über die REST-Schnittstelle.

Domain-Klassen:

- Club
- Location
- Picture
- User

Datentransfer-Klassen:

- ClubDto
- PictureDto

Diagramme

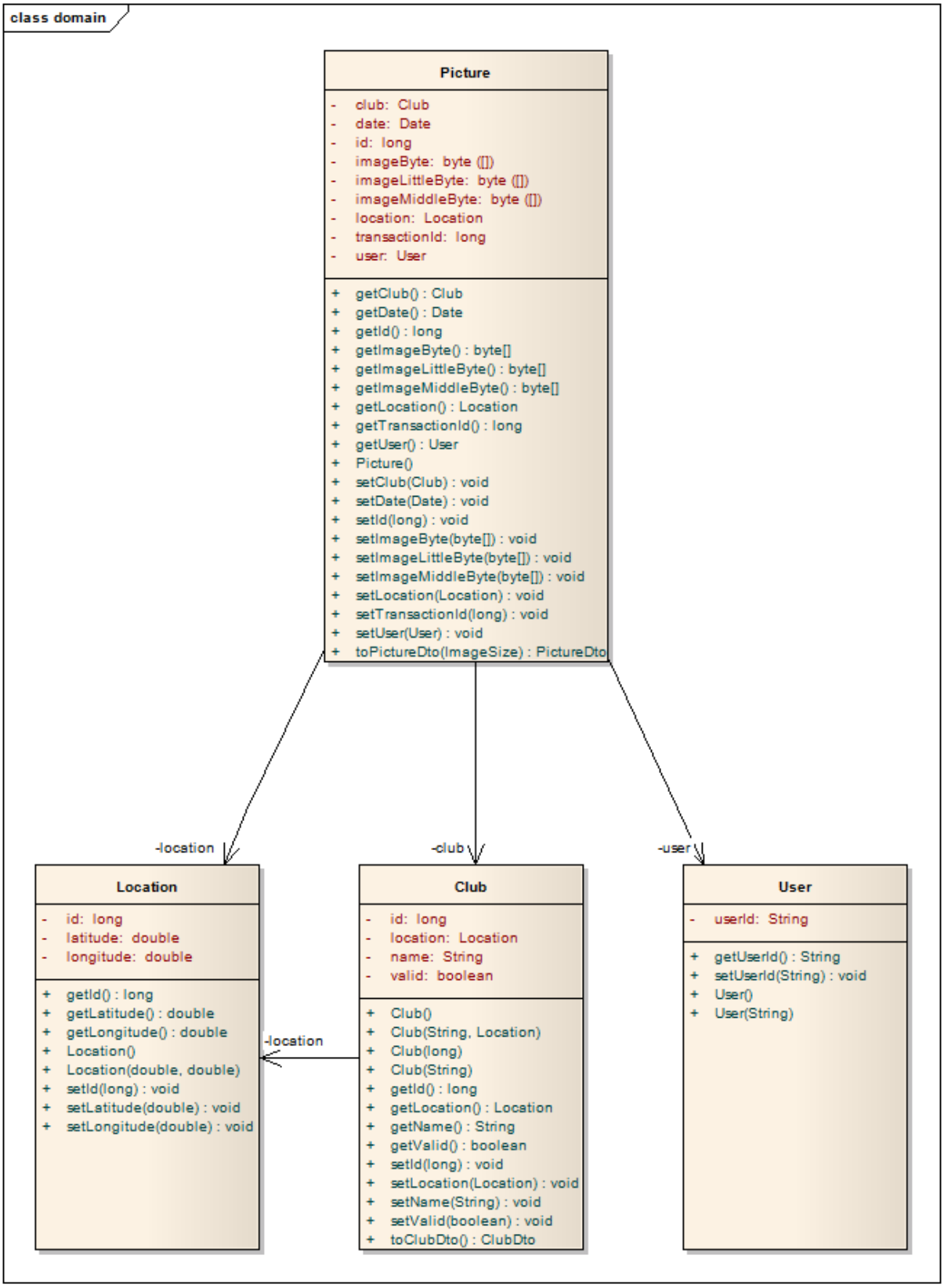

Abbildung 8-12: Domain-Klassen

Das obere Bild zeigt, alle Domain-Klassen, diese werden, wie weiter oben erwähnt hauptsächlich von Hibernate gebraucht. Einzig die Methode der Picture-Klasse

toPictureDto() wird verwendet, um daraus PictureDtos zu erstellen. Analog dazu verhält sich die entsprechende Methode in der Club-Klasse.

Abbildung 8-13: Datentransfer-Klassen

Das obere Bild zeigt, die zwei Datentransfer-Klassen. Diese fassen einzelne Domain-Klassen zusammen. Dies wurde gemacht, um eine Trennung von Business-Objekten und Transfer-Objekten zu haben und damit der Gson-Parser eindeutige Attributnamen, sowie eine flache, nicht verschachtelte, JSON-Objekte erstellen kann.

ch.snapit.exception

Beschreibung

Das Exception-Package beinhaltet alle SnapIt-Server-Exceptions. Die allgemeinste Ausnahme ist die SnapItException. Diese erweitert die Java-Runtime-Exception. Alle spezifischen Ausnahmen erweitern diese SnapItException. Für weitere Erklärung wird auf das entsprechende Kapitel unter Implementationskonzepte verwiesen.

Diagramm

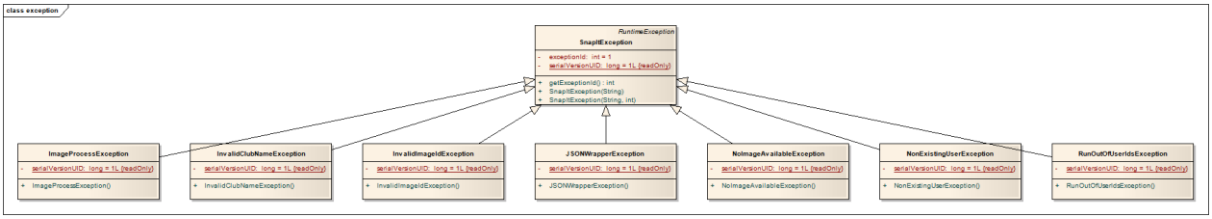

Abbildung 8-14: Exception-Klassen

Das obere Diagramm zeigt alle SnapIt-Exceptions.

ch.snapit.persist

Beschreibung

Das Package Persist beinhaltet alle Klassen, um auf die PostgreSQL-Datenbank zuzugreifen.

Diagramm

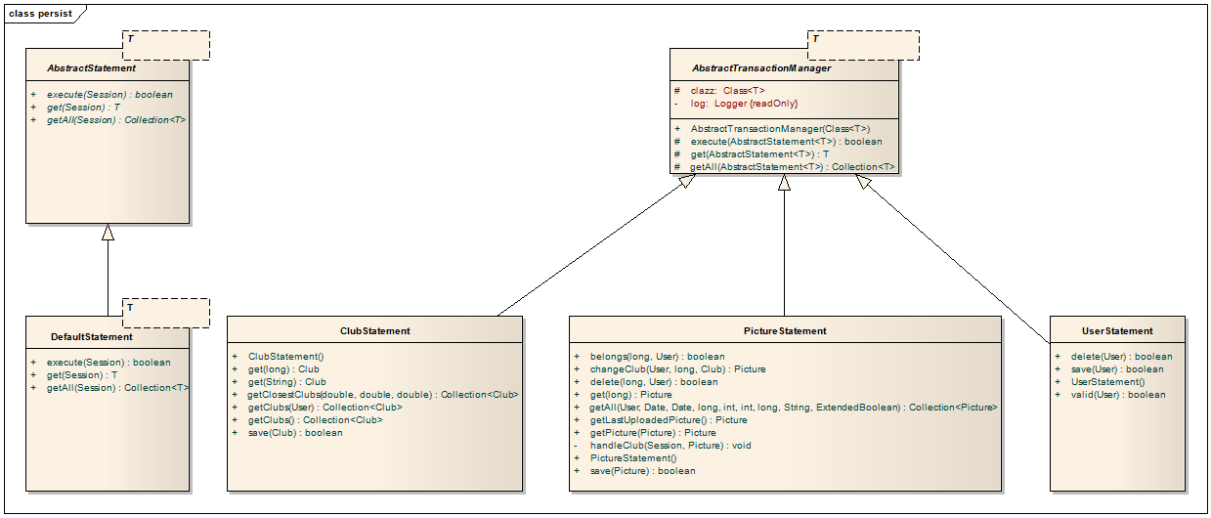

Abbildung 8-15: Persist-Klassen

Die Grafik zeigt alle Klassen im Persist-Package.

Klassen

- AbstractTransactionManager
  - Diese abstrakte Klasse wird genutzt, um den Verbindungsaufbau bzw. Abbau von Datenbanktransaktionen zu regeln. Es werden drei Methoden zur Verfügung gestellt, um von Kind-Klassen aufgerufen zu werden. Diese Kind-Klassen übergeben einfach die gewünschte überschriebene Methode der DefaultStatement-Klasse und die Eltern-Klasse führt diese in der gesicherten Umgebung aus (siehe Codeinjection).
- Für die spezifischen Klassen: ClubStatement, PictureStatement, UserStatement wird an dieser Stelle direkt auf den Quellcode verwiesen.

ch.snapit.processor

Beschreibung

Dieses Package dient vor allem, um Bilder zu bearbeiten. Bis jetzt beinhaltet es Prozessoren, also Code, um Bilder zu verkleinern, Meta-Informationen zu löschen und kleine Vorschaubilder zu generieren. Durch eine Factory kann ein bestimmter Satz bzw. List von Prozessoren erstellt werden, die Bilder bearbeiten können.

Diagramm

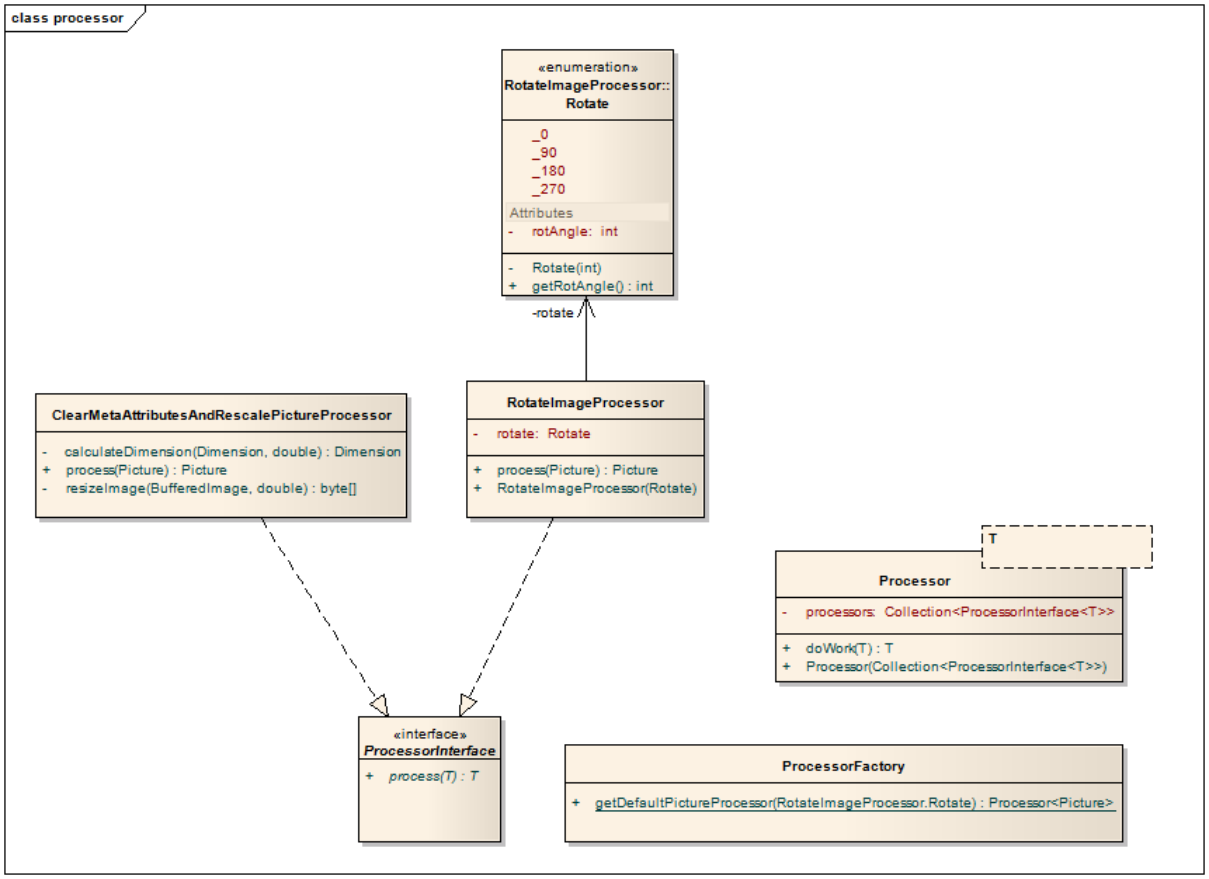

Abbildung 8-16: Prozessor-Klassen

Die obere Grafik zeigt den Inhalt des Processor-Packages.

Erklärung

Ein Klasse, welche das ProzessorInterface implmentiert nimmt mit der Methode process() ein bestimmtes Objekt vom Typ <T> entgegen, macht damit etwas und gibt es wieder zurück. Um mehrere dieser Prozessoren nacheinander auf das gleiche Objekt anzuwenden kann mittels der ProcessorFactory der gewünschte Prozessor erhalten werden. Wenn die Methode doWork mit dem zu bearbeitenden Objekt aufgerufen wird, dann werden alle gewünschten Prozessoren der Reihe nach aufgerufen mit dem übergebenen Objekt. Nachdem alle Prozessoren durchgelaufen sind wird das Objekt zurückgegeben und der Aufrufer kann damit weiterarbeiten.

Bisher sind nur zwei spezifische Processoren implementiert worden und zwar:

- ClearMetaAttributesAndRescalePictureProcessor (CMAARPP)
- RotateImageProcessor (RIP)

Der RIP dreht das Bild in die gewünschte Richtung.

Der CMAARPP beinhaltet aus Performance-Gründen gleich mehrere Aufgaben, zum einen verkleinert er das Bild auf die gewünschte Grösse, dabei werden gleich verschiedene Grössen erstellt. Zum anderen werden in diesem Schritt (Verkleinern) automatisch alle Meta-Informationen aus dem Bild gelöscht, da diese neu berechnet werden. Somit können gleich mehrere Aufgaben in einem Durchgang erledigt werden.

Weitere denkbare Prozessoren wären noch:

- Bild-Filter-Prozessoren
- Bild-Verbesserungs-Prozessoren
- usw.

Zudem kann noch gesagt werden, dass dieses Konstrukt sehr generisch gehalten wurde und es für andere Zwecke, wie z.B. Filtern von Anfragen, gebraucht werden könnte.

ch.snapit.service

Beschreibung

Das Package Service stellt die Verbindung zur "Aussenwelt" dar. Die abgeleiteten Klassen der Service-Klasse werden vom Jersey-Framework aufgerufen, um die gewünschten REST-Anfragen an die Applikation SnapIt zu übergeben.

Diagramm

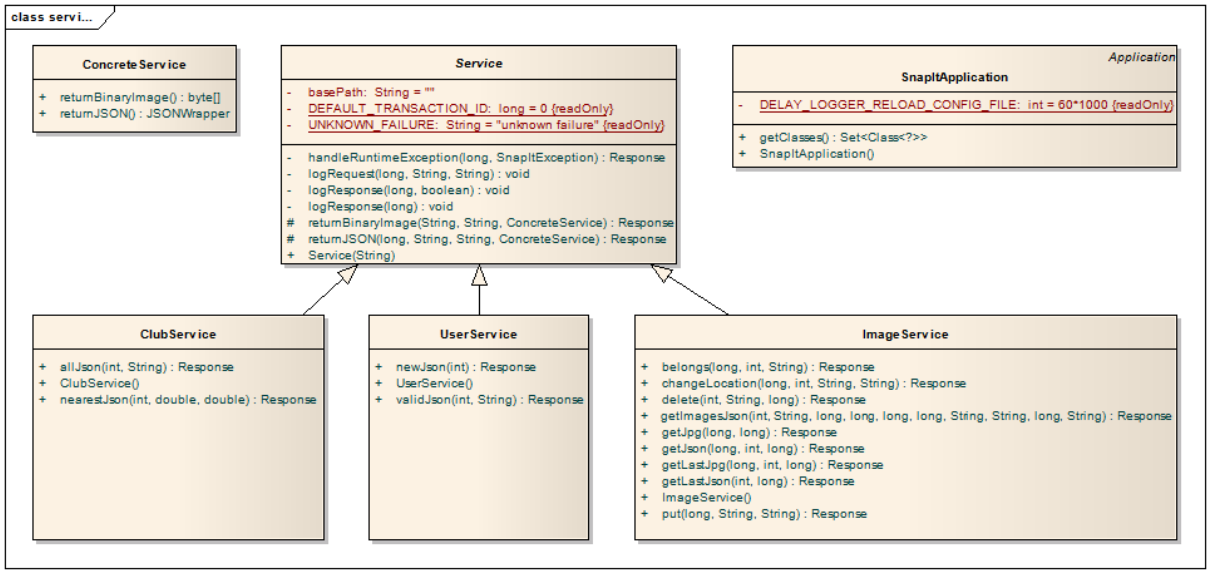

Abbildung 8-17: Service-Klassen

Das obere Bild zeigt die Service-Klassen der REST-Schnittstelle.

Erklärung

Die Klasse SnapItApplication stellt den Einstiegspunkt zur Applikation dar. Darin werden die Klassen, welche REST-Services anbieten dem Jersey-Framework mittels der Methode getClasses() bekannt gemacht. Das Logging-System log4j wird im Konstruktor dieser Klasse initialisiert.

Die Funktionsweise der Service-Klassen gleicht dem TransactionManager im Package ch.snapit.persist.

- ClubService
  - Stellt alle Services der Ressource "club" zur Verfügung.
- UserService
  - Stellt alle Services der Ressource "user" zur Verfügung.
- ImageService
  - Stellt alle Services der Ressource "image" zur Verfügung.

Der Ablauf wird im Kapitel REST Architektur genauer beschrieben. Für das Verständnis der einzelnen Services wird auf die Java-Doc und den Quellcode verwiesen.

ch.snapit.util

Beschreibung

Dieses Package enthält verschiedene Hilfsklassen, welche von den anderen Packages bzw. Klassen gebraucht werden.

Klasse und Erklärungen

- ExtendedBoolean
  - Wird verwendet, um einen Wahrheitswert mehrfach in der gleichen Klasse mittels Getter/Setter auszulesen bzw. zu setzen. Die normale Boolean-Klasse kann dort nicht verwendet werde, wenn mit dem Final-Schlüsselwort gearbeitet wird. Da dann nur einmal ein Wert im Konstruktor zugewiesen werden kann.
- HibernateUtil
  - Initialisiert Hibernate bzw. den Datenbank-Verbindungspool C3P0 und stellt Methoden, welche Datenbank-Sessions verwalten, zur Verfügung. Es ist als Singleton implementiert, da es nur einen Datenbank-Verbindungspool geben darf.
- ImageSize
  - Enthält die drei möglichen Bildgrössen.
- JSONWrapper
  - Stellt verschiedene Hilfsmethoden für die JSON-Verarbeitung bereit.
- LogClass
  - Enthält den Logger der Applikation.
- RotateImage
  - Ist für das Drehen von Bildern zuständig. Weitere Informationen sind auch im Processor-Package zu finden.
- SnapItConfig
  - Globale Konfigurationsdatei, welche einfachen Zugriff auf die Datei "snapit.properties" bietet.

Die Klasse **JSONConstants** enthält alle Konstanten sogenannte Schlüssel welche für den Meldungsaustausch über die REST-Schnittstellegebraucht werden. Dabei werden sogenannte Schlüssel K_XXX (K für Key) und Werte V_XXX (V für Value) definiert. Des Weiteren enthält sie auch Konstanten für die angebotenen Ressourcen R_XXX.

Die nachfolgende Grafik zeigt den Inhalt von JSONConstants.

Abbildung 8-18: JSONConstants

9 Testen

9.1 Client-Server-Architektur

Die Client-Server-Architektur wurde clientseitig anhand eines Testplans getestet:

Dieser Testplan testet nicht explizit den Upload eines Fotos, sondern den generelle Datenverkehr über die Restarchitektur. Er zeigt, dass die REST-Architektur fähig ist, die verschiedenen REST-Operationen auszuführen und alle Ausnahmesituationen behandeln kann. (Test durchgeführt am 09.11.10 und 18.12.10)

Testnr.	Testcase	Vorbedingungen	Soll	Status
1	Benutzer knippst mit der SnapIt-App ein Foto (PUT)	Unterbrechungsfreie Internetverbindung vorhanden	Das geknippste Bild wird direkt auf den SnapIt-Server geladen und auf der Webseite angezeigt.	ok
2	Der Benutzer löscht das zuletzt heraufgeladene Foto (DELETE)	Unterbrechungsfreie Internetverbindung vorhanden	Das Bild wird auf dem Server gelöscht und nicht mehr auf der Webseite dargestellt	ok
3	Der Benutzer gibt einen Namen ein undklickt auf den Button "Post Image-Name!" (POST)	Unterbrechungsfreie Internetverbindung vorhanden	Der Name des Bilds ändert sich auf dem Server dementsprechend	ok
4	Der Benutzer fordert das zuletzt geknipste Foto an (GET)	Unterbrechungsfreie Internetverbindung vorhanden	Auf dem Smartphone wird das zuletzt geknipste Foto und dessen Namen angezeigt	ok
5	Der Benutzer fordert wiederholt das zuletzt geknipste Foto an (GET)	Unterbrechungsfreie Internetverbindung vorhanden	Es wird nur dann ein neues Foto angefordert, wenn die zuletzt ausgeführte Transaktion komplett ist	ok

6	Der Benutzer klickt auf den Button "Show Transactions!"	Unterbrechungsfreie Internetverbindung vorhanden, mind. eine Transaktion am laufen	Es werden alle laufenden Transaktionen angezeigt	ok
7	Der Benutzer klickt auf den Button "Register Tr.!"	Unterbrechungsfreie Internetverbindung vorhanden	Die Transaktion wird angezeigt, wenn der Benutzer den Button "Transaktion anzeigen" anklickt	ok
8	Der Benutzer gibt die Transaction-Id ein und klickt auf den Button "Start Tr.!"	Unterbrechungsfreie Internetverbindung vorhanden, mind. eine registrierte Transaktion vorhanden	Die Transaktion wird erfolgreich ausgeführt (siehe Testfall 1)	ok
9	Der Benutzer gibt die Transaction-Id ein und klickt auf den Button "Delete Tr.!"	Unterbrechungsfreie Internetverbindung vorhanden, mind. eine registrierte Transaktion vorhanden	Die Transaktion wird gelöscht	ok
10	Der Benutzer führt eine POST-Transaktion aus	Keine Internetverbindung vorhanden	Es wird eine Meldung "KEINE INTERNETVERBINDUNG VORHANDEN!" angezeigt und die Transaktion wird gelöscht	ok

9.2 Lokalisierung

Für die Lokalisierungstests wurde eine fiktive Testlandschaft erstellt.

9.2.1 Ziel

Die virtuelle Partymeile HSR dient zum Testen der Lokationsbestimmung. So kann mit dem Android-Mobiltelefon unter echten Bedingungen getestet werden.

Hierzu wird jedem Gebäude einen virtuellen Club zugeordnet und dieser wird dann mit den richtigen Koordinaten in der Datenbank erfasst.

Für den produktiven Einsatz können die Daten in der Datenbank ausgetauscht werden.

9.2.2 Landkarte

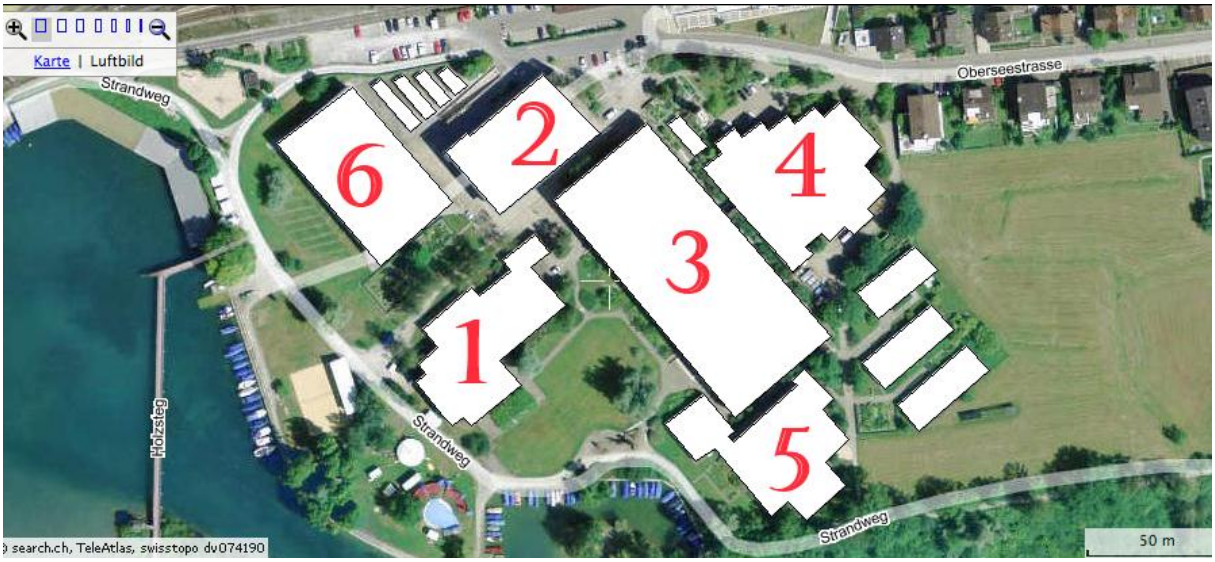

Abbildung 9-1: Virtuelle Partymeile

Diese Karte stellt die Grundlage für die virtuellen Clubs dar. Jedes Gebäude mit einer Nummer wird in der Datenbank erfasst. Die folgende Aufzählung bezieht sich auf die obige Grafik. Sie zeigt, welche Koordinaten in der Datenbank erfasst wurden.

Club	Latitude	Longitude
1. Mensa	47.22310433930531	8.816485404968262
2. Bibliothek	47.22379654652788	8.816716074943542
3. Hauptgebäude	47.22334114805647	8.81751000881195
4. Gebäude Praktikum	47.223628960345316	8.818100094795227
5. Physik	47.222707226871385	8.817896246910095
6. IFS	47.223636246712104	8.816066980361938
Istanbul Kebab	47.228016	8.819157
Nelson	47.22549387322478	8.816364705562592
Ponte Lumi	47.22761226765389	8.820691108703613
Restaurant Thai Orchid	47.228719700690405	8.819189071655273
Hotel Schwanen	47.22646109562935	8.814103603363037
Bonanno Bistro & Bar	47.228573987139136	8.821849822998047
La Corona	47.227466551058114	8.816442489624023
Mr. Hanno Stettler Bar	47.226446523671314	8.814382553100586
Hotel Speer	47.22584906994364	8.816807270050049
Corso Lounge	47.22646109562935	8.815562725067139
Restaurant Manolo's	47.22656309922326	8.817064762115479
Restaurant mundArtbeiz	47.22928797971284	8.824574947357178
Restaurant Sayori	47.22592193051487	8.816742897033691
Restaurant Falkenburg	47.227043970665086	8.816206455230713
Pfauenbier	47.227699697419055	8.826870918273926
Zak	47.23244983121199	8.83425235748291
Restaurant Weisser Rabe	47.233061780682114	8.836677074432373

Engelburg Schulhaus	47.442213	9.344264

Es wurden bewusst nicht alle weissen Flächen (Gebäuden) einen Club zugeordnet. Diese dienen später dazu, um noch nicht erfasste Clubs zu simulieren. Gedacht ist, dass ein Benutzer, welche in einem noch nicht erfassten Club Fotos knipst, diesen selber hinzufügen kann.

Da Lokalisierung aus technischer Sicht nicht sehr genau vorgenommen werden kann, wurde die Testumgebung im Verlauf des Projekts auf ganz Rapperswil-Jona erweitert (Clubs ohne Nummern). Somit ist es besser möglich, die Testresultate zu beurteilen.

Für Testzwecke wurde eine Lokation (Engelburg Schulhaus) ausserhalb der Testumgebung erfasst.

9.2.3 Testplan Lokalisierung

Allgemeine Informationen

Der Test findet auf der virtuellen Partymeile HSR statt. Es wird mit dem HTC Testgerät durchgeführt. Die Einstellungen auf dem Handy sind wie folgt zu konfigurieren:

- Location
  - Use wireless networks - checked
  - Use GPS satellites - checked
- Wireless & networks
  - Wi-Fi connected to HSR-Secure

Testdurchführung

Der komplette Testplan wurde erfolgreich durchgeführt (Durchgeührt am 13.11.2010 und am 02.12.2010).

Test 1: Normalfall

Ausgangslage	**Die Testperson befindet sich in der Nähe des Gebäude Nr. 6 (IFS) auf der Partymeile.**
Ablauf	1. SnapIt wird gestartet 2. IFS wird als momentanen Standort ausgewählt 3. Mittels 'PUT Image!' wird ein Foto aufgenommen 4. Das soeben geknipste Foto wird mit 'GET Image!' angezeigt.
Erwartetes Verhalten	1. Es erscheint eine Liste mit einigen Lokationen aus der virtuellen Partymeile. 2. Der Eintrag IFS sollte nicht an letzter Stelle stehen. 3. Das Foto wird erfolgreich heraufgeladen. Eine entsprechende Meldung erscheint. 4. Das Foto erscheint mit der richtigen Lokation.
Tatsächliches Verhalten	Das beobachtete Verhalten entsprach dem erwarteten.

Test 2: Keine Lokation ausgewählt

Ausgangslage	**Die Testperson befindet sich in der Nähe des Gebäude Nr. 1 (Mensa) auf der Partymeile.**
Ablauf	1. SnapIt wird gestartet 2. Das Auswahlfenster wird mit dem 'Back-Button' weggeklickt 3. Mittels 'PUT Image!' wird ein Foto aufgenommen 4. Das soeben geknipste Foto wird mit 'GET Image!' angezeigt.
Erwartetes Verhalten	1. Es erscheint eine Liste mit einigen Lokationen aus der virtuellen Partymeile. Dieses wird weggeklickt und die Club-ID ist auf 0 gesetzt 2. Der Eintrag Mensa sollte nicht an letzter Stelle stehen. 3. Das Foto wird erfolgreich heraufgeladen. Eine entsprechende Meldung erscheint. 4. Das Foto erscheint ohne Lokationsangabe.
Tatsächliches Verhalten	Das beobachtete Verhalten entsprach dem erwarteten.

Test 3: Kein GPS verfügbar

Ausgangslage	**Die Testperson befindet sich in der Nähe des Gebäude Nr. 5 (Physik) auf der Partymeile. Die Location-Einstellungen werden geändert nach:** • **Use GPS satellites - unchecked** **Danach wird das Handy für 30 Minuten ausgeschaltet und danach wieder eingeschaltet. Der Test beginnt danach.**
Ablauf	1. SnapIt wird gestartet 2. Es wird angezeigt, dass keine Locations verfügbar sind. 3. Mittels „PUT Image!“ wird ein Foto aufgenommen 4. Das soeben geknipste Foto wird mit „GET Image!“ angezeigt.
Erwartetes Verhalten	1. Es erscheint keine eine Liste mit einigen Lokationen aus der virtuellen Partymeile. 2. Es wird keine Auswahl angezeigt. 3. Das Foto wird erfolgreich heraufgeladen. Eine entsprechende Meldung erscheint. 4. Das Foto erscheint ohne Lokationsangabe.
Tatsächliches Verhalten	Das beobachtete Verhalten entsprach dem erwarteten.

Test 4: Nur mögliche Lokationen werden angezeigt

Ausgangslage	**Die Testperson befindet sich in der Nähe des Gebäude Nr. 3 (Hauptgebäude) auf der Partymeile.**
Ablauf	• SnapIt wird gestartet • Es wird eine Liste mit Lokationen aus der virtuellen Parymeile angezeigt.
Erwartetes	• Es erscheint eine Liste mit einigen Lokationen aus der

Verhalten	virtuellen Partymeile. • Die Liste enthält folgende Einträge nicht: ○ Kebab ○ Engelburg Schulhaus
Tatsächliches Verhalten	Das beobachtete Verhalten entsprach dem erwarteten.

9.3 Systemtest – SnapIt

9.3.1 Client

Der folgende Testplan deckt sämtliche definierten UseCases ab. Beim Durchführen der Tests wurde neben der Korrektheit der Testcases auch die Darstellung zu jedem Zeitpunkt kontrolliert. Die Testresultate werden anhand der Anzeige auf dem Smartphone sowie der Webseite validiert. Da die Webservices alle mit JUnit-Test getestet wurden, wird von einer fehlerfreien Webseite ausgegangen.

Alle Testfälle die keine Verbindung voraussetzen müssen einerseits bei fehlender Internetverbindung und andererseits bei fehlender Verbindung zum Server getestet werden. So wird überprüft, das die App bei Internetausfall und Serverausfall korrekt reagiert.

Aktivierung

Testnr.	Testcase	Vorbedingungen	Soll	Status
1.1	Benutzer öffnet die installierte Applikation	App ist neu installiert (oder alle App-Daten wurden gelöscht)	Aktivierungs-Ansicht wird angezeigt und korrekt dargestellt	ok
1.2	Benutzer öffnet die installierte Applikation	App wurde bereits aktiviert	Kamera-Ansicht wird angezeigt und korrekt dargestellt	ok
1.3	Benutzer klickt auf den Button "ID generieren"	Verbindung vorhanden	Ladefenster wird angezeigt, User-ID empfangen und zum Schluss die Kamera-Ansicht angezeigt; Die über die Menu-Funktion "Benutzer-ID anzeigen" anzuzeigende ID enspricht der Erwartung	ok

1.4	Benutzer klickt auf den Button "ID generieren" und bricht den Vorgang mit einem Klick auf den Back-Button ab	Verbindung vorhanden	Ladefenster wird angezeigt und verschwindet nach dem Klick auf den Back-Button; TC 1.3 kann nun erfolgreich ausgeführt werden	ok
1.5	Benutzer klickt auf den Button "ID eingeben"		Fenster mit Eingabefeld wird angezeigt	ok
1.6	Benutzer klickt auf den Back-Button	TC 1.5	Fenster schliesst sich; Beim erneuten Öffnen sind zuvor getätigte Eingabe nicht mehr ersichtlich	ok
1.7	Benutzer gibt vorhandene User-ID ein und klickt auf den Button "Benutzer-ID überprüfen"	TC 1.5, Verbindung vorhanden	User-Id wird validiert und Kamera-Ansicht angezeigt; Die über die Menu-Funktion "Benutzer-ID anzeigen" anzuzeigende ID enspricht der Erwartung	ok
1.8	Benutzer gibt nicht vorhandene User-ID ein und klickt auf den Button "Benutzer-ID überprüfen"	TC 1.5, Verbindung vorhanden	User-Id wird validiert und Fehlermeldung angezeigt	ok
1.9	Benutzer klickt auf den Button "ID generieren"	Keine Verbindung	Ladefenster wird angezeigt, "Keine Verbindung"-Meldung und "Nochmals versuchen"-Button erscheinen im Fenster	ok
1.1	Benutzer klickt auf den Button "Benutzer-ID überprüfen"	TC 1.5, Keine Verbindung	Ladefenster wird angezeigt, "Keine Verbindung"-Meldung und "Nochmals versuchen"-Button erscheinen im Fenster	ok

1.11	Benutzer klickt auf den Button "Nochmals versuchen"	TC 1.9, Verbindung vorhanden	Ladefenster wird angezeigt, User-ID empfangen und zum Schluss die Kamera-Ansicht angezeigt; Die über die Menu-Funktion "Benutzer-ID anzeigen" anzuzeigende ID enspricht der Erwartung	ok
1.1	Benutzer gibt eine vorhandene Benutzer-ID ein und klickt auf den Button "Nochmals versuchen"	TC 1.10, Verbindung vorhanden	User-Id wird validiert und Kamera-Ansicht angezeigt; Die über die Menu-Funktion "Benutzer-ID anzeigen" anzuzeigende ID enspricht der Erwartung	ok
1.13	Benutzer schwenkt das Smartphone		Die Ansicht passt sich nicht an	ok

Kamera

Testnr.	Testcase	Vorbedingungen	Soll	Status
2.1.1	Benutzer klickt auf das Lokalisieren-Icon oder den Clubnamen links oben	Verbindung vorhanden, Lokalisierung erfolgreich	Es werden alle Clubs in der Nähe angezeigt	ok
2.1.2	Benutzer gibt einen Namen ein	TC 2.1.1	Die Liste zeigt nur die Clubs an, die das eingegebene Wort enthalten	ok
2.1.3	Benutzer wählt einen Club aus	TC 2.1.1	Die Liste zeigt nur die Clubs an, die das eingegebene Wort enthalten	ok

2.1.4	Benutzer gibt einen Wert ein und klickt auf den Back-Button	TC 2.1.1	Das Fenster schliesst sich und der Wert in der Titelleiste ändert sich nicht	ok
2.1.5	Benutzer gibt einen Wert ein und klickt auf den Button "Lokation bestätigen"	TC 2.1.1	Das Fenster schliesst sich und der Wert in der Titelleiste ändert sich ensprechend der Eingabe	ok
2.1.6	Benutzer klickt auf das Lokalisieren-Icon oder den Clubnamen links oben	Keine Verbindung	"keine Verbindung"-Meldung und Eingabefeld werden angezeigt	ok
2.1.7	Benutzer klickt auf das Lokalisieren-Icon oder den Clubnamen links oben	Verbindung vorhanden, Lokalisierung gibt kein Resultat zurück	Eingabefeld wird angezeigt, keine Liste ersichtlich	ok
2.1.8	Benutzer klickt auf das Löschen/Schliessen-Icon rechts oben	Clubname in der Titelleiste ersichtlich	Clubname in der Titelleiste wird gelöscht	ok
2.2.1	Benutzer knippst ein Foto	Verbindung vorhanden, Benutzer hat einen Club ausgewählt (somit handelt es sich um einen existierenden Club)	Foto wird erfasst, heraufgeladen und dem korrekten Club zugeordnet (Club wird als validiert angezeigt); Es erscheint eine Mitteilung, welche die Anzahl heraufgeladenen und ausstehenden Fotos anzeigt	ok

2.2.2	Benutzer knippst ein Foto	Verbindung vorhanden, Benutzer hat einen neuen Club eingegeben	Foto wird erfasst, heraufgeladen und dem korrekten, neu erstellten Club zugeordnet (Club wird als nicht validiert angezeigt).	ok
2.2.3	Benutzer knippst ein Foto	Verbindung vorhanden, Benutzer hat keinen neuen Club eingegeben	Foto wird erfasst, heraufgeladen und keinem Club zugeordnet	ok
2.2.3	Benutzer knippst ein Foto	Keine Verbindung	Foto wird erfasst; Es erscheint eine Mitteilung, welche die Anzahl heraufgeladenen und ausstehenden Fotos anzeigt	ok
2.2.4	Benutzer öffnet die Mitteilungsliste und klickt auf die Mitteilung	TC 2.2.1	Galerie-Ansicht wird angezeigt, Mitteilung wird gelöscht	ok
2.2.5	Benutzer startet App erneut	TC 2.2.3, Verbindung vorhanden	Ausstehende Fotos werden heraufgeladen, Mitteilung wird angezeigt und zum Schluss Kamera-Ansicht dargestellt	ok
2.2.6	Benutzer knippst ein Foto (Kamera in Portrait-Position)	Verbindung vorhanden	Foto wird heraufgeladen und auf Webseite korrekt angezeigt (nicht verdreht)	ok
2.2.7	Benutzer knippst ein Foto (Kamera in verkehrter Portrait-Position)	Verbindung vorhanden	Foto wird heraufgeladen und auf Webseite korrekt angezeigt (nicht verdreht)	ok

2.2.8	Benutzer knippst ein Foto (Kamera in Landscape-Orientierung, nach links gekippt)	Verbindung vorhanden	Foto wird heraufgeladen und auf Webseite korrekt angezeigt (nicht verdreht)	ok
2.2.9	Benutzer knippst ein Foto (Kamera in Portrait-Position, nach rechts gekippt)	Verbindung vorhanden	Foto wird heraufgeladen und auf Webseite korrekt angezeigt (nicht verdreht)	ok
2.3.1	Benutzer öffnet das Option-Menü und klickt auf das "Benutzer-ID anzeigen"-Icon		Die korrekte Benutzer-ID wird im Format "xxxx-xxxx-xxxx-xxxx" angzeigt	ok
2.4.1	Benutzer öffnet das Option-Menü und klickt auf das "Galerie"-Icon	Verbindung vorhanden, es sind keine Fotos vom Benutzer vorhanden	Die Galerie mit einer "Keine Fotos"-Meldung wird angezeigt	ok
2.4.2	Benutzer öffnet das Option-Menü und klickt auf das "Galerie"-Icon	Verbindung vorhanden, es sind weniger als Fotos vom Benutzer vorhanden	Die Galerie wird angezeigt, alle Fotos werden nach Aufnahmedatum sortiert dargestellt, kein "Mehr Fotos"-Button ist am Ende der Liste ersichtlich	ok
2.4.3	Benutzer öffnet das Option-Menü und klickt auf das "Galerie"-Icon	Verbindung vorhanden, es sind mehr als 11 Fotos vom Benutzer vorhanden	Die Galerie wird angezeigt, alle Fotos werden nach Aufnahmedatum sortiert dargestellt, ein "Mehr Fotos"-Button ist am Ende der Liste ersichtlich	ok

2.4.4	Benutzer öffnet das Option-Menü und klickt auf das "Bilderübersicht"-Icon	Keine Verbindung	Die Galerie-Ansicht wird mit einer "keine Verbindung"-Meldung angezeigt	ok
2.5.1	Benutzer klickt auf den Back-Button		Android-Desktop wird angezeigt	ok

Galerie

Testnr.	Testcase	Vorbedingungen	Soll	Status
3.1.1	Benutzer klickt auf ein Foto	Verbindung vorhanden	Das Foto, der zugeordnete Club sowie der Aufnahmezeitpunkt werden angezeigt	ok
3.1.2	Benutzer klickt auf ein Foto	Keine Verbindung	Es wird eine "keine Verbindung"-Meldung in der Foto-Ansicht angezeigt	ok
3.1.3	Benutzer klickt auf den Backbutton		Die Kamera-Ansicht wird angezeigt	ok
3.2.1	Benutzer klickt lange auf ein Foto und wählt im Menu den Eintrag "Löschen"	Verbindung vorhanden, Foto vorhanden	Das Foto wird gelöscht	ok
3.2.2	Benutzer klickt lange auf ein Foto und wählt im Menu den Eintrag "Löschen"	Keine Verbindung, Foto vorhanden	Das Foto wird nicht gelöscht, es erscheint eine "keine Verbindung"-Meldung	ok

3.2.3	Benutzer klickt lange auf ein Foto und wählt im Menu den Eintrag "Relokalisieren"	Verbindung vorhanden, Foto vorhanden	Es erscheint eine Liste mit den nahegelegenden Clubs	ok
3.2.4	Benutzer wählt einen Club aus der Liste und klickt "Lokation bestätigen"	TC 2.6, Verbindung vorhanden, Foto mit nicht validiertem Club vorhanden	Das Icon des Fotos wechselt die Farbe auf grün und der korrekte Club wird beim anzeigen des Fotos dargestellt	ok
3.2.5	Benutzer gibt einen nicht vorhanden Club ein und klickt "Lokation bestätigen"	TC 2.6, Verbindung vorhanden, Foto mit validiertem Club vorhanden	Das Icon des Fotos wechselt die Farbe auf rot und der korrekte Club wird beim anzeigen des Fotos dargestellt	ok
3.2.6	Benutzer gibt einen Club ein und klickt "Lokation bestätigen"	TC 2.6, Keine Verbindung, Foto vorhanden	Es erscheint eine "keine Verbindung"-Meldung	ok
3.3.1	Benutzer klickt auf die Menü-Taste und klickt auf das Icon "Kamera"		Die Kamera-Ansicht wird angezeigt	ok
3.3.2	Benutzer klickt auf die Menü-Taste und klickt auf das Icon "Credits"		Die Credits-Ansicht wird angezeigt	ok
3.3.3	Benutzer klickt auf die Menü-Taste und klickt auf das Icon "Aktualisieren"	Verbindung vorhanden	Die Fotos werden aktualisiert, neu heraufgeladene Bilder werden angezeigt	ok
3.3.4	Benutzer klickt auf die Menü-Taste und klickt auf das Icon „Aktualisieren“	Keine Verbindung	Es erscheint eine „keine Verbindung“-Meldung	ok

Foto

Testnr.	Testcase	Vorbedingungen	Soll	Status
4	Benutzer klickt auf den Back-Button	Verbindung vorhanden	Die Galerie wird angezeigt	ok

Credits

Testnr.	Testcase	Vorbedingungen	Soll	Status
6	Benutzer klickt auf den Back-Button	Verbindung vorhanden	Die Galerie wird angezeigt	ok

9.3.2 Server

Um die Stabilität der Serverapplikation während der Entwicklung zu gewährleisten wurden drei Arten von JUnit-Test erstellt, welche in den nächsten Abschnitten beschrieben werden.

Die Test wurden auf dem lokalen Rechner durchgeführt.

Datenbank-Tests

Der Datenbanktest testet, ob Domain-Objekte in der Datenbank gespeichert, gelöscht und verändert werden können.

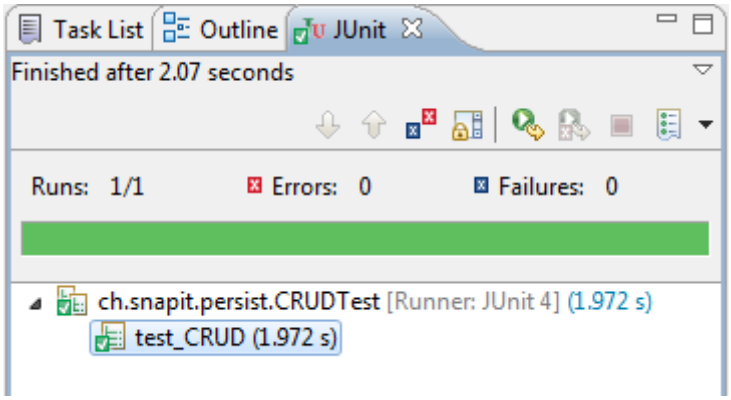

Abbildung 9-2: JUnit-Test Datenbank CRUD

Automatische Systemtests

Die automatisierten Systemtests testen das komplette Serversystem. Vom Service-Layer bis zur Datenbank. Diese Tests sind keine JUnit-Tests im gewohnten Sinn, denn sie testen das ganze System und benötigen daher auch eine gewisse Ausführungszeit.

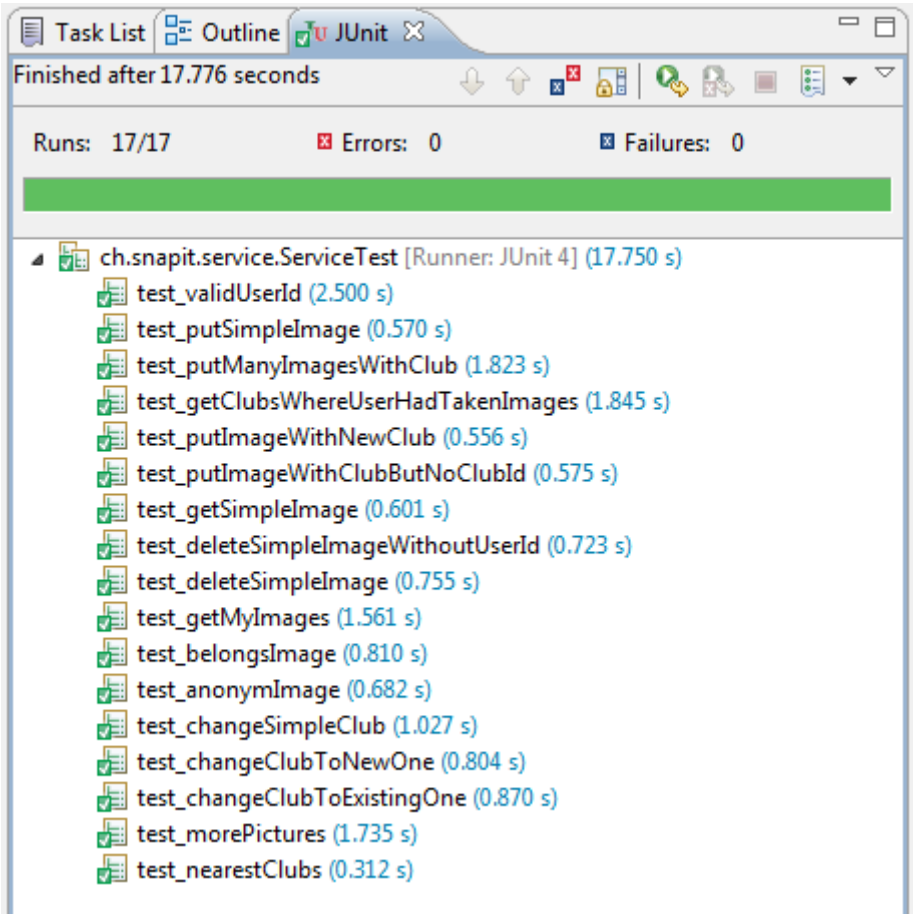

Abbildung 9-3: JUnit-Tests – automatische Systemtests

Stress-Test

Der Stress-Test zeigt die Belastbarkeit des Systems. Wir können so die nichtfunktionale Anforderung sehr leicht testen und es bietet einen gewissen Anhaltspunkt, wie viel Benutzer das System ohne Probleme aushalten kann.

Dieser Stress-Test wurde auf einem Entwicklerrechner ausgeführt.

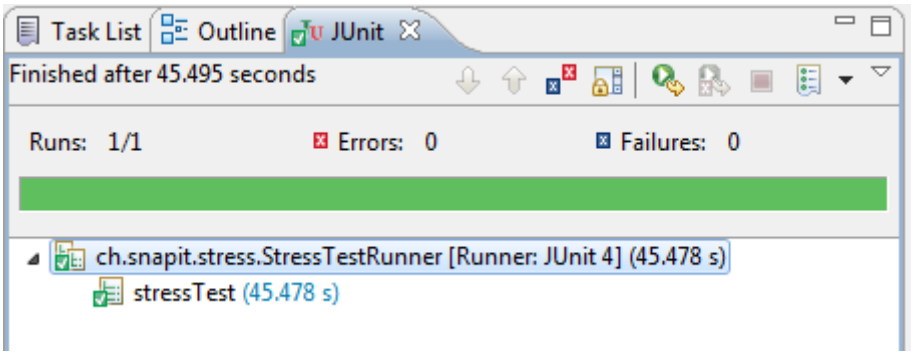

Abbildung 9-4: JUnit-Test – Belastbarkeitstest

Es wurde der parallele Zugriff von 50 verschiedenen Benutzern simuliert, welche gleichzeitig versuchen je zwei Bilder hochzuladen. Danach werden zufällig vier verschiedene Bilder heruntergeladen und die zuvor heraufgeladenen Bilder wieder gelöscht. Daraus ergeben sich 400 (50 Benutzer * 8 Anfragen) Anfragen. Da der Test innerhalb von 45.5 Sekunden durchgelaufen ist, sind das 8.79 Anfragen pro Sekunde (400 Anfragen / 45.5 Sekunden). Somit sind die Anforderungen sehr gut erfüllt.

9.4 Usability

Aus Zeit gründen und da der Fokus unserer App nicht auf dem GUI lag wurde kein heuristisches Verfahren verwendet um die Usability zu prüfen. Die Usability-Tests wurden durchgeführt, in dem verschiedene Benutzer die App für mehrere Tage verwendeten und uns ein mündliches Feedback gaben. Die daraus resultierenden Änderungen sind im Abschnitt Screens unter dem Titel Userinterface zu finden.

9.5 Codequalität

9.5.1 SnapIt-Server

Die Codequalität ist einerseits mit dem Eclipse-Plugin PMD geprüft worden und ist direkt im SnapItServer-Projekt des SVN-Repositories abgespeichert. Zu finden ist der Bericht im Ordner „reports“.

Des Weiteren wurden Screenshots erstellt von Auswertungen mit dem Eclipse Metric Plugin. Zur Vollständigkeit sind alle berechneten Werte auf den folgenden Abbildungen einsehbar.

Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
Number of Overridden Methods (avg/max per type)	3	0.062	0.242	1	/SnapItServer/src/ch/snapit/service/SnapItApplicatio...	
src	2	0.048	0.213	1	/SnapItServer/src/ch/snapit/service/SnapItApplicatio...	
test	1	0.167	0.373	1	/SnapItServer/test/ch/snapit/stress/Client.java	
Number of Attributes (avg/max per type)	55	1.146	2.227	10	/SnapItServer/src/ch/snapit/domain/PictureDto.java	
src	40	0.952	2.203	10	/SnapItServer/src/ch/snapit/domain/PictureDto.java	
test	15	2.5	1.893	6	/SnapItServer/test/ch/snapit/stress/Client.java	
Number of Children (avg/max per type)	44	0.917	3.181	17	/SnapItServer/src/ch/snapit/persist/DefaultStatement...	
src	44	1.048	3.38	17	/SnapItServer/src/ch/snapit/persist/DefaultStatement...	
test	0	0	0	0	/SnapItServer/test/ch/snapit/persist/CRUDTest.java	
Number of Classes (avg/max per packageFragment)	48	4.364	2.385	8	/SnapItServer/src/ch/snapit/exception	
src	42	6	1.195	8	/SnapItServer/src/ch/snapit/exception	
test	6	1.5	0.5	2	/SnapItServer/test/ch/snapit/service	
Method Lines of Code (avg/max per method)	1361	5.533	6.793	38	/SnapItServer/src/ch/snapit/persist/PictureStatement....	getAll
src	1004	4.758	6.279	38	/SnapItServer/src/ch/snapit/persist/PictureStatement....	getAll
test	357	10.2	7.826	38	/SnapItServer/test/ch/snapit/service/TestSuite.java	prepareDbForTesting
Number of Methods (avg/max per type)	231	4.812	5.692	23	/SnapItServer/src/ch/snapit/util/JSONWrapper.java	
src	199	4.738	5.581	23	/SnapItServer/src/ch/snapit/util/JSONWrapper.java	
test	32	5.333	6.394	19	/SnapItServer/test/ch/snapit/service/ServiceTest.java	
Nested Block Depth (avg/max per method)		1.374	0.643	4	/SnapItServer/src/ch/snapit/business/UserBusiness.java	getNewID
src		1.389	0.654	4	/SnapItServer/src/ch/snapit/business/UserBusiness.java	getNewID
test		1.286	0.564	3	/SnapItServer/test/ch/snapit/service/ServiceTest.java	test_putManyImagesWit...
Depth of Inheritance Tree (avg/max per type)		1.833	1.419	5	/SnapItServer/src/ch/snapit/exception/ImageProcess...	
src		1.929	1.486	5	/SnapItServer/src/ch/snapit/exception/ImageProcess...	
test		1.167	0.373	2	/SnapItServer/test/ch/snapit/stress/Client.java	
Number of Packages	11					
src	7					
test	4					

Abbildung 9-5: Codestatistik - Teil 1 – Metrics

Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
Afferent Coupling (avg/max per packageFragment)		5.909	8.754	30	/SnapItServer/src/ch/snapit/util	
src		8.857	9.804	30	/SnapItServer/src/ch/snapit/util	
test		0.75	0.829	2	/SnapItServer/test/ch/snapit/util	
Number of Interfaces (avg/max per packageFragment)	5	0.455	1.157	4	/SnapItServer/src/ch/snapit/business	
src	5	0.714	1.385	4	/SnapItServer/src/ch/snapit/business	
test	0	0	0	0	/SnapItServer/test/ch/snapit/persist	
McCabe Cyclomatic Complexity (avg/max per method)		1.569	1.214	9	/SnapItServer/src/ch/snapit/persist/PictureStatement....	getAll
src		1.588	1.253	9	/SnapItServer/src/ch/snapit/persist/PictureStatement....	getAll
test		1.457	0.936	4	/SnapItServer/test/ch/snapit/service/TestSuite.java	prepareDbForTesting
Total Lines of Code	2541					
src	2010					
test	531					
Instability (avg/max per packageFragment)		0.586	0.347	1	/SnapItServer/src/ch/snapit/service	
src		0.539	0.297	1	/SnapItServer/src/ch/snapit/service	
test		0.667	0.408	1	/SnapItServer/test/ch/snapit/persist	
Number of Parameters (avg/max per method)		0.858	1.151	10	/SnapItServer/src/ch/snapit/service/ImageService.java	getImagesJson
src		0.967	1.194	10	/SnapItServer/src/ch/snapit/service/ImageService.java	getImagesJson
test		0.2	0.466	2	/SnapItServer/test/ch/snapit/stress/Client.java	Client
Lack of Cohesion of Methods (avg/max per type)		0.127	0.283	0.94	/SnapItServer/src/ch/snapit/domain/PictureDto.java	
src		0.111	0.279	0.94	/SnapItServer/src/ch/snapit/domain/PictureDto.java	
test		0.238	0.282	0.667	/SnapItServer/test/ch/snapit/service/TestSuite.java	
Efferent Coupling (avg/max per packageFragment)		3.818	2.622	8	/SnapItServer/src/ch/snapit/business	
src		5.286	2.119	8	/SnapItServer/src/ch/snapit/business	
test		1.25	0.829	2	/SnapItServer/test/ch/snapit/service	
Number of Static Methods (avg/max per type)	15	0.312	0.87	4	/SnapItServer/src/ch/snapit/business/BusinessFactor...	
src	12	0.286	0.881	4	/SnapItServer/src/ch/snapit/business/BusinessFactor...	
test	3	0.5	0.764	2	/SnapItServer/test/ch/snapit/service/TestSuite.java	
Normalized Distance (avg/max per packageFragment)		0.364	0.366	1	/SnapItServer/test/ch/snapit/util	
test		0.333	0.408	1	/SnapItServer/test/ch/snapit/util	
src		0.381	0.339	0.909	/SnapItServer/src/ch/snapit/util	

Abbildung 9-6: Codestatistik - Teil 2 – Metrics

Wie McCabe vorschlägt, ist die zyklomatische Zahl nicht höher als 10. Somit ist der Code einfach und verständlich.

Auffällig ist der hohe Wert der Anzahl von Parametern einer einzigen Methode. Diese Methode "getImageJson" liefert aufgrund von vielen Parametern eine Liste von Bildern. Da die meisten Parameter optional sind, ist diese Zahl relativ zu bewerten.

Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum
Abstractness (avg/max per packageFragment)		0.101	0.151	0.444	/SnapItServer/src/ch/snapit/business
src		0.159	0.164	0.444	/SnapItServer/src/ch/snapit/business
test		0	0	0	/SnapItServer/test/ch/snapit/persist
Specialization Index (avg/max per type)		0.03	0.153	1	/SnapItServer/src/ch/snapit/service/SnapItApplicatio...
src		0.025	0.152	1	/SnapItServer/src/ch/snapit/service/SnapItApplicatio...
test		0.067	0.149	0.4	/SnapItServer/test/ch/snapit/stress/Client.java
Weighted methods per Class (avg/max per type)	386	8.042	9.857	54	/SnapItServer/src/ch/snapit/util/JSONWrapper.java
src	335	7.976	10.218	54	/SnapItServer/src/ch/snapit/util/JSONWrapper.java
test	51	8.5	6.801	21	/SnapItServer/test/ch/snapit/service/ServiceTest.java
Number of Static Attributes (avg/max per type)	84	1.75	8.102	57	/SnapItServer/src/ch/snapit/util/JSONConstants.java
src	78	1.857	8.651	57	/SnapItServer/src/ch/snapit/util/JSONConstants.java
test	6	1	0.816	2	/SnapItServer/test/ch/snapit/service/TestSuite.java

Abbildung 9-7: Codestatistik - Teil 3 – Metrics

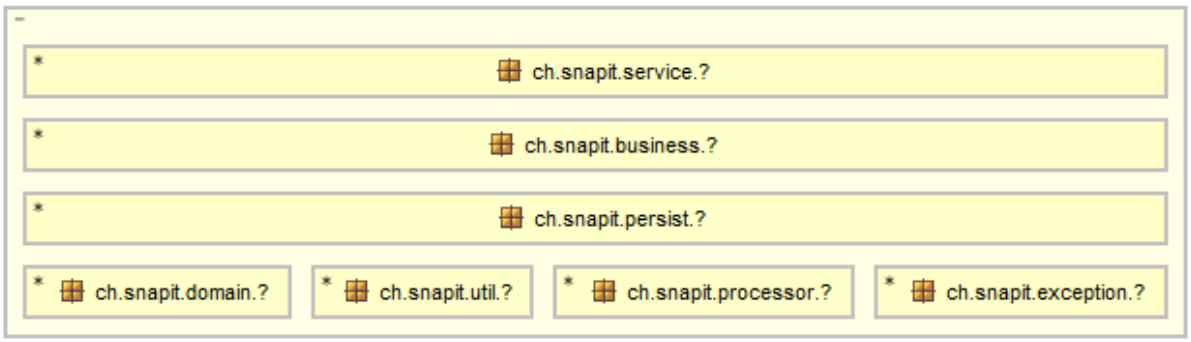

Abbildung 9-8: Packageübersicht

Die Schichtenarchitektur wurde sehr strikt umgesetzt. Es werden nur Klassen aus tieferen Shcichten verwendet. Die Ausnahmen stellen die Klassen des Utility-Packages, sowie die Domain-, Processor- und Exception-Klassen dar.

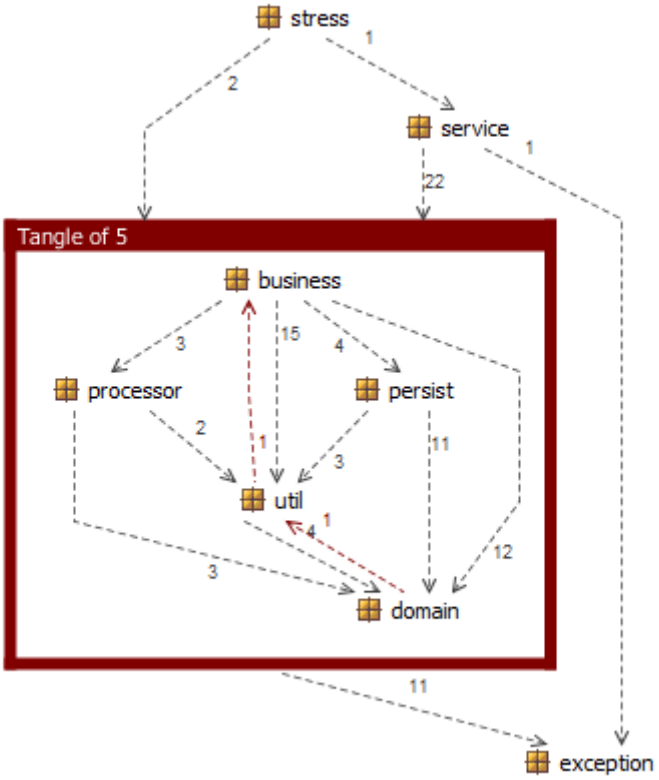

Abbildung 9-9: Abhängigkeitsgraph

Diese Abbildung zeigt die Abhängigkeiten zwischen den verschiedenen Packages. Die Abhängigkeiten wurden über das gesamte Projekt ermittelt, also auch über die Tests.

Der „Tangle of 5“ entsteht, aufgrund von zyklischen Abhängigkeiten unter den Packages. Um diese Abhängigkeiten aufzulösen hätte noch ein Refactoring gehört, konnte aber wegen Zeitknappheit nicht mehr durchgeführt werden. Folgende Aktionen wäre notwendig, um diese aufzulösen.

- Verschieben der statischen Methode "stringToSHA256"-Methode aus dem Business-Package ins Util-Package.
- Verschieben der Klasse ImageSize aus dem Util-Package ins Business-Package.

Nach diesen Änderungen würde der Tangle aufgelöst werden.

9.5.2 SnapIt-Client

Metric	Total	Mean	Std. Dev.	Maxim...	Resource causing Maximum	Method
Number of Overridden Methods (avg/max per	19	0.413	0.923	4	/SnapItClient/src/ch/snapit/activities/GalleryActivity.java	
src	19	0.487	0.984	4	/SnapItClient/src/ch/snapit/activities/GalleryActivity.java	
gen	0	0	0	0	/SnapItClient/gen/ch/snapit/R.java	
Number of Attributes (avg/max per type)	94	2.043	3.078	12	/SnapItClient/src/ch/snapit/activities/LocationAwareActivity.java	
src	94	2.41	3.208	12	/SnapItClient/src/ch/snapit/activities/LocationAwareActivity.java	
gen	0	0	0	0	/SnapItClient/gen/ch/snapit/R.java	
Number of Children (avg/max per type)	18	0.391	1.422	8	/SnapItClient/src/ch/snapit/rest/helper/ServiceResultReceiver.java	
src	18	0.462	1.533	8	/SnapItClient/src/ch/snapit/rest/helper/ServiceResultReceiver.java	
gen	0	0	0	0	/SnapItClient/gen/ch/snapit/R.java	
Number of Classes (avg/max per packageFragr	46	4.182	2.367	8	/SnapItClient/src/ch/snapit/activities/resultreceiver	
src	39	3.9	2.3	8	/SnapItClient/src/ch/snapit/activities/resultreceiver	
gen	7	7	0	7	/SnapItClient/gen/ch/snapit	
Method Lines of Code (avg/max per method)	1263	4.401	6.084	47	/SnapItClient/src/ch/snapit/activities/LocationAwareActivity.java	createEnterLocationDialog
src	1263	4.401	6.084	47	/SnapItClient/src/ch/snapit/activities/LocationAwareActivity.java	createEnterLocationDialog
gen	0	0	0			
Number of Methods (avg/max per type)	268	5.826	6.308	22	/SnapItClient/src/ch/snapit/rest/transaction/RESTTransaction.java	
src	268	6.872	6.305	22	/SnapItClient/src/ch/snapit/rest/transaction/RESTTransaction.java	
gen	0	0	0	0	/SnapItClient/gen/ch/snapit/R.java	
Nested Block Depth (avg/max per method)		1.244	0.61	4	/SnapItClient/src/ch/snapit/activities/GalleryActivity.java	finishDelete
src		1.244	0.61	4	/SnapItClient/src/ch/snapit/activities/GalleryActivity.java	finishDelete
gen		0	0			
Depth of Inheritance Tree (avg/max per type)		2.565	1.802	7	/SnapItClient/src/ch/snapit/activities/GalleryActivity.java	
src		2.846	1.819	7	/SnapItClient/src/ch/snapit/activities/GalleryActivity.java	
gen		1	0	1	/SnapItClient/gen/ch/snapit/R.java	
Number of Packages	11					
src	10					
gen	1					
Afferent Coupling (avg/max per packageFragn		8	6.994	24	/SnapItClient/src/ch/snapit/rest/helper	
src		7.3	6.958	24	/SnapItClient/src/ch/snapit/rest/helper	
gen		15	0	15	/SnapItClient/gen/ch/snapit	

Abbildung 9-10: Codestatistik - Teil 1 – Metrics

Metric	Total	Mean	Std. Dev.	Maxim...	Resource causing Maximum	Method
Number of Interfaces (avg/max per packageFr	0	0	0	0	/SnapItClient/src/ch/snapit/activities	
src	0	0	0	0	/SnapItClient/src/ch/snapit/activities	
gen	0	0	0	0	/SnapItClient/gen/ch/snapit	
McCabe Cyclomatic Complexity (avg/max per		1.491	1.078	10	/SnapItClient/src/ch/snapit/activities/CameraActivity.java	onCreate
src		1.491	1.078	10	/SnapItClient/src/ch/snapit/activities/CameraActivity.java	onCreate
gen		0	0			
Total Lines of Code	2728					
src	2600					
gen	128					
Instability (avg/max per packageFragment)		0.36	0.25	0.833	/SnapItClient/src/ch/snapit/rest/tasks	
src		0.396	0.234	0.833	/SnapItClient/src/ch/snapit/rest/tasks	
gen		0	0	0	/SnapItClient/gen/ch/snapit	
Number of Parameters (avg/max per method)		1.108	0.994	5	/SnapItClient/src/ch/snapit/rest/service/RESTServiceHelper.java	initiate
src		1.108	0.994	5	/SnapItClient/src/ch/snapit/rest/service/RESTServiceHelper.java	initiate
gen		0	0			
Lack of Cohesion of Methods (avg/max per typ		0.209	0.341	0.909	/SnapItClient/src/ch/snapit/activities/CameraActivity.java	
src		0.246	0.358	0.909	/SnapItClient/src/ch/snapit/activities/CameraActivity.java	
gen		0	0	0	/SnapItClient/gen/ch/snapit/R.java	
Efferent Coupling (avg/max per packageFragm		3.182	2.443	8	/SnapItClient/src/ch/snapit/activities/resultreceiver	
src		3.5	2.335	8	/SnapItClient/src/ch/snapit/activities/resultreceiver	
gen		0	0	0	/SnapItClient/gen/ch/snapit	
Number of Static Methods (avg/max per type)	19	0.413	1.883	11	/SnapItClient/src/ch/snapit/rest/helper/RESTLogger.java	
src	19	0.487	2.036	11	/SnapItClient/src/ch/snapit/rest/helper/RESTLogger.java	
gen	0	0	0	0	/SnapItClient/gen/ch/snapit/R.java	
Normalized Distance (avg/max per packageFra		0.586	0.286	1	/SnapItClient/gen/ch/snapit	
gen		1	0	1	/SnapItClient/gen/ch/snapit	
src		0.545	0.267	0.889	/SnapItClient/src/ch/snapit/rest/service	
Abstractness (avg/max per packageFragment)		0.059	0.1	0.286	/SnapItClient/src/ch/snapit/activities	
src		0.065	0.103	0.286	/SnapItClient/src/ch/snapit/activities	
gen		0	0	0	/SnapItClient/gen/ch/snapit	

Abbildung 9-11: Codestatistik - Teil 2 – Metrics

Metric	Total	Mean	Std. Dev.	Maxim...	Resource causing Maximum
Specialization Index (avg/max per type)		0.203	0.445	1.75	/SnapItClient/src/ch/snapit/activities/CameraActivity.java
src		0.24	0.475	1.75	/SnapItClient/src/ch/snapit/activities/CameraActivity.java
gen		0	0	0	/SnapItClient/gen/ch/snapit/R.java
Weighted methods per Class (avg/max per typ	428	9.304	10.54	45	/SnapItClient/src/ch/snapit/activities/GalleryActivity.java
src	428	10.974	10.616	45	/SnapItClient/src/ch/snapit/activities/GalleryActivity.java
gen	0	0	0	0	/SnapItClient/gen/ch/snapit/R.java
Number of Static Attributes (avg/max per type	214	4.652	8.357	41	/SnapItClient/gen/ch/snapit/R.java
gen	113	16.143	15.132	41	/SnapItClient/gen/ch/snapit/R.java
src	101	2.59	3.65	15	/SnapItClient/src/ch/snapit/rest/transaction/RESTTransaction.java

Abbildung 9-12: Codestatistik - Teil 3 – Metrics

10 Benutzer- und Entwicklerhandbuch

10.1 Webservice Extension Developer Handbuch

Die folgenden Kapitel beschreiben, wie die SnapIt-REST-Schnittstellen angesprochen werden können. Diese Schnittstellen können gebraucht werden, um eine Webseite oder um die Client-Applikation auf anderen Plattformen zu erstellen.

10.1.1 Einführung

Diese Abschnitte erklären und zeigen anhand konkreter Beispiele auf, wie die REST-Schnittstelle auf dem Server angesprochen wird.

Die IP des Testservers lautet wie folgt: 152.96.56.24

Der Basispfad zur REST-Service-Schnittstelle der Applikation SnapItServer lautet:

- <BASE-URL> = http://152.96.56.24:80/SnapItServer/rs/

Für die bessere Lesbarkeit wird folglich nur noch mit der "<BASE-URL>" gearbeitet. Diese kann, falls die Serveradresse ändert, einfacher korrigiert werden. Des Weiteren wird definiert, dass mit Applikation die SnapItServer-Applikation gemeint ist.

10.1.2 User

Der Pfad zur Benutzer resp. User-Ressource lautet:

- <BASE-URL-USER> = <BASE-URL>user/

new.json

Beschreibung

Der GET-Aufruf new.json fordert die Applikation auf eine neue eindeutige User-Id zu generieren. Falls dies erfolgreich war, wird diese zurückgegeben.

Aufruf

GET <BASE-URL-USER>new.json?transId=<Int32>

Beispiel

GET <BASE-URL-USER>new.json?transId=1234567

{"userId":"1u9900gueoswqinu","transId":1234567,"status":0}

valid.json

Beschreibung

Der GET-Aufruf valid.json wird gebraucht, um eine schon mal generierte User-Id auf ihre Gültigkeit zu überprüfen. Damit sichergestellt werden kann, dass mit einer korrekten User-Id gearbeitet wird, denn für CUD-Operationen wird eine vorausgesetzt.

Aufruf

GET <BASE-URL-USER>valid.json?transId=<Int32>&userId=<UserId>

Beispiel

GET <BASE-URL-USER>valid.json?transId=1234567&userId=1u9900gueoswqinu

{"userIdValid":true,"transId":1234567,"status":0}

10.1.3 Image

Um Bild bzw. Image-Ressourcen anzusprechen wird die BASE-URL erweitert.

- <BASE-URL-IMAGE> = <BASE-URL>image/

{imageId}.json

Beschreibung

Die Bild-Ressource mit der ID imageId kann über die URL <BASE-URL-IMAGE>/{imageId}.json angesprochen werden. Dabei ist {imageId} durch die Bild ID zu ersetzen. Mit GET wird einfach das entsprechende Bild geholt und mit DELETE wird das Bild aus der Applikation gelöscht.

Beim GET-Aufruf kann noch die Grösse (= size) des Bildes mitgeschickt werden, in welcher man es erhalten möchte.

- size:
  - 0 -> default, nur Bildinformationen
  - 1 -> klein
  - 2 -> mittel
  - 3 -> gross

Aufruf

GET <BASE-URL-IMAGE>/{imageId}.json?transId=<Int32>&size=<SIZE>

DELETE <BASE-URL-IMAGE>/{imageId}.json?transId=<Int32>&userId=<UserId>

Beispiel

GET <BASE-URL-IMAGE>/5.json?transId=1234567&size=1

{"id":5,"date":1292353538402,"imageByte":"BASE64_IMAGE","clubName":"","valid":false,"transId":1234567,"status":0}

DELETE <BASE-URL-IMAGE>/5.json?transId=1234567&userId=1u9900gueoswqinu

{"transId":1234567,"status":0}

{imageId}.jpg

Dieser Aufruf erfolgt gleich, wie der GET von {imageId}.json mit dem Unterschied, dass hier nur das Bild in binärem Format zurückgegeben wird, ohne zusätzliche Informationen.

Wichtig:

Wenn kein "size"-Parameter mitgegeben wurde, dann kommt kein Bild zurück. Es muss also zwingend eine Grösse von 1, 2 oder 3 mitgeschickt werden.

Beispiel

GET <BASE-URL-IMAGE>/5.jpg&size=2

Bild: 5.jpg

Abbildung 10-1: Beispielbild

last.json

Beschreibung

Der last.json-Aufruf verhält sich exakt gleich wie der GET {imageId}.json-Aufruf. Einziger Unterschied ist, dass hier einfach das zu letzt heraufgeladene Bild angezeigt wird. Somit fällt die Bild ID weg.

Aufruf

GET <BASE-URL-IMAGE>last.json?transId=<Int32>

Beispiel

GET <BASE-URL-USER>last.json?transId=1234567&size=1

{"id":5,"date":1292353538402,"imageByte":"BASE64_IMAGE","clubName":"","valid":false,"transId":1234567,"status":0}

last.jpg

Beschreibung

Der last.json-Aufruf verhält sich exakt gleich wie der GET {imageId}.jpg-Aufruf. Einziger Unterschied ist, dass hier einfach das zu letzt heraufgeladene Bild angezeigt wird. Somit fällt die Bild ID weg.

Aufruf

GET <BASE-URL-IMAGE>last.jpg?transId=<Int32>

Beispiel

GET <BASE-URL-USER>last.jpg?transId=1234567&size=1

Bild: last.jpg

Abbildung 10-2: Beispielbild

{imageId}/belongs.json

Beschreibung

Dieser Aufruf gibt zurück, ob das Bild mit der ID {imageId} dem mitgegebenen Benutzer "userId" gehört. D.h. ob dieser Benutzer das Bild geknipst hat.

Aufruf

GET <BASE-URL-IMAGE>{imageId}/belongs.json?transId=<Int32>&userId=<UserId>

Beispiel

GET <BASE-URL-IMAGE>5/belongs.json?transId=1234567&userId=1u9900gueoswqinu

{"belongs":false,"transId":1234567,"status":0}

all.json

Beschreibung

Mit diesem Aufruf wird eine Liste von Bildern zurückgegeben. Diese Liste kann mit verschieden parametrisiert werden. Die Beschreibung der einzelnen Parameter ist unten zu finden.

Aufruf

GET <BASE-URL-IMAGE>all.json?transId=<Int32>&userId=<UserId>&skip=<Int32>&count=<Int32>&size=<SIZE>&clubId=<ClubId>&startDate=<DateTime>&endDate=<DateTime>&id=<ImageId>&orderBy=<ORDER>

Es müssen nicht alle Parameter mitgegeben werden. Wenn nichts angegeben wurde wird dies ignoriert und den Standardwert genommen.

Beschreibung der Parameter:

- userId: Fotos nur vom Benutzer
- skip: Überspringt die angegebene Anzahl an Bildern, Standardwert ist 0
- count: Wie viele Bilder maximal zurückgegeben werden sollen, kann kleiner sein
- size: In welcher Grösse die Bilder zurückgegeben werden sollen.
- clubId: nur jene Bilder dieses bestimmten Clubs
- startDate: Nur Bilder, die nach diesem Zeitpunkt gemacht worden sind.

- endDate: Nur Bilder, die vor diesem Zeitpunkt gemacht worden sind.
- id: Bilder mit grösserer Id als die mitgegebene
- orderBy: Bilder werden entweder aufsteigend oder absteigend nach dem Erstellungszeitpunkt geordnet.

Beispiel

GET <BASE-URL-USER>/all.json?transId=1234567&orderBy=dateASC&count=20&startDate=&endDate=&userId=


```
{ "images" : [ { "clubName" : "",
        "date" : 1292334782668,
        "id" : 1,
        "latitude" : 47.223347425460815,
        "longitude" : 8.816989660263062,
        "valid" : false
      },
      { "clubId" : 20,
        "clubLatitude" : 47.22592193051487,
        "clubLongitude" : 8.816742897033691,
        "clubName" : "Restaurant Sayori",
        "date" : 1292334794668,
        "id" : 3,
        "latitude" : 47.223272323608398,
        "longitude" : 8.816887736320496,
        "valid" : true
      },
      { "clubName" : "",
        "date" : 1292353534918,
        "id" : 4,
        "valid" : false
      },
      { "clubName" : "",
        "date" : 1292353538402,
        "id" : 5,
        "valid" : false
      },
      { "clubName" : "",
        "date" : 1292354951293,
        "id" : 6,
        "valid" : false
      }
    ],
  "moreImages" : false,
  "status" : 0,
  "transId" : 1234567
}
```


Abbildung 10-3: JSON-Antwort - Images

Der Schlüssel "moreImages" gibt an, ob mit der gleichen Abfrage bzw. mit skip = count und count = count + 1 noch weitere Bilder abrufbar wären. Dieser Wert sollte nur benutzt werden, wenn genau in diesem Fall jetzt 20 Bilder zurückgegeben worden sind. Wenn die Liste weniger als 20 Bilder bzw. kleiner als der "count"-Parameter-Wert ist, dann gibt es keine weiteren Bilder.

{imageId}/changeClub.json

Beschreibung

Dieser Service wird genutzt, um einem Bild {imageId} eine neuer Club zuzuweisen. Dabei muss zwingend eine Benutzer ID mitgegeben werden.

Aufruf

POST <BASE-URL - IMAGE>{imageId}/changeClub.json?transId=<Int32>&userId=<UserId>

Body:

{"clubId":<ClubId>} oder {"clubName":"<ClubName>"} wenn beides vorhanden ist, wird die Club ID genommen.

Beispiel

POST <BASE-URL- IMAGE>5/changeClub.json?transId=1234567&userId=1u9900gueoswqinu

Body:

{"clubId":4}

{"transId":1234567,"status":0}

new.json

Beschreibung

Dieser Service wird aufgerufen, um ein neues Bild dem System hinzuzufügen. Dabei muss ein gültiger Benutzer mitgegeben werden.

Zusätzlich kann noch die Club ID oder der Name des Clubs mitgegeben werden. Dies verhält sich gleich wie wenn der Club des Bildes geändert werden soll.

Des Weiteren ist es möglich die Kamera-Orientation mitzugeben. Das System dreht dann das Bild in die gewünschte Position.

Folgende Werte für die Orientation sind möglich:

- 1: Portrait
- 2: Landscape right
- 3: Landscape left
- 4: Portrait backwards

Aufruf

PUT <BASE-URL-IMAGE>/new.json?transId=<Int32>&userId=<UserId>

Body:

{imageBytes:"<BASE64_Picture>", clubId=<ClubId>, cameraOrientation=<Orientation>}

Beispiel
PUT <BASE-URL-IMAGE>/new.json?transId=1234567&userId=1u9900gueoswqinu

Body: {imageBytes:"..BASE64_Picture..", clubId=4, cameraOrientation=1}

{"id":5,"valid":true,"transId":1234567,"status":0}

"valid" gibt an ob der mitgegebene Club ein validierter Club ist.

10.1.4 Club

Um Clubs bzw. Club-Ressourcen anzusprechen wird die BASE-URL erweitert.

- <BASE-URL-CLUB> = <BASE-URL>club/

nearest.json

Beschreibung
Dieser Service ermöglicht es alle Clubs in einem gewissen Radius zu den mitgegebenen Koordinaten zu erhalten.

Aufruf
GET <BASE-URL-CLUB>/nearest.json?transId=<Int32>&latitude=<Latitude>&longitude=<Longitude>

Beispiel
GET <BASE-URL-CLUB>/nearest.json?transId=1234567&latitude=47.228573987139136&longitude=8.821849822998047

```
{ "clubs" : [ { "clubId" : 13,
        "clubLatitude" : 47.228573987139136,
        "clubLongitude" : 8.821849822998047,
        "clubName" : "Bonanno Bistro & Bar",
        "valid" : true
      },
      { "clubId" : 17,
        "clubLatitude" : 47.22646109562935,
        "clubLongitude" : 8.815562725067139,
        "clubName" : "Corso Lounge",
        "valid" : true
      },
      { "clubId" : 8,
        "clubLatitude" : 47.442213000000002,
        "clubLongitude" : 9.3442640000000008,
        "clubName" : "Engelburg Schulhaus",
        "valid" : true
      },
      { "clubId" : 2,
        "clubLatitude" : 47.22379654652788,
        "clubLongitude" : 8.816716074943542,
        "clubName" : "Gebäude Bibliothek",
        "valid" : true
      },
      { "clubId" : 6,
        "clubLatitude" : 47.223636246712104,
        "clubLongitude" : 8.816066980361938,
        "clubName" : "Gebäude IFS",
        "valid" : true
      }
    ],
  "status" : 0,
  "transId" : 1234567
}
```


Abbildung 10-4: JSON-Antwort - Clubs

all.json

Beschreibung

Dieser Service ermöglicht es alle Clubs zu erhalten. Wenn noch eine Benutzer-ID mitgegeben wurde, dann werden nur jene Clubs zurückgeschickt, in denen der Benutzer mindestens ein Bild geknipst hat.

Aufruf

GET <BASE-URL-CLUB>/all.json?transId=<Int32>&userId=<UserId>

Beispiel

GET <BASE-URL-CLUB>/all.json?transId=1234567&userId=1u9900gueoswqinu

```
{ "clubs" : [ { "clubId" : 13,
        "clubLatitude" : 47.228573987139136,
        "clubLongitude" : 8.821849822998047,
        "clubName" : "Bonanno Bistro & Bar",
        "valid" : true
      },
      { "clubId" : 17,
        "clubLatitude" : 47.22646109562935,
        "clubLongitude" : 8.815562725067139,
        "clubName" : "Corso Lounge",
        "valid" : true
      },
      { "clubId" : 8,
        "clubLatitude" : 47.442213000000002,
        "clubLongitude" : 9.3442640000000008,
        "clubName" : "Engelburg Schulhaus",
        "valid" : true
      }
    ],
  "status" : 0,
  "transId" : 1234567
}
```


Abbildung 10-5: JSON-Antwort - Alle Clubs

In diesem Beispiel wurde der Benutzer mitgegeben und somit werden nur diese Clubs zurückgeschickt, in denen dieser mindestens ein Bild gemacht hat.

Wenn kein Benutzer angegeben wurde, dann werden alle Clubs, welche validiert sind zurückgeschickt.

10.2 Installation Server

Um die SnapIt-Server-Applikation auf einem Server zu installieren sind folgende Schritte notwendig:

1. Tomcat Version 6 herunterladen und installieren.
    a. Port 80
2. PostgreSQL Version 8.4 herunterladen und mit den Standard-Werten installieren.
    a. Passwort: postgres
    b. User: postgres
    c. Port: 5432
3. Jetzt kann entweder das Ant-Script, welches im SVN-Ordner (dbscript) des Servers ausgeführt werden, oder eine Datenbank mit dem Name „snapit“ erstellt werden und die Scripts manuell ausgeführt werden, die sich auch im oben genannten Ordner befinden.
4. Danach kann der Applikationsserver gestartet werden und das SnapIt-War-File über den Tomcat Web Application Manager installiert werden.

10.3 SnapIt – Client-App

10.3.1 Installation

Die SnapIt-App kann wie jede andere Android-App installiert werden. Da die App aber zur Zeit nicht veröffentlicht wurde, muss die lokale apk-Datei vom Computer aus installiert werden. Einfacher ist die Installation über die Eclipse-SEU möglich, wo die App direkt auf dem verbundenen Gerät gestartet werden kann.

Die neuste Version des App kann auch direkt über das Smartphone installiert werden. Hierzu greift man mit dem Browser auf folgende URL zu: http://152.96.56.24/SnapItServer/SnapItClient.apk

10.3.2 Anwendung

Eine gute Übersicht über die Grundfunktionen der App ist im Produktevideo zu sehen (http://www.youtube.com/watch?v=Jm8i83lvtqI). Da der Funktionsumfang übersichtlich und die Bedienung grösstenteils selbsterklärend ist, werden hier die wichtigsten Möglichkeiten nur kurz erläutert. Genauere Informationen sind auch der Userinterface-Dokumentation zu entnehmen.

Benutzer-ID

Die Benutzer-ID dient dem zuordnen der Fotos zu einem Fotografen. Da diese ID im System eindeutig sein muss, wird sie vom Server zur Verfügung gestellt. Die ID-kann auch auf mehreren Smartphones verwendet werden. Die Benutzer-ID ist eine reine Zufallszahl und hat somit keinen Bezug zum wahren Benutzer.

Wir die Applikation das erste Mal gestartet, erscheint die Aktivierungs-Seite. Der Benutzer kann hier eine bestehende Benutzer-ID eingeben („ID eingeben“) oder sich eine generieren lassen („ID generieren“). Für diesen Vorgang ist zwingend eine Internetverbindung erforderlich.

Lokation

Nach dem erfolgreichen Aktivieren oder jedem weiteren Starten der App, gelangt der Benutzer direkt zur Kamera. Es erscheint ein Fenster, das die Eingabe bzw. Auswahl der aktuellen Lokation verlangt. Die vorgeschlagenen Lokationen wurden anhand von Geo-Daten ermittelt. Es wird empfohlen, wenn möglich, eine Lokation auszuwählen und nur eine manuelle Eingabe zu tätigen, wenn die gewünschte Lokation nicht ersichtlich sein sollte. Jede Eingabe oder Auswahl muss mit einem Klick auf den Button „???“ bestätigt werden. Die aktuell gewählte Lokation ist in der Leiste oben im Display ersichtlich.

Sollte die gewählte Lokation nicht mehr mit dem realen Aufenthaltsort des Benutzers übereinstimmen, sollte der sich über einen Klick auf den Lokationsnamen erneut lokalisieren.

Falls der Benutzer nicht Bescheid über den aktuellen Aufenthaltsort hat oder seine Position aus anderen Gründen nicht übermitteln will, kann die aktuelle Lokation mit einem Klick auf das Kreuz-Icon im oberen linken Ecken gelöscht werden.

Foto knipsen und übermitteln

Ist das gewünschte Motiv visiert, genügt ein Klick auf den SnapIt-Button um ein Fotos aufzunehmen und den Schnappschuss und die aktuelle Lokation dem Server zu

übermitteln. Wurde das Foto übermittelt oder ist der Upload-Vorgang fehlgeschlagen, erhält der Benutzer über eine Meldung im linken, oberen Ecken des Display Bescheid.

Fotos, die aufgrund fehlender Verbindung nicht übermittelt werden konnten, werden beim nächsten Starten der Applikation oder beim manuellen Aktualisieren der Gallery über das Menü erneut übermittelt.

Eigene Fotos anzeigen

Die eigenen Fotos sind in der Galerie ersichtlich. Dazu genügt ein Klick auf eine Upload-Meldung oder auf das Gallery-Icon im Menü. Es werden Zwecks Performance immer nur elf Bilder auf einmal geladen. Weitere Bilder können über das Icon am Ende der Galerie nachgeladen werden. Sind keine weiteren Fotos vorhanden, ist auch das Icon nicht mehr ersichtlich.

Das Symbol im Foto gibt Auskunft über die Lokalisierung des Fotos. Wird ein grüner Haken angezeigt, konnte das Foto einer validen Lokation zugeordnet werden. Das gelbe Ausrufezeichen hingegen zeigt an, dass die Lokation nicht existiert, noch nicht freigegeben oder keine Lokation gewählt wurde.

Mit einem Klick auf ein Foto wird eine grössere Version geladen und der Aufnahmezeitpunkt und die Lokation angezeigt.

Foto verwalten

Alle Fotos können gelöscht oder einer anderen Lokation zugeteilt werden. Hierfür muss ein langer Klick auf das gewünschte Bild ausgeführt werden. Nun kann der Benutzer eine Aktion auswählen.

Credits der App anzeigen

Informationen zur App findet man auf der Credits-Seite, welche in der Kamera-Ansicht über das Menü erreichbar ist. Dort findet man auch einen Link, um die Webseite aufzurufen, wo die Fotos von allen Anwendern ersichtlich sind.

11 Projekt Rückblick

11.1 Danksagung

Wir möchten unserem Betreuer Prof. Dr. Markus Stolze für die hilfreichen Tipps und die Unterstützung während unserer Studienarbeit ganz herzlich danken.

11.2 Erfahrungsberichte

11.2.1 Christoph Süess

Schon einige Zeit vor dem Start der Semesterarbeit haben wir uns mit unserer Idee beschäftigt. Es freute mich, dass wir Herr Stolze von der Idee überzeugen konnten und er als Betreuer der Arbeit zusagte. Die moderne Thematik und der Einsatz von neuen Technologien sorgten für weitere Motivationsschübe.

Die Arbeit verlief sehr gut. Vor allem die Zusammenarbeit mit Raphael Nagy machte mir Spass, da man auch bei ihm dieselbe Motivation spürte. Das Vertrauen in die gegenseitigen Kenntnisse und die gute Kommunikation während und vor dem Projekt sorgten für eine gelungene Arbeitsteilung, was sich wiederum auf die Produktivität auswirkte.

Herr Stolze betreute das Projekt erfolgreich, in dem er uns immer wieder mit kritischen Fragen und Anregungen auf Stolpersteine aufmerksam machte. Ausserdem erinnerte er uns immer wieder an die Grenzen und Ziele und sorgte somit dafür, dass das Projekt nicht aus den Rudern lief. Diese Sicht einer dritten Person auf den Verlauf des Projekts war sehr wertvoll.

Probleme entstanden wie so oft bei der Aufwand- und Zeitplanung. Der bekannte Übereifer und unrealistische Zielsetzungen mussten Anfangs der Projekts gebändigt werden. Die bereits vorhandene Erfahrung half uns dies zu meistern. Mühe hatten wir auch mit dem einhalten bzw. bestimmen des Vorgehensmodell unseres Projektes (siehe Lessons Learned).

Alles in allem war das Projekt wieder eine super Vorbereitung auf das Berufsleben. Neben einem sehr grossen Knowhow-Gewinn im technischen Bereich konnte ich auch viel zum Thema Projektarbeit lernen. Ich freue mich jetzt schon darauf die neuen Erkenntnisse beim Erarbeiten der Bachelorarbeit einzusetzen und so wieder ein Schritt näher an das perfekte Projekt zu kommen. Ich bin sehr zufrieden was unser Team in der doch sehr kurzen Zeit erreicht hat und bedanke mich bei meinem Teamkollegen Raphael Nagy für die super Zusammenarbeit.

11.2.2 Raphael Nagy

Für mich begann das Projekt schon vor dem eigentlichen Projektstart am Anfang des Semesters. Christoph Süess und ich arbeiteten schon während den Sommer-Semesterfeien intensiv an unserer Arbeit, dadurch dass wir unser eigenes Projektthema einbringen konnten war die Motivation umso grösser. Diese Vorarbeiten mit den Interviews und Fragebögen bzw. deren positive Rückmeldungen bestätigten mich immer stärker mit der Idee.

Die Frage nach den einzusetzenden Technologien war einerseits durch die Aufgabenstellung vorgegeben und andererseits komplett offen. Relativ schnell evaluierten wir die einzusetzenden Technologien und stellten voller Eifer einen Projektplan auf die Beine. Aber bald merkten wir, dass wir uns ein wenig zu viel vorgenommen hatten. Dadurch hatten wir die Möglichkeit den Umfang runterzusetzen oder einfach mehr zu Arbeiten. Für uns beide war klar, dass eine gewisse Funktionalität vorhanden sein werden musste, um das ganze Potential der Idee aufzuzeigen. Darum arbeiteten wir härter und mehr daran.

Ich war vor allem für die serverseitige Applikation zuständig. Zu den Schwerpunkten gehörten die Datenspeicherung mit Hibernate und PostgreSQL, sowie die Service-Schnittstelle mit Jersey-REST. Im Nachhinein, kann ich sagen, dass ich mich jederzeit wieder für diese Wahl entscheiden würde. Dies zeigt sich auch in der performanten Applikation, die daraus entstanden ist.

Die wöchentlichen Besprechungen mit Herrn Stolze waren sehr hilfreich, da der Input aus Sicht einer dritten Person wichtig ist, um das Ziel nicht aus den Augen zu verlieren und neue Ideen oder Anregungen zu erhalten. Mit der Betreuung bin ich vollkommen zufrieden und durch die kreativen Anregungen von Herrn Stolze sehr förderlich für das Projekt.

Der Knowhow-Gewinn in allen Bereichen war für mich enorm. Auch die Arbeit mit meinem Teamkollegen Christoph Süess war sehr gut. Es gab nie Probleme und sein enormes Wissen in Android-Bereich war ein wesentlicher Punkt, weshalb die Arbeit aus meiner Sicht die Anforderungen bei weitem übertroffen hat. An dieser Stelle möchte ich mich für die super Zusammenarbeit ganz herzlich bei Christoph Süess bedanken.

11.3 Lessons Learned

11.3.1 Definieren des Vorgehensmodell

Von Anfang an war für uns klar, dass wir ein agiles Vorgehensmodell verwenden wollten. Dies führte leider am Anfang zu Schwierigkeiten. Einerseits lag das Problem an unserem fehlenden Knowhow, andererseits basierten viele der SA-Vorgaben auf dem RUP-Vorgehensmodell. So planten wir anfänglich gewisse Dinge nach RUP und versuchten dann mit SCRUM weiterzuarbeiten. Da die beiden Vorgehensmodelles sich absolut unterscheiden, war es nicht möglich dies sauber durchzuführen. Schlussendlich organisierten wir unser eigenes Vorgehensmodell, das sich aber stark nach SCRUM richtete.

Fazit: Das Vorgehensmodell muss vor Projektstart festgelegt werden und ausnahmslos eingehalten werden. Nur so kann man von den Vorteilen profitieren. Ein iteratives Model ist unbedingt einem Wasserfall-Vorgehen vorzuziehen. Nur wer das gewählt Model auch komplett versteht, kann es effizient einsetzen.

11.3.2 Arbeiten mit neuen Technologien

Das Arbeiten mit neuen Technologien ist immer eine sehr interessante Sache. Neue Technologien basieren meistens auf modernen Konzepten, was das Arbeiten vereinfacht. Wichtiger als die Kenntnis der Vorteile, ist das man weiss was die Schwierigkeiten sind, die grosse Stolpersteine darstellen können. Folgende Punkte sind vor allem bei der Arbeit mit dem Android-Framework aufgefallen:

- **Verbesserungswürdiges Framework**: Junge Technologien haben auch öfters noch gewisse Schwachstellen oder effektiv Fehler. Für den Entwickler besteht die Schwierigkeit darin, abzuschätzen ob ein gewisser Fehler durch die Technologie verursacht wird, oder durch die falsche Implementierung durch den Programmierer.
- **Müllhalde Internet**: Dieser Titel ist absolut gerechtfertigt. Gerade bei Trendtechnologien wie Android, versuchen tausende von Anfängern Tutorials, Anleitungen und Tipps im Internet zu verbreiten. Leider sind sehr viele der Beispiele sehr schlecht und auf die Beispielimplementation absolut kein Verlass. Wer blind kopiert, wird irgendwann auf magische Fehler bei der Ausführung des App stossen. Diese Fehler zu finden und zu beheben kostet massiv Zeit.
- **Dokumentation**: Leider geben sich nicht alle Entwickler von solchen Technologien so viel Mühe mit der Dokumentation, wie es Google bei Android macht. Das Android-Framework ist im Vergleich zu anderen Technologien sehr gut dokumentiert, auch wenn die einte oder andere Lücke noch zu finden ist.

Fazit: Das Arbeiten mit neuen Technologien macht Spass und ist sehr interessant. Wer die Motivationskurve hoch halten will, sollte die Sache jedoch vorsichtig angehen und weder dem Framework noch den Dokumentationen blindlinks vertrauen.

11.3.3 Media-Wiki als Projektplattform

In unserem Projekt haben wir ein einfaches Media-Wiki als Projektplattform eingesetzt. Das Wiki ist innerhalb von wenigen Minuten installiert und konfiguriert. Auch das Arbeiten mit dem Wiki ist sehr einfach.

Wir haben sehr gute Erfahrungen damit gemacht und würden es immer wieder einsetzen. Im Gegensatz zu der Variante direkt mit Dokumenten zu arbeiten, ist im Wiki alles zentral vorhanden und die Versionsverwaltung wird direkt durch das System gemanagt. Ausserdem kann man von überall her auf die Plattform zugreifen.

Das Erfassen von neuem Inhalt ist sehr simple gehalten, was sehr positiv war. Leider ergab die Einfachheit auch Probleme. So ist das Erstellen einer Tabelle sehr unübersichtlich, da es keine grafische Unterstützung gibt. Auch das Einbinden von Bildern ist mühsam, da das praktische Drag-and-Drop nicht funktioniert.

Fazit: Wiki eignet sich sehr gut für eine Projektplattform. Gewisse Features wären jedoch sehr wünschenswert.

11.3.4 Arbeitsteilung und Teamwork

Wir sind überzeugt, dass unsere Projektarbeit ein gutes Beispiel für gelungene Teamarbeit war. Unser Projekt eignete sich super um das Problem in zwei Teile zu gliedern, den Client- und den Serverteil. Raphael hat sich fast ausschliesslich mit dem Server auseinandergesetzt und Christoph mit dem Client. Wichtige Entscheidungen wurden aber immer zusammen beschlossen.

Diese Aufteilung brachte diverse Vorteile: Jeder von uns musste nur halb so viele Technologien beherrschen. Durch die schmale Schnittstelle kamen unsere Arbeiten sich nur selten in die Quere und der Koordinierungsaufwand hielt sich in Grenzen. Beide von uns hatten auch immer eine Person, die die Problemstellung von einem anderen Winkel betrachtete. So entstanden gute Ideen.

Fazit: Will man eine Teamarbeit effektiv anpacken, sollte eine geeignete Aufteilung der Arbeiten gemacht werden. Wichtig ist auch, dass die Schnittstellen klar definiert werden. Trotz allem bleibt die Kommunikation unter den Teammitgliedern der wichtigste Punkt und darf auf keinen Fall zu kurz kommen.

12 Anhang

12.1 Glossar

Club	Bezieht sich auf eine konkrete Party-Lokation
GPS	Global Position System. Technologie zur Bestimmen der aktuellen Position.
JDBC	Java Database Connectivity. Datenbankschnittstelle der Java-Plattform.
Lokation	In den technischen Berichten eine Alias für ein Länge-, Breitengrad-Paar
PMD	Werkzeug zur statischen Codeanalyse
REST	Representational State Transfer. Softwarearchitektur für verteilte Informationssysteme.
SnapIt	Entwicklungsname der Software. Eine eigene Wortschöpfung aus den Englischen Worten "Snap" (knipsen) und "It" (es).

12.2 Literatur- und Quellenverzeichnis

12.2.1 Literatur

[BUC04] Buchmann, Johannes: Einführung in die Kryptographie, Springer, 3., erweiterte Auflage, 2004

12.2.2 Internet

[1] http://tools.ietf.org/html/rfc2616 - 02.12.2010

12.3 Abbildungsverzeichnis

12.4 Zeitplan

Anmerkung: In der Zeiterfassung sind die drei letzten Tagen nicht enthalten.

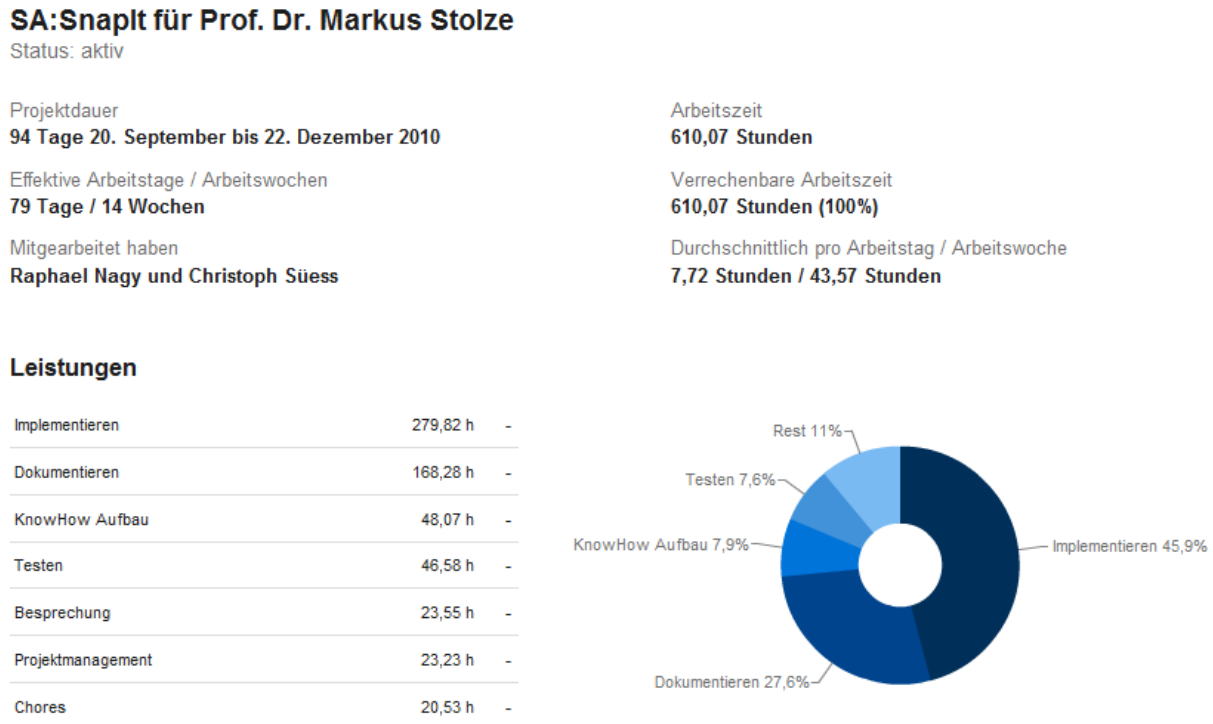

Abbildung 12-1: Zeiterfassung

Datum	Leistung	Benutzer	Bemerkung	Stunden
Gesamt				610:04
20. Sep 10	Dokumentieren	Raphael Nagy	Projektplan aufgesetzt	1:00
20. Sep 10	Projektmanagement	Christoph Süess		1:00
21. Sep 10	Dokumentieren	Raphael Nagy	Risikomanagement erstellt	1:00
21. Sep 10	Dokumentieren	Raphael Nagy	Anforderungsspezifikation erstellt (Initial)	1:00
22. Sep 10	Dokumentieren	Raphael Nagy	User-Stories	2:00
22. Sep 10	Dokumentieren	Raphael Nagy	Glossar erstellt (Initial)	0:10
22. Sep 10	Dokumentieren	Raphael Nagy	Zeitplan erstellt	2:00

22. Sep 10	Projektmanagement	Christoph Süess	Sitzungsvorbereitung	2:00
23. Sep 10	Chores	Raphael Nagy	Eclipse installiert	2:30
23. Sep 10	Chores	Raphael Nagy	SVN aufgesetzt	0:20
23. Sep 10	Besprechung	Christoph Süess		1:00
23. Sep 10	Projektmanagement	Christoph Süess		4:15
23. Sep 10	Dokumentieren	Christoph Süess		3:00
23. Sep 10	Dokumentieren	Christoph Süess	Skizzen der Iterationen	2:00
24. Sep 10	Dokumentieren	Raphael Nagy	Sitzungsprotokoll nachgeführt	0:30
24. Sep 10	Dokumentieren	Christoph Süess	- Iterationen geplant - Vision & Projektskizze - User Stories für Iteration erfasst	3:00
24. Sep 10	Chores	Christoph Süess	Wiki eingerichtet	0:15
25. Sep 10	Dokumentieren	Raphael Nagy	Anforderungsspezifikation	1:00
25. Sep 10	Dokumentieren	Raphael Nagy	UseCases	0:30
25. Sep 10	Chores	Raphael Nagy	Datenbank	0:30
25. Sep 10	Chores	Raphael Nagy	Hibernate vorbereitet	0:30
27. Sep 10	KnowHow Aufbau	Christoph Süess	REST, JSON, Android	1:45
27. Sep 10	Projektmanagement	Christoph Süess		1:00
27. Sep 10	Dokumentieren	Raphael Nagy	Projektplan	2:00
28. Sep 10	Dokumentieren	Christoph Süess	U. a. Umfrage und Resultate ins Wiki übernommen	1:15
28. Sep 10	Projektmanagement	Christoph Süess		1:20
29. Sep 10	Projektmanagement	Christoph Süess		1:00
29. Sep 10	Dokumentieren	Christoph Süess		1:00
29. Sep 10	KnowHow Aufbau	Raphael Nagy	Hibernate	6:00
30. Sep 10	Besprechung	Christoph Süess		1:00
30. Sep 10	Projektmanagement	Christoph	Sitzungresultate	0:15

		Süess	nachgetragen	
30. Sep 10	Dokumentieren	Christoph Süess		4:30
30. Sep 10	Projektmanagement	Christoph Süess		3:00
1. Okt 10	Chores	Christoph Süess		1:00
1. Okt 10	Dokumentieren	Christoph Süess		1:15
2. Okt 10	KnowHow Aufbau	Raphael Nagy	REST	5:00
3. Okt 10	KnowHow Aufbau	Raphael Nagy	REST	3:30
4. Okt 10	KnowHow Aufbau	Christoph Süess	REST-Architektur für Android	2:30
6. Okt 10	Chores	Christoph Süess		2:00
6. Okt 10	Projektmanagement	Christoph Süess		1:00
6. Okt 10	KnowHow Aufbau	Raphael Nagy	PostgreSQL & Hibernate	5:00
7. Okt 10	Besprechung	Christoph Süess		1:00
7. Okt 10	Implementieren	Christoph Süess	- REST Connection	6:00
7. Okt 10	KnowHow Aufbau	Christoph Süess		2:00
7. Okt 10	Dokumentieren	Raphael Nagy	Verschiedene	8:00
8. Okt 10	KnowHow Aufbau	Christoph Süess		2:00
8. Okt 10	Dokumentieren	Christoph Süess		0:30
9. Okt 10	Implementieren	Raphael Nagy	REST	3:00
10. Okt 10	Implementieren	Raphael Nagy	REST	4:00
11. Okt 10	KnowHow Aufbau	Christoph Süess		2:00
11. Okt 10	Projektmanagement	Christoph Süess		1:00
13. Okt 10	Dokumentieren	Raphael Nagy	Testplan Architektur	2:00
13. Okt 10	Dokumentieren	Raphael Nagy	Alte Dokumente aktualisiert	2:00
14. Okt 10	KnowHow Aufbau	Christoph Süess		2:00
14. Okt 10	Implementieren	Christoph		7:00

		Süess		
14. Okt 10	Besprechung	Christoph Süess		1:00
14. Okt 10	Implementieren	Raphael Nagy	Architektur	8:00
15. Okt 10	Dokumentieren	Christoph Süess		2:00
16. Okt 10	Implementieren	Raphael Nagy	Architektur	5:00
17. Okt 10	Testen	Raphael Nagy	JUnit-Tests erstellt	2:30
18. Okt 10	Projektmanagement	Christoph Süess		2:00
18. Okt 10	Dokumentieren	Christoph Süess		1:00
18. Okt 10	Implementieren	Christoph Süess		3:00
19. Okt 10	Testen	Raphael Nagy	REST Webservice Tests	2:30
19. Okt 10	Projektmanagement	Raphael Nagy	SVN, Tag erstellt	0:30
19. Okt 10	Implementieren	Raphael Nagy	REST Businesslogik	2:00
19. Okt 10	Dokumentieren	Raphael Nagy		0:30
20. Okt 10	Implementieren	Raphael Nagy	Logging	1:00
20. Okt 10	Implementieren	Raphael Nagy	Server	1:30
20. Okt 10	Testen	Raphael Nagy	Deployment testen	2:00
21. Okt 10	Implementieren	Raphael Nagy	Architektur	3:00
21. Okt 10	Besprechung	Christoph Süess		1:00
21. Okt 10	Besprechung	Raphael Nagy		1:00
21. Okt 10	Projektmanagement	Christoph Süess	Todos der letzten Sitzung abgearbeitet	1:00
21. Okt 10	Implementieren	Christoph Süess	Android Archtitektur	9:10
21. Okt 10	Implementieren	Raphael Nagy	Architektur	2:00
21. Okt 10	Implementieren	Raphael Nagy	Architektur	1:00
21. Okt 10	Implementieren	Raphael Nagy	Architektur	0:20
22. Okt 10	Implementieren	Raphael	Architektur	5:23

		Nagy		
22. Okt 10	Implementieren	Christoph Süess		1:48
22. Okt 10	Projektmanagement	Christoph Süess	Sitzungresultate nachgetragen	0:15
22. Okt 10	Projektmanagement	Christoph Süess	Zeitplan nachgetragen	0:42
22. Okt 10	Projektmanagement	Christoph Süess	Besprechung mit Raphael	0:17
25. Okt 10	Implementieren	Raphael Nagy	Architektur	1:30
25. Okt 10	Dokumentieren	Raphael Nagy	Architektur	0:30
25. Okt 10	Implementieren	Christoph Süess	TransactionLog implementieren	1:40
25. Okt 10	Implementieren	Raphael Nagy	Architektur	0:50
25. Okt 10	Implementieren	Raphael Nagy	Architektur	0:50
26. Okt 10	Implementieren	Christoph Süess	ContentProvider --> RestTransactionDb	1:39
26. Okt 10	KnowHow Aufbau	Raphael Nagy	Location	1:29
27. Okt 10	KnowHow Aufbau	Raphael Nagy	Architektur	3:06
27. Okt 10	Implementieren	Christoph Süess	ContentProvider --> RestTransactionDb	4:41
27. Okt 10	Besprechung	Christoph Süess		0:40
27. Okt 10	Implementieren	Raphael Nagy		2:02
27. Okt 10	Besprechung	Raphael Nagy		0:50
27. Okt 10	KnowHow Aufbau	Raphael Nagy	Location	3:38
27. Okt 10	KnowHow Aufbau	Christoph Süess	Callbacks für Verbindungsänderungen	0:39
28. Okt 10	Testen	Raphael Nagy	Stress Test für Webservice	2:00
28. Okt 10	Implementieren	Raphael Nagy	Architektur	2:00
28. Okt 10	Implementieren	Raphael Nagy	Architektur	0:35
28. Okt 10	KnowHow Aufbau	Raphael Nagy	Android Testing	0:40
28. Okt 10	Dokumentieren	Raphael Nagy	Architektur	1:57
28. Okt 10	Implementieren	Christoph		9:00

		Süess		
29. Okt 10	Dokumentieren	Raphael Nagy	Architektur	1:48
29. Okt 10	Chores	Raphael Nagy	DB Script	1:51
29. Okt 10	Chores	Raphael Nagy	Server	2:19
30. Okt 10	Implementieren	Raphael Nagy	Architektur	0:53
2. Nov 10	Implementieren	Raphael Nagy	Architektur	1:00
2. Nov 10	Implementieren	Christoph Süess		2:00
3. Nov 10	Testen	Raphael Nagy	REST Webservice Tests	2:39
3. Nov 10	Chores	Raphael Nagy	Weiteres Vorgehen	1:59
3. Nov 10	Dokumentieren	Raphael Nagy		1:35
3. Nov 10	Implementieren	Christoph Süess		4:00
3. Nov 10	Chores	Raphael Nagy	Workspace aufgeräumt	0:20
3. Nov 10	Chores	Raphael Nagy	Server	1:03
4. Nov 10	Dokumentieren	Christoph Süess		2:34
4. Nov 10	Dokumentieren	Raphael Nagy	Architektur	6:39
4. Nov 10	Besprechung	Christoph Süess		1:00
4. Nov 10	Besprechung	Raphael Nagy		1:23
4. Nov 10	Projektmanagement	Christoph Süess		0:40
4. Nov 10	Dokumentieren	Christoph Süess	SAD	3:16
4. Nov 10	Chores	Raphael Nagy	Latex	0:00
4. Nov 10	Implementieren	Christoph Süess		2:05
5. Nov 10	Implementieren	Christoph Süess		4:10
5. Nov 10	Dokumentieren	Raphael Nagy		1:20
5. Nov 10	Implementieren	Raphael Nagy	Refactoring	2:27
5. Nov 10	Chores	Raphael	Server aufsetzen	1:57

		Nagy		
6. Nov 10	Implementieren	Christoph Süess		2:31
6. Nov 10	Chores	Raphael Nagy	Serverkonfig. testen	0:26
6. Nov 10	Dokumentieren	Raphael Nagy	Virtuelle Partymeile	0:20
6. Nov 10	KnowHow Aufbau	Raphael Nagy	Locationsbestimmung auf Android	1:22
7. Nov 10	Implementieren	Raphael Nagy	Locationsbestimmung auf Android	1:49
7. Nov 10	Implementieren	Raphael Nagy	Lokation	1:12
8. Nov 10	Implementieren	Christoph Süess		2:37
8. Nov 10	Dokumentieren	Christoph Süess	Rest-Client-Architektur dokumentiert	1:03
8. Nov 10	Chores	Raphael Nagy	Erste Test mit Testgerät	1:10
9. Nov 10	Implementieren	Christoph Süess	Kamera implementieren	2:07
9. Nov 10	KnowHow Aufbau	Raphael Nagy	Testing	2:06
9. Nov 10	Implementieren	Raphael Nagy	Server	1:13
9. Nov 10	Dokumentieren	Christoph Süess	JavaDoc	1:13
10. Nov 10	Implementieren	Raphael Nagy	Location	2:13
10. Nov 10	KnowHow Aufbau	Raphael Nagy	Anonym	1:09
10. Nov 10	Implementieren	Raphael Nagy	Features	2:44
10. Nov 10	Implementieren	Raphael Nagy	Features	3:53
10. Nov 10	Implementieren	Christoph Süess		4:23
10. Nov 10	Chores	Raphael Nagy	Webseite, kleine Verbesserungen	1:23
11. Nov 10	Dokumentieren	Christoph Süess		2:36
11. Nov 10	Implementieren	Raphael Nagy	Testing, Fixes	2:56
11. Nov 10	Implementieren	Christoph Süess		4:00
11. Nov 10	Implementieren	Raphael Nagy	location	3:11
11. Nov 10	Implementieren	Raphael	Server	2:03

		Nagy		
11. Nov 10	Implementieren	Raphael Nagy	Server	0:53
13. Nov 10	Dokumentieren	Raphael Nagy	Testplan Iteration 3 & Durchführung	1:39
13. Nov 10	Implementieren	Raphael Nagy	Bild drehen um 90 Grad	0:36
16. Nov 10	Besprechung	Raphael Nagy		12:00
17. Nov 10	Besprechung	Raphael Nagy	Weiteres Vorgehen	0:40
17. Nov 10	Implementieren	Raphael Nagy	User-ID auf serverseite erstellen	0:55
17. Nov 10	Projektmanagement	Christoph Süess		1:00
17. Nov 10	Implementieren	Christoph Süess		3:35
17. Nov 10	Implementieren	Raphael Nagy	User-ID	0:32
17. Nov 10	Chores	Raphael Nagy	Neuer Version auf Server geladen & Logging angepasst	0:37
17. Nov 10	Implementieren	Raphael Nagy	SHA-256 Hash	0:30
17. Nov 10	Implementieren	Raphael Nagy	Webservice: Userkey auf seine Gültigkeit prüfen	0:23
18. Nov 10	Implementieren	Raphael Nagy	Refactoring	0:20
18. Nov 10	Testen	Raphael Nagy		1:23
18. Nov 10	Implementieren	Christoph Süess		1:30
18. Nov 10	Projektmanagement	Christoph Süess		1:00
18. Nov 10	Implementieren	Christoph Süess	GUI für SnapItClient	4:29
18. Nov 10	Implementieren	Christoph Süess		4:30
19. Nov 10	Implementieren	Christoph Süess		6:10
19. Nov 10	Implementieren	Raphael Nagy	Arbeit an diversen Services	3:41
20. Nov 10	Testen	Raphael Nagy		2:03
20. Nov 10	Implementieren	Raphael Nagy	Refactoring	2:37
20. Nov 10	Implementieren	Raphael Nagy	Refactoring	1:13

21. Nov 10	Implementieren	Christoph Süess		2:00
22. Nov 10	Implementieren	Christoph Süess		8:23
22. Nov 10	Implementieren	Raphael Nagy		2:02
22. Nov 10	Implementieren	Raphael Nagy		2:13
22. Nov 10	Implementieren	Raphael Nagy		3:22
22. Nov 10	Implementieren	Raphael Nagy		1:53
23. Nov 10	Implementieren	Christoph Süess		2:58
23. Nov 10	Implementieren	Raphael Nagy		1:53
24. Nov 10	Implementieren	Christoph Süess		2:17
24. Nov 10	Implementieren	Raphael Nagy	Refactoring	3:02
24. Nov 10	KnowHow Aufbau	Raphael Nagy		0:57
25. Nov 10	Implementieren	Christoph Süess		4:33
25. Nov 10	Implementieren	Raphael Nagy		5:47
25. Nov 10	Besprechung	Christoph Süess		1:00
26. Nov 10	Implementieren	Raphael Nagy	Webseite	1:32
29. Nov 10	Implementieren	Christoph Süess		4:14
29. Nov 10	Implementieren	Raphael Nagy	Webseite	1:06
29. Nov 10	Implementieren	Raphael Nagy	Einige Verbesserungen: Tests, Services, Webseite	0:26
30. Nov 10	Implementieren	Raphael Nagy		1:46
1. Dez 10	Implementieren	Raphael Nagy		0:37
1. Dez 10	Implementieren	Christoph Süess		8:01
1. Dez 10	Testen	Raphael Nagy		0:45
1. Dez 10	Testen	Raphael Nagy		0:47
2. Dez 10	Implementieren	Christoph Süess		9:30

2. Dez 10	Implementieren	Raphael Nagy	Verbesserungen	1:33
4. Dez 10	Testen	Raphael Nagy		1:40
6. Dez 10	Implementieren	Christoph Süess		4:57
6. Dez 10	Implementieren	Raphael Nagy	Refactoring	0:49
6. Dez 10	KnowHow Aufbau	Raphael Nagy	Rotate Image	1:13
7. Dez 10	Implementieren	Raphael Nagy		1:15
8. Dez 10	Implementieren	Raphael Nagy	Refactoring	2:00
8. Dez 10	Implementieren	Raphael Nagy	Refactoring	1:00
8. Dez 10	Implementieren	Raphael Nagy	Refactoring	3:39
8. Dez 10	Testen	Christoph Süess		23:15
8. Dez 10	Testen	Christoph Süess		1:31
8. Dez 10	Implementieren	Raphael Nagy	Refactoring	0:32
9. Dez 10	Testen	Christoph Süess		2:09
9. Dez 10	Implementieren	Christoph Süess		7:34
9. Dez 10	Implementieren	Raphael Nagy	Fixes	4:23
9. Dez 10	Implementieren	Raphael Nagy	Refactoring	4:03
9. Dez 10	Implementieren	Raphael Nagy	Inline Dokumentation	1:43
10. Dez 10	Implementieren	Raphael Nagy	Refactoring	3:11
12. Dez 10	Dokumentieren	Raphael Nagy		2:00
13. Dez 10	Dokumentieren	Raphael Nagy	Architektur	1:22
13. Dez 10	Dokumentieren	Raphael Nagy		1:30
13. Dez 10	Dokumentieren	Raphael Nagy		2:00
13. Dez 10	Chores	Raphael Nagy	Update SnapItServer auf Server	0:22
14. Dez 10	Dokumentieren	Raphael Nagy		0:16

14. Dez 10	Implementieren	Christoph Süess	2:16
15. Dez 10	Dokumentieren	Raphael Nagy	0:16
15. Dez 10	Dokumentieren	Raphael Nagy	1:00
15. Dez 10	Dokumentieren	Raphael Nagy	4:33
15. Dez 10	Dokumentieren	Christoph Süess	3:22
15. Dez 10	Dokumentieren	Raphael Nagy	0:52
15. Dez 10	Testen	Christoph Süess	1:23
16. Dez 10	Dokumentieren	Raphael Nagy	4:41
16. Dez 10	Dokumentieren	Raphael Nagy	2:00
16. Dez 10	Dokumentieren	Raphael Nagy	2:36
16. Dez 10	Dokumentieren	Raphael Nagy	1:10
16. Dez 10	Dokumentieren	Christoph Süess	5:23
17. Dez 10	Dokumentieren	Christoph Süess	12:59
17. Dez 10	Dokumentieren	Raphael Nagy	5:30
18. Dez 10	Dokumentieren	Christoph Süess	6:34
19. Dez 10	Dokumentieren	Christoph Süess	8:01
20. Dez 10	Dokumentieren	Raphael Nagy	0:30
20. Dez 10	Dokumentieren	Raphael Nagy	4:00
20. Dez 10	Dokumentieren	Christoph Süess	5:45
21. Dez 10	Dokumentieren	Raphael Nagy	4:55
21. Dez 10	Dokumentieren	Christoph Süess	5:01
22. Dez 10	Dokumentieren	Raphael Nagy	6:47
22. Dez 10	Dokumentieren	Christoph Süess	5:34

12.5 ToDo-Liste

Die Todo-Liste wurde während dem Projekt genutzt um Arbeiten zu notieren die während Besprechungen und Diskussionen auftauchten. Es wurden nicht alle anfallenden Aufgaben hier nachgetragen. Die Todo-Liste diente vor allem dem Zweck, dass keine Aufgabe vergessen ging.

Datum	Task	Prio	Zuständig	Erledigen bis
23.09.2010	Zwei UserStories aus Sicht eines Fotos - CHS (Foto knippsen u. betrachten), RN (Foto knippsen u. keine Location gefunden)	2	-	25.09.2010
23.09.2010	UserStories um Bad Case Szenarios erweitern	2	-	25.09.2010
23.09.2010	Stakeholder-Rollen zuordnen bzw. wer sind die Stakeholder (evt. auflisten)	2	-	25.09.2010
23.09.2010	Szenario aus Sicht eines Barbesitzers	2	-	25.09.2010
23.09.2010	Antrag 1.6 Android Device an Herr Stolze	1	-	25.09.2010
23.09.2010	Formulieren eines Vorschlag im Bezug auf die Nutzerrechte der HSR bzw. von uns.	1	-	25.09.2010
30.09.2010	Abnahmetest der Iteration 1 ändern --> Abnahmetest ist erfolgreich wenn Herr Stolze ok gibt	2	-	06.10.2010
30.09.2010	Empirischer Test = Testplan. Auf Iterationsseite ändern.	2	-	06.10.2010
30.09.2010	FotoStory in Artefact-Story ändern	2	-	06.10.2010
30.09.2010	Im Projektplan JavaDoc als Qualitätsmerkmal festhalten --> Javadoc für alle Klassen	2	-	06.10.2010
14.10.2010	Iteration 1 muss eine Referenz auf den Projektantrag haben	2	-	20.10.2010
14.10.2010	Usecase ergänzen: Benutzer wird über den Status seiner Aktionen informiert	2	-	20.10.2010
14.10.2010	Usecase: Bezug zu User-Stories herstellen (Verlinken)	2	-	20.10.2010
14.10.2010	Domain Modell in spätere Iteration verschieben	2	-	20.10.2010
14.10.2010	Konkurrenz-Analyse als Entwurf kennzeichnen	2	-	20.10.2010
14.10.2010	Anforderungsspezifikation in Iteration 4 verschieben	2	-	20.10.2010
14.10.2010	Risikomanagement unterteilen: Projektrisiken + Spezifische Iterationsrisiken	2	-	20.10.2010
21.10.2010	Zeitplan bis zur nächsten Sitzung nachtragen	1	-	27.10.2010
04.11.2010	Globale ToDo-Liste aufsetzten	2	RN	04.11.2010
27.10.2010	Datenaustauschprotokoll SnapIt --> Verweise auf bestehende Protokolle	2	RN	02.11.2010
04.11.2010	Alte ToDos in die zentrale ToDo-Liste übertragen	2	CHS	04.11.2010

04.11.2010	Aus den Sitzungen resultierende ToDos in zentrale Liste übernehmen, sowie im Sitzungsprotokoll erwähnen (Nachtragen nur in zentraler Liste)	2	CHS	04.11.2010
04.11.2010	In allen Sitzungsprotokollen "Resultate" durch "Entscheidungen" ersetzen	2	CHS	04.11.2010
14.10.2010	Begründen der Technologieentscheide (Iteration 2)	2	RN & CHS	11.10.2010
04.11.2010	Risiken für Iteration 2 festhalten	1	CHS & RN	11.11.2010
04.11.2010	Risiken für Iteration 3 festhalten	2	CHS & RN	11.11.2010
04.11.2010	Sequenzdiagramm für REST-Ablauf (Server)	2	RN	11.11.2010
06.11.2010	Beschreiben was Client-REST-Architektur kann und was nicht (Kann: Transaction-Tracking, Kann nicht: Caching)	2	CHS	11.11.2010
04.11.2010	Zentrales REST-Architektur-Dokument erstellen	1	CHS	11.11.2010
21.10.2010	Entscheidungen zur Architektur begründen	2	-	11.10.2010
11.11.2010	Wird beim Bildupload kein Club oder eine ganz neue Location mitgegeben, erscheint eine Liste um den Club zu korrigieren (Clientseitig implementieren)	1	CHS	15.11.2010
11.11.2010	Klubname beim GET eines Bildes neben Bildname anzeigen (Client)	1	CHS	15.11.2010
11.11.2010	Klubname beim GET eines Bildes mitliefern (Server)	1	RN	15.11.2010
11.11.2010	PUT-Operationen werden auf dem Server nur einmal ausgeführt (Damit keine Bild-Duplikate entstehen)	1	RN	15.11.2010
11.11.2010	Testplan für Iteration 3 erstellen und durchführen (Einen Klub erfassen der nicht in der Nähe ist und einen der in Rapperswil ist)	1	RN	15.11.2010
11.11.2010	Angepasster Testplan für Iteration 2 erstellen und durchführen	1	CHS	15.11.2010
17.11.2010	Service der neuen User-Key zurückliefert und Hash speichert	1	RN	24.11.2010
17.11.2010	Webservice: Prüft einen Userkey auf seine gültigkeit (Return: true, false)	2	RN	24.11.2010
17.11.2010	Matrix, bei Vorteilen, einen Verweis auf z.B. ÄnderbarkeitTestplan It. 2 --> Datum des Test einfügen	2	CHS	24.11.2010
17.11.2010	Alle benutzerbezogenen Services korrekt anpassen	1	RN	24.11.2010
17.11.2010	Webservice: Alle Location in denen ein User mind. ein Bild erfasst hat	2	RN	24.11.2010
17.11.2010	Webservice: Alle Bilder die von einem User erfasst wurden	2	RN	24.11.2010
17.11.2010	Webservice: Ein Bild eines Users (Validieren ob Bild wirklich zu User gehört)	2	RN	24.11.2010
17.11.2010	Testplan It. 3 --> Datum des Test einfügen	2	RN	24.11.2010
17.11.2010	Möglichkeit einen Club selber zu erfassen (in jeder Situation)	1	CHS	24.11.2010

17.11.2010	Allen Userbezogenen Operationen UserKey mitschicken (immer Hash des Keys)	1	CHS	24.11.2010
17.11.2010	Show Key Methode	1	CHS	24.11.2010
17.11.2010	App nur starten wenn Key vorhanden	1	CHS	24.11.2010
17.11.2010	Service für selbst erfasste Clubs erstellen. Selbst erfasste Clubs werden auf dem Server wenn möglich einem Existierenden zugeordnet und ansonsten mit einem "NOT_VALIDATED_CLUB"-Status versehen	1	RN	24.11.2010
04.11.2010	Entscheid festhalten & begründen: "Warum keine Unittests für REST-Serverseite?" & "Unit-Tests vs. TestNG"	2	RN	11.11.2010
04.11.2010	Packagediagramm REST-Architektur (Client)	2	CHS	11.11.2010
13.12.2010	Risikomanagement ergänzen	1	RN / CHS	-
17.11.2010	Dokumentieren welche Probleme uns bei der Ermittlung der Position aufgefallen sind und wie wird diese ermittelt haben.	2	RN	24.11.2010
30.09.2010	Am Schluss muss ein Video in dieser Form vorliegen (zeitbeständige Demo) --> Scrum Table	3	CHS / RN	-
04.11.2010	Dokumentieren Entscheid "Maven vs. Ant" - Entwicklungsumgebung	2	RN	11.11.2010
04.11.2010	Dokumentieren Technologieentscheid PostgreSQL - Architektur Server/Snapit	2	RN	11.11.2010
30.09.2010	Für die Iterationen jeweils auch die Chores ("Hausarbeiten" wie z. B. SVN aufsetzen) festhalten	2	-	-
04.11.2010	In den Iterations-Grafiken steht "SnaptIt" --> ändern --> "SnapIt"	2	CHS	-
17.11.2010	Client "Pseudo-Diagramm" --> Nummerierung (für Ablauf)	1	CHS	24.11.2010

12.6 Sitzunsprotokolle

Sitzung vom 23.09.2010

- Woche: 1
- Teilnehmer: CHS, RN, MS
- Abgenommen am: 30.09.2010

Traktanden

- Projekt Kick-Off
- Genauer Projektrahmen festlegen
- Offene Fragen besprechen
- Verschiebungstermin für Meeting vom 30.09.2010
- Milestones & Projektplan
- Inhaltsverzeichnis (Entwurf)

Offene Fragen

- Virtuelle Clublandschaft HSR
- Besteht die Möglichkeit ein Testgerät (Android-Handy) leihweise zu erhalten?
- Wir benötigen einen Server, um ev. Hudson, Tomcat usw. laufen zu lassen, werden diese von der HSR zur Verfügung gestellt?
- SVN-Repository?
- Rechtliche Lage à Fertiges Programm, Code, usw. nach Ende der Arbeit, während der Entwicklung?
- Möglichkeit Bachelor-Arbeit darauf aufzubauen und fortzuführen?
- Können wir auf Dokumente (Word, PDF, ..) verzichten und alles in einem Wiki dokumentieren (abgesehen von der Standardangabe --> siehe Richtlinien)?

Entscheidungen

- Die Benutzer von SnapIt bleiben total anonym. Ihre Fotos sind nicht auf Sie zurückzuführen. Ihre Identität besteht nur aus einem generierten Key. Wir speichern keine Angaben zum Telefon und zur Handynr.
- Auf Bilder die aufgrund der heraufgeladenen Informationen nicht vollständig verarbeitet werden konnten, erhält der Benutzer nicht direkt nach dem Upload ein Feedback. Der Benutzer entdeckt diese Bilder auf der Übersichtsseite und wird beim Starten von SnapIt informiert.
- Die Clientapplikation wird mit Android realisiert und nicht mit HTML 5. Diese Entscheidung wird in einigen Sätzen festgehalten.
- Die Iterationen werden wie im Entwurf besprochen festgehalten. Jeder Iteration wird ein Abnahmetest zugewiesen und auf eine UserStory verwiesen.
- Auf Personas kann verzichtet werden.
- Android 1.6 Geräte sind vorhanden, die wir nutzen können (--> Antrag). Es wird über die Anschaffung von einem Android 2.2 Gerät nachgedacht, das wir Tagesweise verwenden könnten.
- Für das Projekt wird kein Hudson verwendet.
- Die Sitzung vom 30.09.2010 findet ohne Herr Nagy statt.

Resultierende ToDos

- Design Decision Android vs. HTML 5 festhalten
  - Features werden nur partiell von den verschiedenen Browsern unterstützt.
  - Nicht alle Neuentwicklungen in HTML 5 sind schon als Standard zu betrachten.
  - Für uns gibt es zu wenige Möglichkeiten mit dem Mobiltelefon zu interagieren.
  - Applikation kann (teilweise) nur mit einer bestehenden Internetverbindung genutzt werden.
- Beschreiben wie weit das GUI von Android und von der Webseite implementiert wird.
- Zwei UserStories aus Sicht eines Fotos - CHS (Foto knipsen u. betrachten), RN (Foto knipsen u. keine Location gefunden)
- UserStories um Bad Case Szenarios erweitern
- Stakeholder-Rollen zuordnen bzw. wer sind die Stakeholder (evt. auflisten)

- Szenario aus Sicht eines Barbesitzers
- Antrag 1.6 Android Device an Herr Stolze
- Formulieren eines Vorschlags in Bezug auf die Nutzerrechte der HSR bzw. von uns.

Sitzung vom 30.09.2010

- Woche: 2
- Teilnehmer: CHS, MS
- Abgenommen am: 07.10.2010

Traktanden

- Bewertungskriterien von Herr Stolze besprechen
- Iterationsplan präsentieren & besprechen
- Weitere Resultate präsentieren
  - Vision & Projektübersicht
  - Entwurf Projektplan
  - Knowledge Dokument "Android REST client applications"

Offene Fragen
Keine

Entscheidungen
Keine

Resultierende ToDos

- Für die Iterationen jeweils auch die Chores ("Hausarbeiten" wie z. B. SVN aufsetzen) festhalten
- Abnahmetest der Iteration 1 ändern --> Abnahmetest ist erfolgreich wenn Herr Stolze ok gibt
- Empirischer Test = Testplan. Auf Iterationsseite ändern.
- FotoStory in Artefact-Story ändern
- Im Projektplan JavaDoc als Qualitätsmerkmal festhalten --> Javadoc für alle Klassen
- Bewertungstabelle von Herr Stolze nochmals mit RN besprechen
- Am Schluss muss ein Video in dieser Form vorliegen (zeitbeständige Demo) --> Scrum Table

Sitzung vom 07.10.2010

- Woche: 3
- Teilnehmer: RN, CHS, MS
- Abgenommen am: 14.10.2010

Traktanden

- Abschluss Iteration 1
- Besprechen und Unterzeichen der Nutzungsrechte-Vereinbarung

- Nochmals Bewertungskriterien durchgehen

Offene Fragen

- Möglichkeit Bachelor-Arbeit darauf aufzubauen und fortzuführen?
- Antrag Android Handy 2.2?

Entscheidungen

- Vereinbarung erfolgreich unterzeichnet
- Erweiterungsidee: User macht viele Fotos Amateur -> Profi
- Bewertungskriterien besprochen bis Punkt 110
- Handy
  - 1.6 Android bekommen wir bis spätestens nächste Woche
  - 2.2 Android können wir tageweise auslehnen
- Fachliche Risiken werden bei der jeweiligen Iteration im Analysedokument festgehalten

Resultierende ToDos

- Idee der "Walkcam" untersuchen / Gedanken machen -> nicht in Dokumentation erwähnen
- In 3 Verschiedene Clubs / Bars gehen am Wochenende
  - 5 Fotos schiessen mit Koordinaten
    - Je 1 Foto schiessen, nachdem Handy für 30 Minuten abgeschaltet ist
- Fachliche Risikofaktoren ergänzen: Koordinaten bestimmen als Risikofaktor aufnehmen
- Motivation verständlicher machen
  - "Walkcam"
  - Benutzer
- Visionsdokument erstellen --> Sinn
- Abklärung: Sony Automatikcam
- Scrum studieren

Sitzung vom 14.10.2010

- Woche: 4
- Teilnehmer: RN, CHS, MS
- Abgenommen am: 21.10.2010

Traktanden

- Abnahme Sitzungsprotokoll
- Besprechen Bewertungskriterien ab Punkt 110
- Bestätigung Abschluss 1. Iteration
- Weitere Resultate & Probleme besprechen (2. Iteration)

Offene Fragen

- Visionsdokument wird während dem Projekt ergänzt --> Siehe Scrum
- "Motivation verständlicher machen " --> Motivation wird aus UserStories ersichtlich (siehe Titel).
- Dinge wie Sony-Cam usw. sind Dinge die unser Projekt nicht beeinflussen. Es gibt hunderte von Idee, welche aber keinen Einfluss auf die SA haben. Wir sind der Meinung das solche Abklärungen unsere SA übersteigen?

Entscheidungen

- Abnahme des letzten Sitzungsprotokolle = OK
- Android Handy 1.6 ist kaputt, ein neues Handy wurde bestellt und ist in max. 2 Wochen nutzbar
- Vision: Aus User-Stories
- Abnahmekriterien erstellen für die Iterationen
- User-Stories OK
- Verschiedene Dokumente in spätere Iterationen verlegen
- Zu den Fragen: Cam: Nicht Teil unserer SA

Resultierende ToDos

- Vision aus User-Stories erstellen (erster Dokumententwurf)
- Iteration 1 muss eine Referenz auf den Projektantrag haben
- Testcase: Netzwechsel (WLAN - GSM)
- Usecase ergänzen: Benutzer wird über den Status seiner Aktionen informiert
- Usecase: Bezug zu User-Stories herstellen (Verlinken)
- Domain Modell in spätere Iteration verschieben
- Für die nächste Iteration, die Abnahmekriterien definieren
- Für alle Iterationen Abnahmekriterien erstellen (aber erst am Anfang der Iteration)
- Konkurrenz-Analyse als Entwurf kennzeichnen
- Anforderungsspezifikation in Iteration 4 verschieben
- Risikomanagement unterteilen: Projektrisiken und Spezifische Iterationsrisiken
- Begründen der Technologieentscheide

Sitzung vom 21.10.2010

- Woche: 5
- Teilnehmer: RN, CHS, MS
- Abgenommen am: 27.10.2010

Traktanden

- Abnahme Sitzungsprotokoll
- Besprechen Bewertungskriterien ab Punkt 110
- Bestätigung Abschluss 1. Iteration
- Weitere Resultate & Probleme besprechen (2. Iteration)
- Kleine DEMO
- Zeitplanung

Offene Fragen
Keine

Entscheidungen

- Die Iteration 1 wurde abgenommen
- Die Iteration 2 endet mit der Publizierung der Architektur in Foren und Communities (allfällige Verbesserungsvorschläge werden in den späteren Iteration umgesetzt)
- Das UI unserer Applikation wird "nur" auf Usability getestet, Design spielt im SA-Projekt keine Rolle
- Die nächste Sitzung wird auf den Mittwoch 27.10.10 08:05 verschoben!

Resultierende ToDos

- Experimentieren mit Latex um später das Wiki in ein Dokument umzuwandeln
- Zeitplan bis zur nächsten Sitzung nachtragen
- State-Machine eines Server-Request/Response ins Architektur-Dokument aufnehmen, falls Nutzen ersichtlich
- Entscheidungen zur Architektur begründen
- Begründung und planen des weiteren Vorgehens, falls die Zeitplanung nicht eingehalten werden kann
    - Scrum ist eigentlich Time Boxed: Nochmals einlesen und Vorgehen ermitteln

Sitzung vom 27.10.2010

- Woche: 6
- Teilnehmer: RN, CHS, MS
- Abgenommen am: 04.11.2010

Traktanden

- Abnahme Sitzungsprotokoll
- Einsehen der Zeitplanung
- Stand Iteration 2 besprechen, gegeben falls Massnahmen diskutieren
- Weitere Resultate & Probleme besprechen (2. Iteration)
    - Demo

Offene Fragen
Keine

Entscheidungen

- Letztes Sitzungsprotokoll abgenommen (OK)
- Zeitplanung bisher OK
- Sprint wird verlängert

Resultierende ToDos

- Zentrale ToDo-Liste einführen
- Datenaustauschprotokoll SnapIt --> Verweise auf bestehende Protokolle

Sitzung vom 04.11.2010

- Woche: 7
- Teilnehmer: RN, CHS, MS
- Abgenommen am: 15.11.2010

Traktanden

- Abnahme Sitzungsprotokoll
- Stand Iteration 2
- Server für SnapItServer-App.
- Weiteres Vorgehen

Offene Fragen

Keine

Entscheidungen

- Ziel: In der nächsten Woche wird Iteration 2 & 3 abgeschlossen
    - Blick in die REST-Architektur

Resultierende ToDos

- Alte ToDos in die zentrale ToDo-Liste übertragen
- Aus den Sitzungen resultierende ToDos in zentrale Liste übernehmen, sowie im Sitzungsprotokoll erwähnen (Nachtragen nur in zentraler liste)
- In allen Sitzungsprotokollen "Resultate" durch "Entscheidungen" ersetzen
- Sequenzdiagramm für REST-Ablauf (Client)
- Sequenzdiagramm für REST-Ablauf (Server)
- Zentrales REST-Architektur-Dokument erstellen
- Entscheid festhalten & begründen: "Warum keine Unittests für REST-Serverseite?" & "Unit-Tests vs. TestNG"
- Risiken für Iteration 2 festhalten
- Risiken für Iteration 3 festhalten
- Herausfinden, wie man im MITE-Tool die Leistungen pro Wochen anzeigen kann
- Dokumentieren Entscheid "Maven vs. Ant"

Sitzung vom 15.11.2010

- Woche: 9
- Teilnehmer: RN, CHS, MS
- Abgenommen am: 25.11.2010

Traktanden

- Abnahme Sitzungsprotokoll
- Abschluss Iteration 2
- Abschluss Iteration 3
- Stand Iteration 4

Offene Fragen

- Architektur Matrix, wie sollen Features mit User-Stories verbunden werden?
- SIM-Karte für Handy, damit Location abhängige Funktionen getestet werden können (Lokalisierung über das Handynetz)

Resultate

- Iteration 2:
  - Matrix, bei Vorteilen, einen Verweis auf z.B. Änderbarkeit
  - Client "Pseudo-Diagramm" --> Nummerierung (für Ablauf)
  - Unterbrechungsfreie .. beim Testplan (Rechtschreibung)
  - Übersichtsseite: gut dokumentierte / serverseitig (Rechtschreibung)
  - Nichtfunktionale Anforderungen, in wie weit schon erfüllt?
  - Als abgeschlossen ansehen (It. 2)
- Iteration 3:
  - Tests --> dann folgende Probleme
  - Datum beim Testplan einsetzen
- Nächste Woche gibt es eine SIM-Karte
- Funktionale Anforderungen erst in der nächsten Iteration (beim GUI)

Sitzung vom 25.11.2010

- Woche: 10
- Teilnehmer: RN, CHS, MS
- Abgenommen am: 09.12.2010

Traktanden

- Abnahme Sitzungsprotokoll
- Definitiver Abschluss Iteration 2 & 3

Offene Fragen

- Ajax: Wie kann man einen Request auf fremde Domains machen?

Resultate

- Iteration 2 & 3 abgeschlossen
- Entscheid: Komprimierung dokumentieren
- JUnit-Test: anonymisierte Bilder
- JUnit-Tests der anderen Gruppe ansehen
  - Wie haben sie das Lokalisierungsproblem gelöst?

- SIM-Karte erhalten wir bei der nächsten Sitzung

Sitzung vom 09.12.2010

- Woche: 11
- Teilnehmer: RN, CHS, MS
- Abgenommen am:

Traktanden

- Abnahme Sitzungsprotokoll
- Demo

Offene Fragen

Keine

Resultate

- Kontrast verbessern (bei Club-Auswahl-Liste), falls noch Zeit bleibt
- Nachdem aktualisiert wurde, bzw. die ausstehenden Bilder heraufgeladen wurden, diese in der Ansicht anzeigen
- Ev. Timer, um jede Stunde noch ausstehende Bilder automatisch heraufzuladen, falls noch Zeit bleibt
- Bilder-Ansicht ergänzen: einen Balken (oben) der anzeigt, wie viele Bilder noch heraufzuladen sind
- Youtube-Video kann einem Werbevideo gleichen
    - Sollte Nutzen aufzeigen
    - Technische Herausforderung kann als Text eingeblendet werden
    - Problem: Bsp. "Fotos anonym hochladen" --> FB ist nicht anonym

Sitzung vom 16.12.2010

- Woche: 12
- Teilnehmer: RN, CHS, MS
- Abgenommen am: -

Traktanden

- Abnahme Sitzungsprotokoll
- Dokumentenreview (vorgängiges Lesen empfohlen)
- Checkliste durchgehen

Offene Fragen

- Unterschied Abstract und Management Summary (aus Musterinhaltsverzeichnis)?
- Hinweis: Schwierigkeit Dokument aufgrund des iterativen Vorgehen
- Hinweis: Wiki nicht auf dem neusten Stand --> Dokument zählt
- Hinweis: Dokumentiert nur was wichtig ist, keine Pseudo-Dokus!
- Hinweis: Personas & Szenarios usw. abgesprochen

- Aufgabenstellung herein kopieren?
- Neuer Titel der Arbeit unpassend?
- Kriterien
  - 15. MS: Review Inhaltsverzeichnis Bericht
  - Zeile 109: Korrektes UML
  - JavaDoc

Resultate

- Abnahme Iteration 4, 5 & 6
- Abstract verbessern und nochmals schicken
- Plakat-Layout und Inhalt besprochen
- Videoablauf besprochen und abgenommen
- Keine Vorgaben bezüglich des Abgabe-Dokumentes (Formatierung usw.)
- 2 CDs brennen
- 1x Schwarz/Weiss gedrucktes Dokument an Herrn Stolze
- JavaDoc: Kommentare bei Klassen und Schnittstellen
- Inhaltsverzeichnis Review

12.7 Urheber- und Nutzungsrechte

Vereinbarung

Gegenstand der Vereinbarung

Mit dieser Vereinbarung werden die Rechte über die Verwendung und die Weiterentwicklung der Ergebnisse der Studienarbeit "SnapIt" von Raphael Nagy und Christoph Süess unter der Betreuung von Markus Stolze geregelt.

Urheberrecht

Die Urheberrechte stehen der Studentin / dem Student zu.

Verwendung

Die Ergebnisse der Arbeit dürfen sowohl von der Studentin / dem Student wie von der HSR nach Abschluss der Arbeit verwendet und weiter entwickelt werden.

Rapperswil, den	**Student – Raphael Nagy**
Rapperswil, den	**Student – Christoph Süess**
Rapperswil, den	**Prof. Dr. Markus Stolze**

12.8 Selbstverfassungserklärung

Erklärung

Wir erklären hiermit,

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

Ort, Datum: Ort, Datum:

Name, Unterschrift: Name, Unterschrift:

12.9 Javadoc

Die JavaDoc für die beiden Projekte befinden sich auf der beiliegenden CD.

12.10 Inhalte der CD

- Abstract
- Studienarbeit
- Poster
- Produktevideo
- JavaDoc-SnapIt-Client
- JavaDoc-SnapIt-Server
- SnapIt-Client
- SnapIt-Server
- SnapIt-Client APK