|  |
| --- |
| Elmer Lukas, Heidt Christina, Steiner Diego, Treichler Delia, Waltenspül Remo  17. Mai 2011 |

|  |
| --- |
| SE2 Projekt MRT |
| Software Architektur |
|  |

****

# Dokumentinformationen

## Änderungsgeschichte

|  |  |  |  |
| --- | --- | --- | --- |
| Datum | Version | Änderung | Autor |
| 28.03.2011 | 1.0 | Erste Version des Dokuments | HC |
| 05.04.2011 | 1.1 | Architekturkonzepte beschrieben | EL |
| 11.04.2011 | 1.2 | Vervollständigung des Inhaltsverzeichnisses | TD |
| 13.05.2011 | 1.3 | Sequenzdiagramme & Datenspeicherung hinzugefügt | WR |
| 14.05.2011 | 1.5 | Architekturkonzepte Rails beschrieben | DS |
| 15.05.2011 | 1.6 | Beschreibung der Packages | HC |
| 15.05.2011 | 1.7 | Review und Korrekturen | EL |
| 16.05.2011 | 1.8 | Prozesse und Threads, Grössen und Mengen | EL |

## Inhaltsverzeichnis

[1 Dokumentinformationen 1](#_Toc293321515)

[1.1 Änderungsgeschichte 1](#_Toc293321516)

[1.2 Inhaltsverzeichnis 1](#_Toc293321517)

[1.3 Abbildungsverzeichnis 4](#_Toc293321518)

[2 Einführung 5](#_Toc293321519)

[2.1 Zweck 5](#_Toc293321520)

[2.2 Gültigkeitsbereich 5](#_Toc293321521)

[2.3 Definitionen und Abkürzungen 5](#_Toc293321522)

[2.4 Referenzen 5](#_Toc293321523)

[3 Software Architektur 6](#_Toc293321524)

[3.1 Architektonische Ziele & Einschränkungen 6](#_Toc293321525)

[3.1.1 Ziele 6](#_Toc293321526)

[3.1.2 Einschränkungen 6](#_Toc293321527)

[3.2 Architekturübersicht 7](#_Toc293321528)

[3.3 Kernarchitektur Übertragung Zeiteintrag 8](#_Toc293321529)

[3.3.1 Sequenzdiagramm 8](#_Toc293321530)

[3.3.2 Zustandsdiagramm 9](#_Toc293321531)

[3.4 Systemstruktur 10](#_Toc293321532)

[3.4.1 Physische Sicht 10](#_Toc293321533)

[3.4.2 Logische Sicht Rails 10](#_Toc293321534)

[3.4.3 Logische Sicht Android 13](#_Toc293321535)

[3.5 Schnittstellen der Packages 15](#_Toc293321536)

[3.6 Architekturkonzepte 16](#_Toc293321537)

[3.6.1 Rails 16](#_Toc293321538)

[3.6.2 Android 19](#_Toc293321539)

[4 Beschreibung der Packages 23](#_Toc293321540)

[4.1 Interfaces 23](#_Toc293321541)

[4.1.1 Beschreibung 23](#_Toc293321542)

[4.1.2 Klassenstruktur 23](#_Toc293321543)

[4.1.3 Architekturkonzepte für Package (falls vorhanden) 23](#_Toc293321544)

[4.1.4 Klassenspezifikationen 23](#_Toc293321545)

[4.2 Network 24](#_Toc293321546)

[4.2.1 Beschreibung 24](#_Toc293321547)

[4.2.2 Klassenstruktur 24](#_Toc293321548)

[4.2.3 Architekturkonzepte für Package (falls vorhanden) 24](#_Toc293321549)

[4.2.4 Klassenspezifikationen 24](#_Toc293321550)

[4.2.5 Interaktionen innerhalb Package (falls sinnvoll) 25](#_Toc293321551)

[4.3 Models 25](#_Toc293321552)

[4.3.1 Beschreibung 25](#_Toc293321553)

[4.3.2 Klassenstruktur 26](#_Toc293321554)

[4.3.3 Architekturkonzepte für Package (falls vorhanden) 27](#_Toc293321555)

[4.3.4 Klassenspezifikationen 27](#_Toc293321556)

[4.3.5 Interaktionen innerhalb Package (falls sinnvoll) 27](#_Toc293321557)

[4.4 Database 27](#_Toc293321558)

[4.4.1 Beschreibung 27](#_Toc293321559)

[4.4.2 Klassenstruktur 27](#_Toc293321560)

[4.4.3 Architekturkonzepte für Package (falls vorhanden) 27](#_Toc293321561)

[4.4.4 Klassenspezifikationen 27](#_Toc293321562)

[4.4.5 Interaktionen innerhalb Package (falls sinnvoll) 28](#_Toc293321563)

[4.5 Application 28](#_Toc293321564)

[4.5.1 Beschreibung 28](#_Toc293321565)

[4.5.2 Klassenstruktur 28](#_Toc293321566)

[4.5.3 Architekturkonzepte für Package (falls vorhanden) 28](#_Toc293321567)

[4.5.4 Klassenspezifikationen 28](#_Toc293321568)

[4.5.5 Interaktionen innerhalb Package (falls sinnvoll) 28](#_Toc293321569)

[4.6 Services 28](#_Toc293321570)

[4.6.1 Beschreibung 29](#_Toc293321571)

[4.6.2 Klassenstruktur 29](#_Toc293321572)

[4.6.3 Architekturkonzepte für Package (falls vorhanden) 29](#_Toc293321573)

[4.6.4 Klassenspezifikationen 29](#_Toc293321574)

[4.6.5 Interaktionen innerhalb Package 31](#_Toc293321575)

[4.7 Gui/gen 32](#_Toc293321576)

[4.7.1 Beschreibung 32](#_Toc293321577)

[4.7.2 Klassenstruktur 32](#_Toc293321578)

[4.7.3 Architekturkonzepte für Package (falls vorhanden) 32](#_Toc293321579)

[4.7.4 Klassenspezifikationen 32](#_Toc293321580)

[4.7.5 Interaktionen innerhalb Package (falls sinnvoll) 32](#_Toc293321581)

[4.8 Activities 32](#_Toc293321582)

[4.8.1 Beschreibung 32](#_Toc293321583)

[4.8.2 Klassenstruktur 33](#_Toc293321584)

[4.8.3 Architekturkonzepte für Package (falls vorhanden) 33](#_Toc293321585)

[4.8.4 Klassenspezifikationen 33](#_Toc293321586)

[4.8.5 Interaktionen innerhalb Package (falls sinnvoll) 33](#_Toc293321587)

[5 Real Use Cases 34](#_Toc293321588)

[5.1 Beschreibung Interaktionsdiagramm Authentifizierung 34](#_Toc293321589)

[5.2 Beschreibung Interaktionsdiagramm Stundeneintrag erfassen 34](#_Toc293321590)

[5.3 Interaktionsdiagramm Authentifizierung auf Android Client 35](#_Toc293321591)

[5.4 Interaktionsdiagramm Stundeneintrag erfassen 36](#_Toc293321592)

[6 Prozesse und Threads 37](#_Toc293321593)

[6.1 Server 37](#_Toc293321594)

[6.2 Client 37](#_Toc293321595)

[6.2.1 Activity 37](#_Toc293321596)

[6.2.2 Synchronisationsservice 37](#_Toc293321597)

[7 Datenspeicherung 38](#_Toc293321598)

[7.1 Persistenz Rails 38](#_Toc293321599)

[7.2 Persistenz Android 38](#_Toc293321600)

[8 Grössen und Leistung 39](#_Toc293321601)

[8.1 Server 39](#_Toc293321602)

[8.2 Client 39](#_Toc293321603)

[9 Anhang 40](#_Toc293321604)

[9.1 Architekturentscheide 40](#_Toc293321605)

[9.1.1 Datenspeicherung / ORM 40](#_Toc293321606)

[9.2 Auszug Fehlerbericht per Mail von Rails 41](#_Toc293321607)

## Abbildungsverzeichnis

[Abbildung 1 - Architekturübersicht 7](file:///E:\hsr\se2p_svn.elmermx.ch\doc\05_Design\Software%20Architektur.docx#_Toc293321633)

[Abbildung 2 - Sequenzdiagramm Übertragung TimeEntry 8](file:///E:\hsr\se2p_svn.elmermx.ch\doc\05_Design\Software%20Architektur.docx#_Toc293321634)

[Abbildung 3 - Zustandsdiagramm Übertragung TimeEntry 9](#_Toc293321635)

[Abbildung 4 - Deployment Diagram MRT 10](#_Toc293321636)

[Abbildung 5 - Ablauf eines Requests, Quelle: http://gmoeck.github.com/2011/03/10/sproutcore-mvc-vs-rails-mvc.html 12](#_Toc293321637)

[Abbildung 6 - Übersicht über die Packages 14](#_Toc293321638)

[Abbildung 7 - Aufbau Apache Erweiterung Passenger, Quelle: http://www.modrails.com/documentation/Architectural%20overview.html 16](file:///E:\hsr\se2p_svn.elmermx.ch\doc\05_Design\Software%20Architektur.docx#_Toc293321639)

[Abbildung 8 - Allgemeine Fehlermeldung Rails-Server 17](file:///E:\hsr\se2p_svn.elmermx.ch\doc\05_Design\Software%20Architektur.docx#_Toc293321640)

[Abbildung 9 - Logeintrag unter Rails 17](file:///E:\hsr\se2p_svn.elmermx.ch\doc\05_Design\Software%20Architektur.docx#_Toc293321641)

[Abbildung 10 - Fehlerbericht von Ruby-Server 18](file:///E:\hsr\se2p_svn.elmermx.ch\doc\05_Design\Software%20Architektur.docx#_Toc293321642)

[Abbildung 11 - Null Object Pattern für LocationListener 19](file:///E:\hsr\se2p_svn.elmermx.ch\doc\05_Design\Software%20Architektur.docx#_Toc293321643)

[Abbildung 12 - Interface Transmittable 20](file:///E:\hsr\se2p_svn.elmermx.ch\doc\05_Design\Software%20Architektur.docx#_Toc293321644)

[Abbildung 13 - Verwendung Interface Transmittable 20](#_Toc293321645)

[Abbildung 14 - Try Catch Konstrukt (Error Handling) 21](#_Toc293321646)

[Abbildung 15 - Logeinträge anzeigen mit LogCat 22](file:///E:\hsr\se2p_svn.elmermx.ch\doc\05_Design\Software%20Architektur.docx#_Toc293321647)

[Abbildung 16 – Klassenstruktur Package interfaces 23](#_Toc293321648)

[Abbildung 17 - Klassenstruktur Package network 24](#_Toc293321649)

[Abbildung 18 - Klassenstruktur Package models 26](#_Toc293321650)

[Abbildung 19 - Klassenstruktur Package database 27](#_Toc293321651)

[Abbildung 20 - Klassenstruktur Package application 28](#_Toc293321652)

[Abbildung 21 - Klassenstruktur Package services 29](#_Toc293321653)

[Abbildung 22 - Insteraktionsdiagramm Synchronisation 31](#_Toc293321654)

[Abbildung 23 - Klassenstruktur Package activities 33](#_Toc293321655)

[Abbildung 24 - Sequenzdiagramm Authentifizierung 35](#_Toc293321656)

[Abbildung 25 - Interaktionsdiagramm Stundeneintrag erfassen 36](#_Toc293321657)

# Einführung

## Zweck

Dieses Dokument beschreibt die Software Architektur für das Projekt MRT (Mobile Reporting Tool).

## Gültigkeitsbereich

Dieses Dokument ist während der gesamten Projektdauer gültig (21.02 bis 03.06.2011).

## Definitionen und Abkürzungen

Die Definitionen und Abkürzungen befinden sich in der ausgelagerten Datei doc/01\_Projektplan/glossar.docx.

## Referenzen

* doc/03\_Anforderungsspezifikationen/\*
* doc/04\_Domainanalyse/\*
* doc/05\_Design/\*
* doc/templates/template.dotx
* doc/templates/java\_formatting.xml
* doc/templates/ruby\_formatting\_settings.zip
* doc/media/logo.png
* **<<JAVADOC>>**

# Software Architektur

## Architektonische Ziele & Einschränkungen

### Ziele

* Es soll möglichst einfach möglich sein, den Android Client durch einen anderen Client (z.B. iPhone, Windows 7 Phone) zu ersetzen. Deshalb darf die Architektur keine Android-spezifischen Konzepte enthalten.
* Die Benutzeroberfläche des Android Clients soll sehr einfach und übersichtlich gestaltet werden. Deshalb stellt der Android Client nur die wichtigsten Funktionalitäten zur Verfügung.
* Mehrere Mitarbeiter müssen gleichzeitig Stundeneinträge machen können. Darum müssen die einzelnen Einträge voneinander getrennt werden.
* Die Daten müssen konsistent bleiben, auch wenn die Datenverbindung zwischen Client und Server unterbrochen wird. Das soll gezielt durch geplante Abläufe erreicht werden.
* Die Schnittstellen zwischen Client und Server sollen genau definiert werden, damit die zwei Teile unabhängig voneinander entwickelt werden könnten.

### Einschränkungen

* Der Android Client wird über den Touchscreen bedient. Deshalb muss darauf geachtet werden, dass beispielsweise die Buttons gross genug sind.
* Da für die Entwicklung des Prototyps nur wenig Zeit zur Verfügung steht und es in diesem Projekt nicht primär um das grafische Erscheinungsbild geht, wird dieses nicht ausgereift sein.

### Tools

Die folgenden Tools kamen bei diesem Projekt zum Einsatz:

* **JavaDoc:** Das Software-Dokumentantioswerkzeug dient dem automatischen Erstellen von HTML-Dokumentationsdateien aus Java-Quelltextdateien. Javadoc ist fester Bestandteil des Java Development Kits.
* **RubyDoc:** Durch dieses Software-Dokumentationswerkzeug können ebenfalls HTML-Dokumentationsdateien erstellt werden. Jedoch hier von den Ruby-Quelltextdateien.
* **Enterprise Architect:** Durch dieses Programm können verschiedene UML-Diagramme erstellt werden, beispielsweise Klassen- oder Sequenzdiagramme.
* **ObjectAid:** Hierbei handelt es sich um ein UML Plugin für Eclipse. Damit können UML-Diagramme direkt aus dem Source Code generiert werden. Es lassen sich damit wiederum Klassen- und Sequenzdiagramme erstellen.
* **State Of Flow Eclipse Metrics Plugin:** Dieses Plugin berechnet verschiedene Metriken für den Code und zeigt Warnungen an, sobald gewisse Werte überschritten oder Regeln verletzt wurden. Dieses Tool dient der ständigen Überwachung des Code-Zustands. Metriken können zudem als HTML-Dateien exportiert und somit genauer analysiert werden.
* **Crap4J:** Hierbei handelt es sich ebenfalls um ein Eclipse Plugin welches die CRAP (Change Risk Analysis and Predictions) Software Metriken testet. Carp4J kombiniert „Cyclomatic Complexity“ und die Überprüfung der Codeabdeckung bei automatischen Tests (JUnit Tests).  
  „Cyclomatic Complexity“ ist eine Software Metrik welche von Sir Thomas J. McCabe im Jahr 1976 entwickelt wurde. Sie wird genutzt um die Komplexität eines Programms anzugeben. Dabei misst es die Anzahl an linear unabhängigen Pfaden durch den Source Code.

## Architekturübersicht

Wie anhand der nachstehenden Abbildung ersichtlich ist, besteht die Architektur aus einem Client-Server System. Als Clients werden Computer sowie Smartphones mit dem Android Betriebssystem eingesetzt. Die Android Mobiltelefone verwenden eine eigene, lightweight Datenbank, welche verwendet wird, um Kunden, Stundeneintragstypen und Stundeneinträge zwischen zu speichern.



Abbildung - Architekturübersicht

## Kernarchitektur Übertragung Zeiteintrag

### Sequenzdiagramm

Die folgende Abbildung zeigt den Ablauf beim Übertragen eines Zeiteintrags zwischen dem Client (Android) und dem Server.

Abbildung 2 - Sequenzdiagramm Übertragung TimeEntry



Sequenzdiagramm Übertragung TimeEntry

### Zustandsdiagramm

Das Zustandsdiagramm zeigt den Ablauf beim Übertragen eines Zeiteintrags mit allen Übergängen und Aktionen.



Abbildung - Zustandsdiagramm Übertragung TimeEntry

## Systemstruktur

### Physische Sicht



Abbildung - Deployment Diagram MRT

Da der Server die Rails Applikation hostet, wird er als Server Execution Node dargestellt. Der Client verbindet sich mit dem Server mit Hilfe des Protokolls HTTP/HTTPS. Deshalb wird der Client als Client Execution Node dargestellt.

Zu beachten ist, dass auf dem Server und dem Client je eine andere Applikation läuft. Auf dem Server ist dies eine Rails Applikation (rails\_app), auf dem Client eine Android Applikation (mrt.apk).

Server und Client kommunizieren über HTTP/HTTPS. Natürlich läuft HTTP/HTTPS über TCP/IP. Eine klare Abgrenzung ist hier wichtig, demzufolge ist für das Projekt die Schicht HTTP/HTTPS die tiefste Schicht.

Eine ausführliche Begründung für den Architekturentscheid ist im Anhang 11.1.1 Starker Server / mobiler Client zu finden. Am selbigen Ort wird auch begründet, weshalb zur Serialisierung der Daten JSON verwendet wird.

### Logische Sicht Rails

Die Architektur der Rails Applikation ist zu einem grossen Teil vorgegeben. Diese wird nachfolgend beschrieben.

Rails setzt sich aus nachfolgenden individuellen Komponenten zusammen:

#### Action Pack

Das Action Pack besteht aus drei Komponenten, dem Action Controller, der Action View und dem Action Dispatch. Es ist der View Controller - Teil des Architekturmusters “MVC”.

##### Action Controller

Der Action Controller ist die Komponente, welche die Controller in einer Rails-Applikation steuert. Das Framework leitet eingehende Anfragen an die Applikation weiter, extrahiert die Parameter und verteilt diese an die zuständige Aktion. Der Service umfasst das Sitzungsmanagement, das Rendering von Templates und das Weiterleitungsmanagement.

##### Action View

Die Action View verwaltet die Ansichten der Rails-Applikation. Standardmässig kann HTML und XML Output generiert werden. Die Action View beinhaltet das Rendering von Templates, inklusive verschachtelten und Teil-Templates. Zudem schliesst es einen eingebauten AJAX Support ein.

##### Action Dispatch

Der Action Dispatch führt das Routing von Web-Anfragen aus und verteilt diese, je nach Wunsch, an die eigene oder eine andere Rack-Applikation.

#### Action Mailer

Der Action Mailer ist das Framework für einen eingebauten E-Mail Services. Er kann benutzt werden, um eingehende Nachrichten zu empfangen und zu verarbeiten. Einfache Textmeldungen oder komplexe, mehrteilige Emails, auf flexiblen Templates basierend, können versandt werden.

#### Active Model

Das Active Model stellt ein vordefiniertes Interface bereit, welches zwischen dem Service des Action Packs und Object Relationship Mapping-Komponenten wie Active Record fungiert. Es erlaubt Rails, andere ORM Frameworks anstelle des Active Records zu verwenden, sofern dies für die Applikation nötig ist.

#### Active Record

Der Active Record ist die Grundlage der Models einer Rails-Applikation. Er stellt Datenbank-Unabhängikeit, CRUD Funktionalitäten, fortgeschrittene Suchfähigkeiten und die Möglichkeit, Models miteinander oder anderen Services zu verknüpfen, bereit.

#### Active Resource

Die Active Resource bietet ein Framwork zur Verwaltung der Verbindung zwischen Business Objekten und RESTful Web Services. Es implementiert eine Methode, um webbasierte Ressourcen auf lokale Objekte mit CRUD Semantik abbilden zu können.

#### Active Support

Der Active Support ist eine umfangreiche Sammlung an Werkzeugklassen und Standard-Bibliothekserweiterungen von Ruby, welche in Rails, im Kerncode und den Applikationen, benutzt werden.

#### Railties

Railties ist der Kerncode von Rails, welcher neue Rails-Applikationen erzeugt und verschiedene Frameworks und Plugins in einer Rails-Applikation vereint.

#### Rails MVC

##### Models

Ein Model verkörpert die Informationen/Daten einer Applikation und die Regeln, um diese zu manipulieren. Bei Rails werden die Models vor allem verwendet, um die Interaktion mit einer dazugehörigen Datenbank Tabelle zu regeln. In den meisten Fällen wird sich eine Tabelle auf genau ein Model beziehen. Der Grossteil der Business Logik der Applikation befindet sich in den Models.

##### Views

Views stellen das User Interface einer Applikation dar. Bei Rails handelt es sich hierbei meist um HTML Dateien mit eingebettetem Ruby Code. Diese kümmern sich nur um die Präsentation der Daten. Views übergeben die Daten dem Webbrowser oder einem anderen Programm, welches Anfragen an die Applikation sendet.

##### Controllers

Die Controller könnten als Klebstoff zwischen Models und Views bezeichnet werden. Controllers sind in Rails dafür verantwortlich, die eingehenden Anfragen vom Webbrowser weiterzuleiten, von den Models Daten abzufragen und Daten an die Views weiterzuleiten.



Abbildung - Ablauf eines Requests, Quelle: http://gmoeck.github.com/2011/03/10/sproutcore-mvc-vs-rails-mvc.html

##### Schritte

1. Der Browser sendet eine Anfrage an den Webserver.
2. Der Webserver verarbeitet die Anfrage, ermittelt die entsprechende Route und leitet die Anfrage an die dazugehörige Controller Methode weiter.
3. Der Controller erfragt danach all die benötigten Informationen vom Model Layer, die für den aktuellen Request benötigt werden.
4. Der Model Layer sammelt alle Informationen und gibt diese dem Controller zurück.
5. Der Controller gibt die evaluierten Informationen der View weiter und fordert sie auf, diese zu übersetzen.
6. Die View rendert sich selbst und übermittelt das übersetzte HTML dem Controller.
7. Der Controller erstellt daraus das gesamte HTML (inkl. Layout) der angefragten Seite und übergibt sie dem Webserver.
8. Der Webserver gibt die Seite an den Browser zurück.

### Logische Sicht Android

Um den Code zu strukturieren wurden die Klassen auf Packages aufgeteilt. Damit die einzelnen Packages einzeln entwickelt werden können, müssen die Aufgaben und Abhängigkeiten klar definiert sein. In der folgenden Grafik sind die Packages und ihre Abhängigkeiten ersichtlich.

Die einzelnen Dependencies werden im nachfolgenden Unterkapitel 3.5 Schnittstellen der Packages genauer beschrieben.



Abbildung - Übersicht über die Packages

Wie in der Grafik zu sehen ist, sind die Packages nur nach unten abhängig.

#### Interfaces

„interfaces“ beinhaltet nur einzelne Interfaces. Es dient zur Entkopplung von den Packages „network“ und „models“.

#### Network

„network“ ist dafür verantwortlich, dass die Daten zwischen Rails Server und Android Client übertragen werden.

#### Models

„models“ stellt die Klassen bereit, um die Daten persistent zu speichern oder zu laden. Zum Beispiel findet sich in diesem Package der Kunde oder der Stundeneintrag. Zusätzlich enthält dieses Package Helpers, die bei der Speicherung / beim Laden gebraucht werden.

#### Database

„database“ ist dafür zuständig, dass eine Verbindung zur Datenbank hergestellt wird und das Schema erstellt wird. Zusätzlich liefert es sogenannte DAO’s (Database Access Objects), womit auf die Datenbank zugegriffen werden kann.

#### Application

„application“ wird dafür verwendet, um den aktuellen Zustand der Applikation zu managen. Dazu gehören Dinge wie die aktuelle Systemkonfiguration oder gespeicherte Logindaten.

#### Gui/gen

„gui/gen“ wird vom Android generiert. Durch dieses Package können die User Interfaces Angaben, welche in separaten XML-Dateien gespeichert sind, identifiziert werden.

#### Services

„services“ ist dafür verantwortlich, dass Daten vom Server auf den Client und in die entgegengesetzte Richtung, also vom Client auf den Server, synchronisiert werden.

#### Activities

Dieses Package ist die Schnittstelle zum Benutzers. In den „activities“ wird die Problem Domain abgehandelt. „activities“ nehmen Benutzerbefehle entgegen, bearbeiten „models“ und starten „services“.

## Schnittstellen der Packages



Abbildung - Übersicht Packages

Abhängigkeiten der einzelnen Packages:

* **Interfaces:** Auf das Package „interface“ sind die Packages „network“, „services“ und „models“ angewiesen.
* **Network:** Vom Package „network“ sind die Packages „activities“, „services“, „application“ und „database“ abhängig.
* **Models:** Die Packages „activites“, „services“, „application“ und „database“ sind an das Package „models“ gebunden.
* **Database:** Von dem Package „database“ sind die Packages „services“ und „activities“ abhängig.
* Application: Auf das Package „application“ sind ebenfalls die Packages „services“ und „activities“ angewiesen.
* **Gui/gen:** Auf das Package „gui/gen“ ist ebenfalls lediglich „activites“ angewiesen (in Übersicht nicht vorhanden).
* **Services:** „services“ befinden sich auf der obersten Ebene, daher ist kein anderes Package davon abhängig.
* **Activities:** Das Package „activities“ befindet sich ebenfalls auf der obersten Ebene.

## Architekturkonzepte

### Rails

#### Angewandte Patterns

Bei den folgenden drei Patterns handelt es sich um die wichtigsten Patterns im den Server-Teil. Die meisten dieser Patterns sind bereits in das Rails Framework / Rails Server eingebaut.

##### Active Record

Rails verwendet als OR Mapper eine gleichnamige Implementation des Active Record Design Patterns. Mehr zu Active Record im Abschnitt 3.4.2.4 „Active Record“

##### Lazy Loading

Bei Ruby on Rails wird zum Beispiel bei der Datenbankabfrage das Lazy-Loading Prinzip verwendet, um die Performance zu steigern. Zum Beispiel können die Kunden folgendermassen abgefragt werden:

@customers = Customer.where(:valid\_until => nil)

Diese Anweisung lädt die Kunden aber noch nicht! Somit ist es möglich, im Nachhinein die Abfrage noch weiter zu verändern:

@sorted\_customers = @customers.order(:last\_name)

Erst wenn die Iteration über die Kunden tatsächlich startet, wird die Abfrage ausgelöst und die einzelnen Datensätze geladen:

@sorted\_customers.each do |customer|  
 # Do some work  
end

##### Thread-Pool

Damit mehrere Anfragen an den Webserver gleichzeitig gestellt werden können, wird ein spezielles zusätzliches Modul Passenger für den Apache-Server verwendet. Diese Komponente ermöglicht es, mittels eines Thread Pools mehrere Anfragen entgegenzunehmen und somit mehrere Clients zur selben Zeit zu bedienen. Dies ist für das konzipierte System unumgänglich, da es gut möglich ist, dass mehrere Aussendienstmitarbeiter den Server gleichzeitig beanspruchen.



Abbildung - Aufbau Apache Erweiterung Passenger, Quelle: http://www.modrails.com/documentation/Architectural%20overview.html

##### MVC

Ruby on Rails ist auf dem Model-View-Controller Prinzip aufgebaut. Für eine ausführlichere Erklärung zu diesem Architekturprinzip wird auf das Kapitel 3.4.2.8 Rails MVC verwiesen.

#### Errorhandling

Beim Errorhandling wird zwischen der Entwickler und der Benutzersicht unterschieden. Für beide Sichten werden bei einem Fehlerfall die aufgetretenen Fehler in Logfiles protokolliert. Die Logfiles befinden sich im Stammverzeichnis des Rails Server im Ordner log.

##### Fehler aus Benutzersicht

Falls während dem Betrieb des Servers ein Fehler in einem der Module auftaucht, wird die Anfrage des Client abgebrochen. Wenn es sich um eine Anfrage handelt, welche nicht vom lokalen Server aus gesendet wurde, wird als Antwort HTML Code zurückgeschickt, der bei dem Client eine allgemeine Fehlermeldung (siehe Abbildung 14 - Allgemeine Fehlermeldung Rails-Server) anzeigt. Diese Fehlermeldung enthält nur einen Hinweis an den Client, dass ein Fehler aufgetreten ist, jedoch werden keine Fehlerdetails ersichtlich.



Abbildung - Allgemeine Fehlermeldung Rails-Server

Zudem wird im Hintergrund ein Mail mit dem Fehlerbericht an das Entwicklungsteam versandt. Dieser Schritt wird ohne einen Eingriff vom Benutzer selbständig durchgeführt.

##### Fehler aus Entwicklersicht

Da der Entwickler den Server nicht live überwacht, werden vom Server Logfiles erzeugt, damit der Entwickler alle aufgetretenen Fehler nachvollziehen kann. Wie das genau funktioniert wird im nächsten Abschnitt beschrieben.

#### Logging & Debugging

##### Logeinträge

Rails hält sämtliche Requests, Fehler etc. in Logdateien fest. Es existieren verschiedene Typen von Logs: Development (Entwicklung), Production (Produktion) und Test. Diese Unterscheidung dient zur einfacheren, überschaubaren Nachverfolgung von Fehlern und Abläufen in den verschiedenen Umgebungen.

Die nachstehende Abbildung visualisiert einen Ausschnitt eines Logfiles.



Abbildung - Logeintrag unter Rails

##### Fehlerbericht Entwicklungssicht

Damit die Entwicklersicht aktiviert ist, muss der Server im Debugging Modus gestartet werden. In Rails wird das durch sogenannte Environments, also Umgebungen, gemacht.

Um ausführlichere Informationen über den aufgetretenen Fehler zu erhalten, wird die fehlerhafte Anfrage lokal vom Server aus versendet. Als Antwort wird eine detaillierte Fehlerbeschreibung in Form einer HTML Datei zurückgeschickt. Die ursprüngliche Anfrage wird zudem abgebrochen.

Eine Fehlerbeschreibung enthält die Fehlerbezeichnung, den Controllername sowie die Aktion im Titel. Weiter ist der Ablauf bis zum Auftreten des Fehlers anhand der Traceanzeige ersichtlich. Diese Anzeige macht ersichtlich, welche Zeile den Fehler verursachte. Für eine ausführlichere Traceanzeige gibt es zudem noch eine vollständige Traceanzeige (Full Trace). Zwei weitere Kategorien für die Requestparameter sowie die allfälligen Responsewerte sind auch noch vorhanden.

Die nachfolgende Abbildung zeigt einen Ausschnitt eines Fehlerberichts, bei dem auf der Zeile 15 in der Aktion unassigned ein Fehler aufgetreten ist.



Abbildung - Fehlerbericht von Ruby-Server

##### Fehlerbericht Produktionssicht

Falls eine externe Anfrage (Produktionssicht) einen Fehler verursacht wird ein umfassender Fehlerbericht an eine definierte Mailadresse geschickt. Für den kompletten Auszug eines Fehlerberichts wird auf den Anhang verwiesen (siehe 9.2 Auszug Fehlerbericht per Mail von Rails).

#### Speicherverwaltung

Wie bei Java wird bei der Programmiersprache Ruby auf eine automatische Speicherbereinigung gesetzt. Dies wird durch einen Garbage Collector ausgeführt, welcher sämtliche nicht referenzierte Objekte aus dem Arbeitsspeicher entfernt.

### Android

#### Angewandte Patterns

In den folgenden Kapiteln werden nur die wichtigsten angewandten Patterns beschrieben. Weitere Lösungsmuster sind im Code oder im Framework enthalten, werden aber hier nicht speziell beschrieben.

##### Null Object

Das Null Object Pattern wird für die Implementation des Interface LocationListener eingesetzt, da keine Funktionalität bei einer Änderung des Standorts benötigt wird.

Sollte doch eine Methode benötigt werden, könnte die entsprechende Methode überschrieben werden.

* Übergabe Null-Object bei Methode requestLocationUpdates(.., new LocationListener());



Abbildung - Null Object Pattern für LocationListener

##### Mock-Objects

Damit die Tests unabhängig und schneller ablaufen und nicht durch langsame Komponenten (Kollaborationen) behindert werden, mussten verschiedene Mock-Objekte erstellt werden.

Dies wurde speziell bei der Anbindung an die Datenbank sowie die Netzwerkverbindung über http durchgeführt. Die folgende Liste zeigt alle verwendeten Mock-Objekte.

* DatabaseHelperMock (Package ch.hsr.se2p.mrt.database.mocks)
* MockHttpClient (Package ch.hsr.se2p.mrt.network.mocks)
* MockHttpEntity (Package ch.hsr.se2p.mrt.network.mocks)
* MockHttpResponse (Package ch.hsr.se2p.mrt.network.mocks)

##### Polymorphismus

Damit die Anwendung in Zukunft einfach erweitert werden kann, wurde an verschiedenen Stellen auf Polymorphismus gesetzt. Dadurch gestaltet sich das System generischer und zudem auch unabhängiger von der konkreten Implementation.

Bei den nachstehenden Schnittstellen handelt es sich um die wichtigsten, welche am meisten verwendet werden. Weitere Details können den Package- / Klassenbeschreibungen entnommen werden (siehe 4 Beschreibung der Packages).

* Transmittable (Package ch.hsr.se2p.mrt.interfaces)
* Receivable (Package ch.hsr.se2p.mrt.interfaces)
* Confirmable (Package ch.hsr.se2p.mrt.interfaces)
* Synchronizer (Package ch.hsr.se2p.mrt.services)

**Konkretes Beispiel**

Das in diesem Abschnitt geschilderte Beispiel macht ersichtlich, wie Polymorphismus in der Applikation eingesetzt wird. Die Schnittstelle „Transmittable“ definiert die generell notwendigen Methoden für ein Objekt, welches Übertragbar sein soll. Sämtliche Klassen, welche übertragbar sind, implementieren das Interface „Transmittable“.



Abbildung - Interface Transmittable

In der untenstehenden Abbildung (Abbildung 13 - Verwendung Interface Transmittable) wird anhand der Methode transmit aus der Klasse TimeEntryHelper ein Objekt übergeben welches übertragbar ist. Dies verdeutlicht den Polymorphismus, da anstelle einer konkreten Implementation eine implementationsunabhängige Schnittstelle als Parameter definiert wurde.



Abbildung - Verwendung Interface Transmittable

#### Errorhandling

Falls ein Fehler während der Programmausführung auftaucht, wird mittels eines Try, catch Konstrukt der Fehler abgefangen und der Fehler aufgezeichnet. In der untenstehenden Abbildung wird ein typisches Errorhandling durchgeführt.



Abbildung - Try Catch Konstrukt (Error Handling)

Dem Benutzer werden in der Regel Fehler nicht angezeigt, stattdessen werden Fehler mit der im Logging System „LogCat“ von Android erfasst.

Da der Benutzer von den Fehlern nichts mitbekommen sollte, darf ein Fehler den Systemzustand grundsätzlich nicht ändern. Wird z.B. die Internetverbindung während der Übertragung von Daten unterbrochen, so muss der Vorgang wiederholt werden, bis die Übertragung erfolgreich beendet.

#### Logging & Debugging

Um Informationen über den Ablauf des Systems zu erhalten, werden an verschiedenen bedeutenden Stellen im Code Log-Meldungen geschrieben. Dies dient in erster Linie dazu, Fehler oder Abläufe zu reproduzieren und Einsicht in das System zu erhalten.

Dazu werden die nachfolgenden fünf Kategorien von „Logging“ Meldungen erkannt.

##### Logging-Levels

**Verbose**

Diese Kategorie bietet nebensächliche beiläufige Informationen, die nur für erweiterte Debugging Zwecke verwendet werden.

* Verbose erfassen: Log.v(…)

**Debug**

Die Debugmeldungen dienen speziell der Nachverfolgung von einzelnen Schritten. Da die Meldungen in dieser Kategorie detaillierte Informationen über den Ablauf liefern und deshalb speziell bei der Entwicklung eingesetzt werden.

* Debug erfassen: Log.d(…)

**Info**

Unter dieser Kategorie werden typischerweise alle Meldungen zu wichtigen Schlüsselstellen der Applikation angezeigt. Häufig werden solche Einträge erfasst, falls die Anwendung eine neue Aktivität, neuen Task etc. beginnt oder beendet.

Auch diese Kategorie kann gut für das Debugging und Management verwendet werden, da nur noch die wichtigsten Nachrichten angezeigt werden.

* Info erfassen: Log.i(…)

**Warn**

Warnungen werden in der Regel verwendet um Hinweise zu geben, wie z.B. dass die Standardkonfigurationen verwendet werden, da keine Konfigurationsdatei gefunden wurde, obwohl die Konfigurationsdatei vorhanden sein müsste. Zum Teil werden auch behandelte Fehler als Warnungen deklariert, dies kommt auch auf den Schweregrad des Fehlers an.

* Warnung erfassen: Log.w(…)

**Error**

Kritische Fehler, die grundsätzlich nicht auftreten sollten, werden als Error geloggt.

* Error erfassen: Log.e(…)

##### Ablauf Logeintrag erfassen

Jede Klasse besitzt eine eindeutige Identifikation in Form eines konstanten String (Klassenname) der die Klasse repräsentiert. Anhand dieses „Tags“ kann später im Loggingsystem erkannt werden, von welcher Klasse die Meldung ursprünglich kommt.

Der nachfolgende Codeausschnitt illustriert den zur Identifikation verwendeten TAG.

Um einen Eintrag im LogCat zu erfassen, wird je nach Kategorie der Meldung eine von fünf möglichen Methoden gewählt (siehe Logging-Levels).

Im folgenden Codebeispiel wird ein Fehler (Error) im Loggingsystem erfasst. Dazu wird die passende Methode e(…) der Klasse Log mit dem Klassenidentifikator TAG, einer Fehlerbeschreibung sowie der geworfenen Exception übergeben.

##### Logeinträge anzeigen in Eclipse

Um die Logeinträge in Eclipse anzuzeigen, existiert die Ansicht LogCat unter der Perspektive DDMS in Android. Diese Anzeige ist sehr hilfreich, um sämtliche Einträge auf einen Blick darzustellen. Zudem kann man zwischen den fünf Kategorien (siehe vorhergehendes Kapitel) unterscheiden, sowie mittels eines Filters eine spezifische Meldung suchen.



Abbildung - Logeinträge anzeigen mit LogCat

Logging-Level Kategorien

#### Speicherverwaltung

Da das Android-System auf Java basiert, wird zur internen Speicherverwaltung der Garbage Collector verwendet. Mit dieser Java-Komponente werden nicht mehr benötigte Objekte im Arbeitsspeicher freigegeben, sobald keine Referenzen mehr auf diese Objekte bestehen.

# Beschreibung der Packages

Die Packages werden in den folgenden Kapiteln genauer umschrieben und aufgezeichnet. Bei vorhandenem Architekturkonzept wird auf dieses näher eingegangen. Wichtige Klassenspezifikationen werden in diesem Dokument aufgelistet und erläutert, für die übrigen wird auf das JavaDoc verwiesen (siehe 2.4 Referenzen).

**TODO**: Besserer Verweis auf JavaDoc

## Interfaces

### Beschreibung

Das Package „interfaces“ dient als Schnittstelle zwischen Server und Client. Die verschiedenen Interfaces stellen Methoden zur Verfügung, welche die erfolgreiche Datenübertragung sicherstellen. Die Interfaces werden von verschiedenen Klassen des Packages „models“ implementiert. Verwendet werden sie von den Klassen im Package „network“, um die Daten zu übermitteln. Die Beschreibung der abstrakten Funktionalität vereinfacht es wesentlich, weitere Klassen dem Package „models“ hinzuzufügen.

### Klassenstruktur



Abbildung – Klassenstruktur Package interfaces

### Architekturkonzepte für Package (falls vorhanden)

Die Interfaces beruhen auf dem Konzept des Polymorphismus welches in Kapitel 3.6.2.1.3 Polymorphismus genauer erklärt wird.

### Klassenspezifikationen

#### Confirmable

Das Interface Confirmable dient dazu, einen Stundeneintrag auf dem Server zu bestätigen. Dadurch wird sichergestellt, dass die Objekte erfolgreich an den Server übertragen wurden und diese auf dem Client entfernt werden können.

|  |  |
| --- | --- |
| Methode | Beschreibung |
| getIdOnServer() | Gibt die ID zurück, welche vom Server gesetzt wurde. |
| processConfirmation(JSONObject jsonObject) | Gibt zurück, ob die vom Server gesetzte ID mit der des erhaltenen JSONObjects übereinstimmt. |

#### Receivable

Das Interfaces Receivable stellt Methoden zur Verfügung um die Übertragung von Elementen vom Server an den Client zu ermöglichen.

|  |  |
| --- | --- |
| Methode | Beschreibung |
| getIdOnServer() | Gibt die ID zurück, welche vom Server gesetzt wurde. |
| getUpdatedAt() | Gibt den Zeitpunkt der letzten Änderung in Form eines Timestamps zurück. |
| fromJSON(JSONObject jsonObject) | Dem erhaltenen JSONObject können die Attribute der implementierenden Klasse entnommen werden. Die Methode returniert, ob dies erfolgreich verlaufen ist. |

#### Transmittable

Das Interfaces Transmittable ermöglicht es den implementierenden Klassen Objekte an den Server zu schicken. Zudem wird sichergestellt, dass Objekte nicht mehr als einmal auf dem Server gespeichert werden.

|  |  |
| --- | --- |
| Methode | Beschreibung |
| toJSONObject() | Erstellt ein neues JSONObject und setzt dort die jeweiligen Attribute der implementierenden Klasse. Danach wird dieses Objekt zurückgegeben. |
| isTransmitted() | Gibt zurück ob Objekt schon übermittelt wurde oder nicht. |
| processTransmission(JSONObject jsonObject) | Durch das erhaltene JSONObject wird der Hashcode verglichen und die ID gesetzt. Stimmt der Hashcode auf Server und Client Seite überein, wird dieser danach serverseitig entfernt. Es wird zurückgegeben, ob dies erfolgreich war. |

## Network

### Beschreibung

Das Package „network“ ist dafür zuständig, Daten zu übertragen. Es wird vom Package „services“ genutzt. Bei den zu übertragenden Daten handelt es sich um die Daten, die durch die Klassen des Packages „models“ gespeichert werden.

### Klassenstruktur



Abbildung - Klassenstruktur Package network

### Architekturkonzepte für Package (falls vorhanden)

<• Umsetzung der generellen Architekturkonzepte

• Spezifische Architekturkonzepte für Package>

### Klassenspezifikationen

<• nur wo sinnvoll, insbesondere Schnittstellenklassen

o mit wesentlichen Attributen und Methoden (meist als Javadoc)

o Eventuell Datentypen für Attribute, Parameter und Rückgabewerte mit Datentypen für

Methoden, eventuell Zustandsdiagramm für Klasse (falls sinnvoll)>

### Interaktionen innerhalb Package (falls sinnvoll)

## Models

### Beschreibung

Im Package „models“ befinden sich alle Klassen, die dazu dienen, Daten persistent zu speichern. Dabei handelt es sich einerseits um verschiedene Daten, welche der Client von der Server-Seite benötigt, anderseits um Daten, die an den Server übertragen werden müssen.

#### Ausbaufähigkeit

In einigen Klassen befinden sich Methoden die zum jetzigen Zeitpunkt nicht verwendet werden. Da das Projekt jedoch weiter ausbaufähig ist, wurden diese nicht entfernt.

### Klassenstruktur



Abbildung - Klassenstruktur Package models

### Architekturkonzepte für Package (falls vorhanden)

<• Umsetzung der generellen Architekturkonzepte

• Spezifische Architekturkonzepte für Package>

### Klassenspezifikationen

<• nur wo sinnvoll, insbesondere Schnittstellenklassen

o mit wesentlichen Attributen und Methoden (meist als Javadoc)

o Eventuell Datentypen für Attribute, Parameter und Rückgabewerte mit Datentypen für

Methoden, eventuell Zustandsdiagramm für Klasse (falls sinnvoll)>

Die Klassen Customer, GpsPosition, TimeEntryType sowie User implementieren alle das Interface Receivable. Die damit implementierten Methoden werden im Abschnitt 4.1.4.2 Receivable erläutert. Die Klasse TimeEntry implementiert die Interfaces Confirmable (siehe 4.1.4.1 Confirmable) und Transmittable (siehe 4.1.4.3 Transmittable).

### Interaktionen innerhalb Package (falls sinnvoll)

## Database

### Beschreibung

Das Package „database“ ist für die Verbindung zur Datenbank verantwortlich. Zudem ist es für die Erstellung der Datenbank und deren Tabellen zuständig. Durch die Klasse DatabaseHelper können die verschiedenen Klassen der „activities“ auf die Datenbank zugreifen. Des Weiteren kann über diese Klasse die Datenbank auch zurückgesetzt werden.

### Klassenstruktur



Abbildung - Klassenstruktur Package database

### Architekturkonzepte für Package (falls vorhanden)

<• Umsetzung der generellen Architekturkonzepte

• Spezifische Architekturkonzepte für Package>

### Klassenspezifikationen

<• nur wo sinnvoll, insbesondere Schnittstellenklassen

o mit wesentlichen Attributen und Methoden (meist als Javadoc)

o Eventuell Datentypen für Attribute, Parameter und Rückgabewerte mit Datentypen für

Methoden, eventuell Zustandsdiagramm für Klasse (falls sinnvoll)>

### Interaktionen innerhalb Package (falls sinnvoll)

## Application

### Beschreibung

Das Package „application“ dient der Beaufsichtigung des aktuellen Zustands der Applikation. Die schliesst die aktuelle Systemkonfiguration oder gespeicherte Logindaten ein.

### Klassenstruktur



Abbildung - Klassenstruktur Package application

### Architekturkonzepte für Package (falls vorhanden)

<• Umsetzung der generellen Architekturkonzepte

• Spezifische Architekturkonzepte für Package>

### Klassenspezifikationen

<• nur wo sinnvoll, insbesondere Schnittstellenklassen

o mit wesentlichen Attributen und Methoden (meist als Javadoc)

o Eventuell Datentypen für Attribute, Parameter und Rückgabewerte mit Datentypen für

Methoden, eventuell Zustandsdiagramm für Klasse (falls sinnvoll)>

### Interaktionen innerhalb Package (falls sinnvoll)

## Gui/gen

### Beschreibung

Das Package „gui/gen“ wird von Android automatisch generiert. Durch die darin befindliche Klasse R kann auf die in den XML-Dateien definierten User Interface Angaben zugegriffen werden. Dabei handelt es sich beispielsweise um XML-Daten über die Darstellung der verschiedenen Layouts, Listen oder Menüs.

### Klassenstruktur



Abbildung 24 - Klassenstruktur gui/gen

### Architekturkonzepte für Package (falls vorhanden)

<• Umsetzung der generellen Architekturkonzepte

• Spezifische Architekturkonzepte für Package>

### Klassenspezifikationen

<• nur wo sinnvoll, insbesondere Schnittstellenklassen

o mit wesentlichen Attributen und Methoden (meist als Javadoc)

o Eventuell Datentypen für Attribute, Parameter und Rückgabewerte mit Datentypen für

Methoden, eventuell Zustandsdiagramm für Klasse (falls sinnvoll)>

## Services

### Beschreibung

Das Package „services“ sorgt dafür, dass die Daten zwischen Server und Client synchronisiert werden. So ist sind die Services dafür verantwortlich, die Daten aus der Datenbank zu lesen, diese mithilfe des Packages „network“ zu übertragen, und schlussendlich die empfangenen Daten wieder in die Datenbank zu schreiben.

### Klassenstruktur



Abbildung - Klassenstruktur Package services

Was in diesem Bild sofort auffällt, ist die zyklische Abhängigkeit zwischen der Klasse SynchronizationService (weiter Service genannt) und SynchronizationServiceTask (weiter ServiceTask genannt). Während der Grund für die Abhängigkeit nach unten trivial ist (Service erstellt ServiceTask), braucht es für die Abhängigkeit nach oben eine Erklärung: Der ServiceTask muss bei jeder Iteration die Konfiguration neu laden, damit die richtigen Daten übertragen werden. Da für das Laden der Konfiguration dem ServiceTask die Berechtigung fehlt (Andorid spezifisch), muss die Konfiguration über den Service geladen werden.

Diese zyklische Abhängigkeit ist von aussen betrachtet aber gar nicht relevant, da der SynchronizationService die einzige Exportklasse in diesem Package darstellt.

### Architekturkonzepte für Package (falls vorhanden)

<• Umsetzung der generellen Architekturkonzepte

• Spezifische Architekturkonzepte für Package>

#### Synchronizer

Die Interfaces beruhen auf dem Konzept des Polymorphismus welches in Kapitel 3.6.2.1.3 Polymorphismus genauer erklärt wird.

### Klassenspezifikationen

<• nur wo sinnvoll, insbesondere Schnittstellenklassen

o mit wesentlichen Attributen und Methoden (meist als Javadoc)

o Eventuell Datentypen für Attribute, Parameter und Rückgabewerte mit Datentypen für

Methoden, eventuell Zustandsdiagramm für Klasse (falls sinnvoll)>

Das Interface Synchronizer wird im folgenden Kapitel näher beschrieben. Zudem werden die Klassen, welche dieses Interface implementieren, genauer umschrieben. Dies, da die implementierte Methode in jeder Klasse eine sich doch erheblich unterscheidende Aufgabe zu erfüllen hat.

#### Synchronizer

Das Interface Synchronizer wird von fünf Klassen implementiert, welche für die Synchronisation der verschiedenen, im Package „models“ gespeicherten Objekte, zuständig ist.

|  |  |
| --- | --- |
| Methode | Beschreibung |
| synchronize() | Synchronisiert für die vorgesehene Klasse jeweils die Datenbank. |

#### CustomerSynchronizer

Die Klasse CustomerSynchronizer ist dafür zuständig, dass die auf dem Server befindlichen Kundendaten auf den Client übertragen werden.

|  |  |
| --- | --- |
| Methode | Beschreibung |
| @Override  synchronize() | Synchronisiert die Kundendaten. Hierbei wird überprüft welche Daten seit der letzten Synchronisation neu erstellt, angepasst oder gelöscht wurden. Dies geschieht mit Hilfe der Klasse CustomerHelper des Packages „network“. |

#### TimeEntrySynchronizer

Die Klasse TimeEntrySynchronizer ist dafür zuständig, dass die Zeiteinträge vom Client an den Server übermittelt werden.

|  |  |
| --- | --- |
| Methode | Beschreibung |
| @Override  synchronize() | Überträgt die noch nicht übermittelten TimeEntries mit Hilfe der Klasse TimeEntryHelper des Packages „network“ an den Server. |

#### TimeEntryTypeSynchronizer

Die Klasse TimeEntryTypeSynchronizer ist dafür zuständig, dass die auf dem Server befindlichen Stundeneintragstypen-Daten auf den Client übertragen werden.

|  |  |
| --- | --- |
| Methode | Beschreibung |
| @Override  synchronize() | Synchronisiert die Stundeneintragstypen-Daten. Hierbei wird überprüft, welche Daten seit der letzten Synchronisation neu erstellt, angepasst oder gelöscht wurden. Dies geschieht mit Hilfe der Klasse TimeEntryTypeHelper des Packages „network“. |

### Interaktionen innerhalb Package

#### Übersicht Synchronisation

Die folgende Abbildung 22 - Interaktionsdiagramm Synchronisation zeigt eine Übersicht über den Synchronisationsprozess. In einem ersten Schritt wird ein SynchronizationServiceTask erstellt, welcher für das Starten der zu synchronisierenden Klassen zuständig ist. Anschliessend wird ein Timer erzeugt, der periodisch alle 30 Sekunden die Methode run() des ServiceTasks aufruft. In einer Schleife werden für die Klassen „CustomerSynchronizer“,“TimeEntrySynchronizer“ sowie „TimeEntryTypeSynchronizer“ die Methode synchronize() aufgerufen, die für die komplette Synchronisation verantwortlich ist.



Abbildung 22 - Interaktionsdiagramm Synchronisation

#### Synchronisation von Kunden

Beim Abgleichen von Kundendaten werden beim Aufruf der Methode synchronize() zuerst die benötigten Hilfsklassen instanziiert, welche für das Auslesen der Datenbank sowie für das Übertragen notwendig sind. Sobald diese Schritte durchgeführt wurden, werden die lokalen Kundendatensätze mit den Datensätzen auf dem Server verglichen. Falls neue Kundendaten auf dem Server vorhanden sind, wird die Liste mit den geänderten Kunden angepasst. Abschliessend werden die aktualisierten Kundendatensätze mittels der Methode processCustomer() in der Datenbank mutiert, entfernt oder erstellt.



Abbildung 23 – Interaktionsdiagramm Synchronisation Kunde

#### Synchronisation Kunde 2. Version (Generated by ObjectAid)

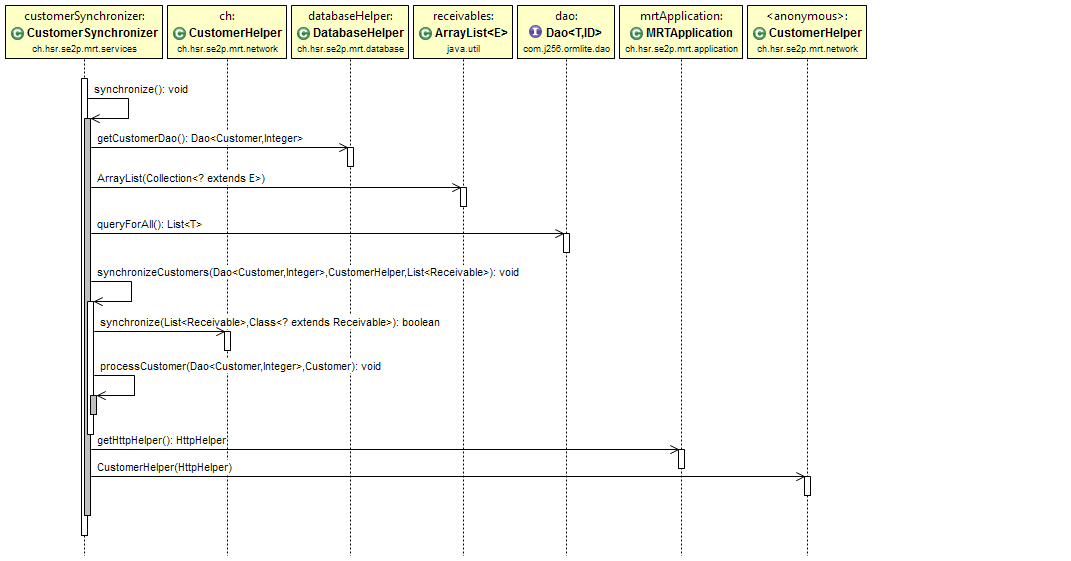


Abbildung 24 - Interaktionsdiagramm Synchronisation Kunde

#### Synchronisation von Stundeneintragstypen

Da der Ablauf beim Synchronisieren von Stundeneintragstypen dem Abgleichen von Kunden gleicht, wurde auf ein Interaktionsdiagramm verzichtet. Stattdessen wird auf das Kapitel 4.7.5.2 Synchronisation von Kunden verwiesen.

#### Synchronisation von Stundeneinträgen

Wie bei dem vorhergehenden Kapitel handelt es sich bei der Synchronisation von Stundeneinträgen um einen ähnlichen Ablauf wie bei der Synchronisation von Kunden, deshalb wird auf das Kapitel 4.7.5.2 Synchronisation von Kunden verwiesen.

## Activities

### Beschreibung

Das Package „activities“ stellt die oberste Schicht und somit die Schnittstelle zum Benutzer dar. Die „activities“ nehmen die Benutzerbefehle entgegen und leiten diese an die unteren Schichten weiter. Sie bearbeiten die „models“ und starten die verschiedenen „services“.

### Klassenstruktur



Abbildung - Klassenstruktur Package activities

### Architekturkonzepte für Package (falls vorhanden)

<• Umsetzung der generellen Architekturkonzepte

• Spezifische Architekturkonzepte für Package>

### Klassenspezifikationen

#### Null Object

Bei der Klasse LocationListenerAdapter, welche das Interface LocationLister implementiert, handelt es sich um ein den Einsatz eines Null Objects. Dies wird im Kapitel 3.6.2.1.1 Null Object genauer erläutert.

### Interaktionen innerhalb Package (falls sinnvoll)

# Real Use Cases

Für die Anwendung sind die wichtigsten und umfangreichsten Use Cases sind UC 01 Stundeneintrag erfassen sowie UC 5 Benutzer authentifizieren. Deshalb wurde der Fokus speziell auf diese zwei Use Cases gerichtet. Die weiteren Use Cases werden nicht mittels eines Interaktionsdiagramms dargestellt, da es sich lediglich um CRUD’s handelt, die keine komplexen Interaktionen zwischen Objekten benötigen.

TODO Verweise

## Beschreibung Interaktionsdiagramm Authentifizierung

Die Authentifizierung ist der erste Dialog, der für den Benutzer ersichtlich ist. Sobald der Mitarbeiter auf die Schaltfläche „Login“ drückt, wird geprüft, ob etwas eingegeben wurde. Falls die Eingabefelder nicht ausgefüllt wurden, wird eine Meldung angezeigt.

Über eine HTTP Anfrage, werden die Benutzerdaten wie: Password, Email, aktueller Benutzer übermittelt. Falls der Server eine positive Bestätigung zurückliefert, werden die Anmeldedaten gespeichert (sofern dies vom Benutzer gewünscht wird). Abschliessend wird die neue „Activity“ TimeEntryActivity gestartet sowie die Ressourcen, welche durch die Activity belegt wurden, wieder freigegeben.

Im Falle einer negativen Antwort vom Server auf die Anfrage, wird dem Benutzer eine Fehlermeldung angezeigt.

Für eine visuelle Darstellung des beschriebenen Ablaufs wird auf die Abbildung 26 - Sequenzdiagramm Authentifizierung verwiesen.

## Beschreibung Interaktionsdiagramm Stundeneintrag erfassen

Nach dem Drücken der Schaltfläche Starten wird in einem ersten Schritt geprüft, ob die Zeitmessung bereits gestartet wurde. Falls die Messung noch nicht gestartet wurde, wird ein neuer Zeiteintrag (TimeEntry) erzeugt und der Zustand mittels der Methode setMeasurement() gespeichert.

Wenn die Zeitmessung bereits aktiv war, wird die Messung gestoppt, die GPS-Anzeige ausgeblendet und keine GPS-Koordinaten mehr empfangen. Anschliessend wird die aktuelle GPS Position in ein Attribut abgelegt. Sobald dies durchgeführt wurde, werden sämtliche Angaben in die lokale Datenbank gespeichert.

Die Abbildung 27 - Interaktionsdiagramm Stundeneintrag erfassen zeigt den sequenziellen Ablauf anhand einer Illustration.

## Interaktionsdiagramm Authentifizierung auf Android Client

Das nachfolgende Interaktionsdiagramm zeigt den Ablauf einer Authentifizierung falls der Mitarbeiter auf die Schaltfläche „Anmelden“ drückt.



Abbildung - Sequenzdiagramm Authentifizierung

## Interaktionsdiagramm Stundeneintrag erfassen

Die folgende Darstellung zeigt den Prozess bzw. die Interaktion zwischen den verschiedenen Objekten, wenn der Benutzer die Schaltfläche starten gedrückt hat. Bereits im Voraus werden bei der Initialisierung dieser „Activity“ alle benötigten Objekte erzeugt, dieser Teil wird jedoch aus der untenstehenden Abbildung nicht ersichtlich.



Abbildung - Interaktionsdiagramm Stundeneintrag erfassen

## Synchronisation Server/Client

Um die Daten zwischen dem Server und dem Android Client periodisch zu übermitteln existiert ein spezieller Service, welcher für die komplette Synchronisation zuständig ist. Dieser Abgleich wird unter dem Kapitel 4.7.5.1 Übersicht Synchronisation näher erläutert.

# Externes Design

Das externe Design von MRT wurde ausgelagert in die Datei Externes Design.docx.

# Server API

Die API von MRT wurde ausgelagert in die Datei API.docx.

# Prozesse und Threads

<Wenn mehrere Prozesse oder Threads eingesetzt werden wird hier beschrieben, wie diese ablaufen, miteinander funktionieren, Daten austauschen, sich synchronisieren, etc...>

**TODO** Lukas, ev. einige Verweise

## Process View

**TODO** Lukas

## Server

Der Ruby on Rails Server kann gleichzeitig mehrere Clients bedienen und funktioniert mit mehreren Prozessen. Hinter der Webapplikation steht ein MySQL Server, der die Daten persistent speichert. Auf der Datenbank können die einzelnen Transaktionen nur seriell erfolgen. Die Synchronisation wird vom Ruby on Rails Framework im Zusammenspiel mit der Datenbank übernommen.

Da die Webapplikations-Prozesse über die Datenbank und nicht direkt interagieren, werden diese nicht weiter beschrieben.

## Client

### Process View

Auf dem Android Client gibt es zwei wichtige Prozesse in Bezug auf MRT. Der eine Prozess ist für die aktuelle Activity verantwortlich, der zweite Prozess für die Synchronisation. Die beiden Prozesse tauschen nicht direkt Daten aus, sondern nur indirekt über die SQLite Datenbank. Die Datenbank ist für die Synchronisation verantwortlich. Konkret heisst das, dass weder die Activity noch die Synchronisationsservices voneinander wissen, wenn etwas in der Datenbank geändert wurde.

#### Activity

Die Activity legt neue Stundeneinträge an, die zum Server übertragen werden müssen. Die Übertragung wird zu einem späteren Zeitpunkt vom Synchronisationsservice gestartet.

Zusätzlich benötigt die Activity sowohl Stundeneintragstypen und Kunden, damit der Benutzer jeweils einen an einem Stundeneintrag zuweisen kann. Diese werden periodisch von der Datenbank gelesen und im GUI angezeigt.

#### Synchronisationsservice

Der Synchronisationsservice ist für die Synchronisation der Stundeneinträge, der Kunden und der Stundeneintragstypen verantwortlich. Die Datensätze werden jeweils direkt von der Datenbank gelesen / in die Datenbank geschrieben bzw. vom Server gelesen / dem Server gesendet. Die Synchronisation wird mithilfe der Felder created\_at, updated\_at und deleted\_at realisiert. Also werden nur die jeweiligen Daten synchronisiert, die noch nicht übertragen wurden oder tatsächlich geändert haben.

Da der Service nichts von der Activity oder dem Server weiss, also auch nicht wann ein neuer Stundeneintrag / Kunde / Stundeneintragstyp angelegt wurde, muss die Synchronisation periodisch erfolgen.

# Datenspeicherung

<Beschreibung mit Diagramm der Datenspeicherung [Data Model]. (zum Beispiel: Datenbank)>

## Persistenz Rails

Auf dem Server werden sämtliche Daten persistent gespeichert. Um dies durchzuführen, werden alle Domänenmodelle mitsamt den Attributen unabhängig von der konkreten Datenbankimplementierung auf so genannte ActiveRecords abgebildet.

Um das Schema der Datenbank aufzubauen werden anhand sogenannter Migrationsdateien die passenden Tabellen in der Datenbank erzeugt. Jede Migrationsdatei enthält einen eindeutigen Timestamp, damit diese in genau einer Reihenfolge ausgeführt werden. Zusätzlich verfügt jede Datei über eine up und down Methode. Die up Methode ist zum Beispiel dafür verantwortlich, eine neue Tabelle anzulegen, während die down Methode jeweils immer das Gegenteil der up Methode machen soll, zum Beispiel eine Tabelle löschen. Dadurch ist es möglich, die Migrationen einzeln einzuspielen oder auch Rollbacks zu machen. Die Migrationen sind natürlich auch datenbankimplementationsunabhängig.

Die Datenbank wurde nach dem Prinzip „Convention over Configuration“ erzeugt. In Ruby on Rails bedeutet das folgende wichtige Dinge:

* Tabellennamen und Feldnamen werden klein\_und\_mit\_underscore geschrieben
* Jede Tabelle verfügt über folgende Attribute
  + id: Die automatisch inkrementierte ID des Records
  + created\_at: Datum und Zeit, wann der Record erstellt wurde
  + updated\_at: Datum und Zeit, wann der Record zuletzt editiert wurde
* Jeder Fremdschlüsselname setzt sich aus Fremdtabelle und „\_id“ zusammen. Zum Beispiel lautet der Fremdschlüssel für die Tabelle „time\_entry“ dann „time\_entry\_id“.

In MRT wird als Datenbankimplementierung die verbreitete und als performant bekannte Implementierung MySQL eingesetzt.

**TODO** Verweis Models

## Persistenz Android

Die Daten werden mithilfe des ORMLite in einer lokalen SQLite Datenbank gespeichert. Dieses ORM hat sich während der Evaluation sehr bewährt. Weitere Infos zum Entscheid sind unter 7.1.1 Datenspeicherung / ORM einsehbar.

**TODO** Verweis Models

# Grössen und Leistung

## Server

Die Speicherauslastung kann je nach Serverkonfiguration stark variieren. Oftmals bedeutet mehr Speicher natürlich bessere Performance.

Für die Ruby on Rails Applikation wird auf dem Server pro Passenger Prozess ca. 20-30 MB Ram benötigt. Standardmässig sind maximal 6 Threads im Thread Pool drin, also benötigt der Server ca. 180 MB Ram für die Passenger Threads. Dadurch ergibt sich, dass der Server, inklusive Linux Betriebssystem und laufendem Apache, ungefähr 512 MB Ram haben sollte (Quelle: http://www.alfajango.com/blog/performance-tuning-for-phusion-passenger-an-introduction/).

Auf einem Server mit bis zu 30 Threads im Thread Pool werden 2 GB Ram empfohlen. Von dieser Konfiguration wird im Folgenden ausgegangen.

### Benutzer

Eine Anfrage dauert im Schnitt etwa 25-50ms, wodurch maximal bis ca. 20-40 Anfragen pro Sekunde beantwortet werden könnten. Dadurch könnten etwa 600-1200 Mitarbeiter, die in Spitzenzeiten pro Minute 1 Anfrage betätigen, arbeiten.

### Einträge

Zurzeit ist keine Pagination implementiert, was die maximale Anzahl Einträge drastisch einschränkt. Geht man davon aus, dass jeder Stundeneintrag ca. 474 Bytes zur Übertragung benötigt, und die maximal übertragenen Daten 2MB nicht überschritten werden dürfen, ergeben sich maximal 4000 Stundeneinträge.

Würde eine Pagination implementiert, was sehr einfach gemacht werden könnte, würden so viele Einträge unterstützt, wie die Datenbank hinter der Applikation verarbeiten kann. Konfiguriert man die richtige Datenbank (z.B. Oracle) richtig, so würden bis zu mehreren Mio. Stundeneinträge gemacht werden können.

### Load Balancing

Mithilfe von Load Balancing kann die Applikation auf mehrere Server verteilt werden. Zusätzlich kann der Datenbankserver auf einen eigenen Rechner oder sogar in die Cloud ausgelagert werden, wodurch auch sehr viele Anfragen behandelbar würden.

### Fazit

Der Serverteil der Applikation ist sehr leistungsstark, müsste aber für sehr viele Anfragen noch getunt und konfiguriert werden.

## Client

Da auf dem Android Client die Stundeneinträge nur zwischengespeichert werden, wären es hier eine bestimmte Anzahl Kunden, die nicht überschritten werden dürfte. In SQLite ist die maximale Anzahl Einträge 1073741823 (Quelle: http://www.sqlite.org/limits.html), wobei sich die tatsächlich maximale Anzahl durch den freien Speicher begrenzen würde.

Um die Usability des Clients nicht negativ zu beeinflussen, sind maximal 300 Stundeneintragstypen und maximal 5‘000 Kunden zu empfehlen.

Falls die Anforderungen höher wären, so müsste die Applikation getunt und angepasst, nicht aber komplett neu entwickelt werden.

# Anhang

## Architekturentscheide

### Starker Server / mobiler Client

Ein starker Server und ein mobiler Client ist ein bewährtes Konzept in der heutigen Welt. Der Server muss immer schneller sein und mehr Anfragen behandeln, während der Client überallhin mitgenommen werden soll.

Im Fall von MRT geht es ja um ein mobiles Reporting-Tool. Es soll also möglich sein, von überall aus Stundeneinträge auf möglichst einfache Art zu erfassen.

#### Client

Für den Client haben wir Android ausgewählt, weil immer mehr Mobiltelefone heute mit dem Betriebssystem Android verkauft werden. Dadurch ist es eine Technologie, die auch in der Zukunft interessant sein wird. Zudem hat jeder Mitarbeiter sowieso ein Telefon und muss dadurch nicht eine spezielle Hardware kaufen.

#### Server

Um schnell einen Server zu entwickeln, haben wir Ruby on Rails gewählt. Dieses mächtige Framework ermöglicht ideales Prototyping und eine sehr schnelle Entwicklung. Dies ist in diesem Projekt umso wichtiger, da nur wenig Budget für die Konstruktion des Produktes zur Verfügung steht.

#### HTTP / HTTPS

HTTP / HTTPS ist ein sehr einfaches Protokoll, das von verschiedensten Geräten und Software unterstützt wird. So ist es mit diesem Protokoll möglich, sowohl PC’s mit einem Browser wie auch mobile Geräte mit einer installierten Software anzusprechen.

#### JSON

JSON ist ein leichtgewichtiges Protokoll zur Serialisierung und Übertragung von Daten, ähnlich wie XML. Durch seine Einfachheit stehen in verschiedensten Sprachen Libraries zur Codierung von JSON zur Verfügung. Durch die Server API (siehe 7 Server API) ist es auch auf einfache Art und Weise möglich, weitere Clients für MRT zu entwickeln.

### Datenspeicherung / ORM

Dass die Daten temporär auf dem Client gespeichert werden müssen, ergibt sich durch die Anforderung, dass der Client auch ohne Serververbindung funktionieren muss. Die verschiedenen Varianten wurden nachfolgend untersucht und ausgewertet.

Schlussendlich wurde für ORMLite entschieden, da sich dieser OR Mapper am besten für die Android Plattform eignet und speziell dafür optimiert wurde. Ausserdem ist dieser OR Mapper schlank und die angebotene Grundfunktionalität reicht völlig aus.

#### Kein ORM, primitives SQLite

Jedes ORM ist ein wenig anders realisiert, verfolgt eine andere Strategie oder hat andere Befehle. Kurz gesagt: jedes ORM bedeutet Einarbeitungszeit. Da das Projekt nicht sehr gross ist und es in erster Linie ums Software Engineering geht, wurde diese Möglichkeit natürlich in Betracht gezogen.

Leider wurde schnell festgestellt, dass ohne ORM schnell redundanter Code entsteht, und dass schnell ein kleines ORM entwickelt werden müsste (mit Annotations oder Reflections), um Redundanzen zu vermeiden. Da dies viel Aufwand wäre oder ohne eigenes ORM den Prinzipien eines guten Software Engineerings wider­sprechen würde (DRY), wurde nach einer alternativen Lösung gesucht.

#### ORMLite

ORMLite ist ein leichtgewichtiger OR Mapper für Java Applikationen (Grösse: 234 KB). Er abstrahiert die Grundfunktionalitäten des SQL. Es werden viele verschiedene Datenbanken nativ unterstützt (volle Unterstützung für SQLite, MySQL, Postgres, Microsoft SQL Server, H2, Derby, HSQLDB, teilweise Unterstützung für DB2, Oracle, ODBC und Netezza) und ist leicht für andere Datenbanken erweiterbar, was für den modularen Aufbau des OR Mappers spricht. Zusätzlich bietet ORMLite eine einfache und strukturierte Integration in die Android Plattform.

Weitere Informationen unter http://ormlite.com/

#### DB4O

DB4O ist ein Schwergewicht in Sachen ORM (> 40MB!). Zusätzlich bietet DB4O sehr viel Funktionalität, die für dieses Projekt nicht genützt werden würde. Aus diesen zwei Gründen wurde gegen DB4O entschieden.

Weitere Informationen unter http://developer.db4o.com/

#### ActiveAndroid

ActiveAndroid ist ein schlankes aber mächtiges ORM, das speziell für Android entwickelt wurde. Leider ist dieses ORM nicht OpenSource und kostet für 5 Entwickler 60 USD. Aus diesem Grund wird ActiveAndroid nicht verwendet.

Weitere Informationen unter https://www.activeandroid.com/

#### Android-active-record

Android-active-record ist zwar grundsätzlich ein sehr gutes ORM, da es sehr klein ist und sich sehr einfach in das Android Framework integrieren lässt. Allerdings ist dieses ORM noch stark in der Entwicklungsphase und ist somit noch nicht stabil genug, um in einer produktiven Applikation eingesetzt zu werden.

Weitere Informationen unter http://code.google.com/p/android-active-record/

## Auszug Fehlerbericht per Mail von Rails

A NoMethodError occurred in time\_entries#index:

undefined method `demo\_for\_error\_\_method\_does\_not\_exist!' for #<TimeEntriesController:0x7f05efbbe1e8>

app/controllers/time\_entries\_controller.rb:7:in `index'

-------------------------------

Request:

-------------------------------

\* URL : http://mrt.elmermx.ch/time\_entries

\* IP address: 84.75.164.150

\* Parameters: {"action"=>"index", "controller"=>"time\_entries"}

\* Rails root: /var/www/mrt

-------------------------------

Session:

-------------------------------

\* session id: "d5343eef18878db41193cc1c7b2a813e"

\* data: {"\_csrf\_token"=>"Al/esLfLC2bHKwbydo0loQSHUZNsZhDNa8OPRTSJS4I=",

"warden.user.user.key"=>["Secretary", [1], "$2a$10$shRjNUvc5t6pqCJAT7dgZ."],

"session\_id"=>"d5343eef18878db41193cc1c7b2a813e"}

-------------------------------

Environment:

-------------------------------

\* DOCUMENT\_ROOT : /var/www/mrt/public

\* HTTP\_ACCEPT : text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

\* HTTP\_ACCEPT\_CHARSET : ISO-8859-1,utf-8;q=0.7,\*;q=0.7

\* HTTP\_ACCEPT\_ENCODING : gzip, deflate

\* HTTP\_ACCEPT\_LANGUAGE : de-ch,de-de;q=0.8,de;q=0.6,en-us;q=0.4,en;q=0.2

\* HTTP\_CACHE\_CONTROL : max-age=0

\* HTTP\_CONNECTION : keep-alive

\* HTTP\_COOKIE : \_\_utma=243999569.1626227150.1296509602.1302298896.1304551613.5; \_\_utmz=243999569.1296509602.1.1.utmccn=(direct)|utmcsr=(direct)|utmcmd=(none); \_mrt\_session=BAh7CCIQX2NzcmZfdG9rZW4iMUFsL2VzTGZMQzJiSEt3YnlkbzBsb1FTSFVaTnNaaEROYThPUFJUU0pTNEk9Ig9zZXNzaW9uX2lkIiVkNTM0M2VlZjE4ODc4ZGI0MTE5M2NjMWM3YjJhODEzZSIZd2FyZGVuLnVzZXIudXNlci5rZXlbCCIOU2VjcmV0YXJ5WwZpBiIiJDJhJDEwJHNoUmpOVXZjNXQ2cHFDSkFUN2RnWi4%3D--8b01677c7005b5e019a2098aed5b7175e93684ef; remember\_user\_token=BAhbB1sGaQYiIiQyYSQxMCRzaFJqTlV2YzV0NnBxQ0pBVDdkZ1ou--757b078178d7402cb54627a385e1d964d32ca371

\* HTTP\_HOST : mrt.elmermx.ch

\* HTTP\_IF\_NONE\_MATCH : "41da6706cdf34f8d63e26c2acf8459a7"

\* HTTP\_KEEP\_ALIVE : 115

\* HTTP\_REFERER : http://mrt.elmermx.ch/users/1

\* HTTP\_USER\_AGENT : Mozilla/5.0 (Windows NT 6.1; WOW64; rv:2.0.1) Gecko/20100101 Firefox/4.0.1

\* PATH\_INFO : /time\_entries

\* QUERY\_STRING :

\* REMOTE\_ADDR : 84.75.164.150

\* REMOTE\_PORT : 21498

\* REQUEST\_METHOD : GET

\* REQUEST\_URI : /time\_entries

\* SCRIPT\_NAME :

\* SERVER\_ADDR : 192.168.0.100

\* SERVER\_ADMIN : [no address given]

\* SERVER\_NAME : mrt.elmermx.ch

\* SERVER\_PORT : 443

\* SERVER\_PROTOCOL : HTTP/1.1

\* SERVER\_SOFTWARE : Apache

\* \_ : \_

\* action\_controller.instance : time\_entries#index

\* action\_dispatch.cookies : {"\_\_utma"=>"243999569.1626227150.1296509602.1302298896.1304551613.5", "remember\_user\_token"=>"BAhbB1sGaQYiIiQyYSQxMCRzaFJqTlV2YzV0NnBxQ0pBVDdkZ1ou--757b078178d7402cb54627a385e1d964d32ca371", "\_\_utmz"=>"243999569.1296509602.1.1.utmccn=(direct)|utmcsr=(direct)|utmcmd=(none)", "\_mrt\_session"=>"BAh7CCIQX2NzcmZfdG9rZW4iMUFsL2VzTGZMQzJiSEt3YnlkbzBsb1FTSFVaTnNaaEROYThPUFJUU0pTNEk9Ig9zZXNzaW9uX2lkIiVkNTM0M2VlZjE4ODc4ZGI0MTE5M2NjMWM3YjJhODEzZSIZd2FyZGVuLnVzZXIudXNlci5rZXlbCCIOU2VjcmV0YXJ5WwZpBiIiJDJhJDEwJHNoUmpOVXZjNXQ2cHFDSkFUN2RnWi4=--8b01677c7005b5e019a2098aed5b7175e93684ef"}

\* action\_dispatch.parameter\_filter : [:password, /RAW\_POST\_DATA/]

\* action\_dispatch.remote\_ip : 84.75.164.150

\* action\_dispatch.request.content\_type :

\* action\_dispatch.request.formats : [#<Mime::Type:0x7f05eecbcd98 @synonyms=["application/xhtml+xml"], @symbol=:html, @string="text/html">]

\* action\_dispatch.request.parameters : {"action"=>"index", "controller"=>"time\_entries"}

\* action\_dispatch.request.path\_parameters : {:action=>"index", :controller=>"time\_entries"}

\* action\_dispatch.request.query\_parameters : {}

\* action\_dispatch.request.request\_parameters : {}

\* action\_dispatch.request.unsigned\_session\_cookie: {"\_csrf\_token"=>"Al/esLfLC2bHKwbydo0loQSHUZNsZhDNa8OPRTSJS4I=", "warden.user.user.key"=>["Secretary", [1], "$2a$10$shRjNUvc5t6pqCJAT7dgZ."], "session\_id"=>"d5343eef18878db41193cc1c7b2a813e"}

\* action\_dispatch.secret\_token : 8f48532803bf9d224e01788cba614051aa4cdbdbd3e141606e6c63af12f3045dfa01c4acd42510cfd96e0989b8fa9506f1c28b53e45190363b5b1500c6d41761

\* exception\_notifier.options : {:sender\_address=>"\"MRT Exception Notifier\" <mrt@elmermx.ch>", :exception\_recipients=>["lukas.elmer@gmail.com"], :ignore\_exceptions=>[ActiveRecord::RecordNotFound, AbstractController::ActionNotFound, ActionController::RoutingError], :email\_prefix=>"[MRT Exception Notification] "}

\* rack.errors : #<IO:0x7f05f4de3ab8>

\* rack.input : #<PhusionPassenger::Utils::RewindableInput:0x7f05efc4ac10>

\* rack.multiprocess : true

\* rack.multithread : false

\* rack.request.cookie\_hash : {"\_\_utma"=>"243999569.1626227150.1296509602.1302298896.1304551613.5", "remember\_user\_token"=>"BAhbB1sGaQYiIiQyYSQxMCRzaFJqTlV2YzV0NnBxQ0pBVDdkZ1ou--757b078178d7402cb54627a385e1d964d32ca371", "\_\_utmz"=>"243999569.1296509602.1.1.utmccn=(direct)|utmcsr=(direct)|utmcmd=(none)", "\_mrt\_session"=>"BAh7CCIQX2NzcmZfdG9rZW4iMUFsL2VzTGZMQzJiSEt3YnlkbzBsb1FTSFVaTnNaaEROYThPUFJUU0pTNEk9Ig9zZXNzaW9uX2lkIiVkNTM0M2VlZjE4ODc4ZGI0MTE5M2NjMWM3YjJhODEzZSIZd2FyZGVuLnVzZXIudXNlci5rZXlbCCIOU2VjcmV0YXJ5WwZpBiIiJDJhJDEwJHNoUmpOVXZjNXQ2cHFDSkFUN2RnWi4=--8b01677c7005b5e019a2098aed5b7175e93684ef"}

\* rack.request.cookie\_string : \_\_utma=243999569.1626227150.1296509602.1302298896.1304551613.5; \_\_utmz=243999569.1296509602.1.1.utmccn=(direct)|utmcsr=(direct)|utmcmd=(none); \_mrt\_session=BAh7CCIQX2NzcmZfdG9rZW4iMUFsL2VzTGZMQzJiSEt3YnlkbzBsb1FTSFVaTnNaaEROYThPUFJUU0pTNEk9Ig9zZXNzaW9uX2lkIiVkNTM0M2VlZjE4ODc4ZGI0MTE5M2NjMWM3YjJhODEzZSIZd2FyZGVuLnVzZXIudXNlci5rZXlbCCIOU2VjcmV0YXJ5WwZpBiIiJDJhJDEwJHNoUmpOVXZjNXQ2cHFDSkFUN2RnWi4%3D--8b01677c7005b5e019a2098aed5b7175e93684ef; remember\_user\_token=BAhbB1sGaQYiIiQyYSQxMCRzaFJqTlV2YzV0NnBxQ0pBVDdkZ1ou--757b078178d7402cb54627a385e1d964d32ca371

\* rack.request.query\_hash : {}

\* rack.request.query\_string :

\* rack.run\_once : false

\* rack.session : {"\_csrf\_token"=>"Al/esLfLC2bHKwbydo0loQSHUZNsZhDNa8OPRTSJS4I=", "session\_id"=>"d5343eef18878db41193cc1c7b2a813e", "warden.user.user.key"=>["Secretary", [1], "$2a$10$shRjNUvc5t6pqCJAT7dgZ."]}

\* rack.session.options : {:secure=>false, :expire\_after=>nil, :httponly=>true, :domain=>nil, :path=>"/", :id=>"d5343eef18878db41193cc1c7b2a813e"}

\* rack.url\_scheme : http

\* rack.version : [1, 0]

\* warden : #<Warden::Proxy:0x7f05efc24b28>

\* Process: 25196

\* Server : lukeshost

-------------------------------

Backtrace:

-------------------------------

app/controllers/time\_entries\_controller.rb:7:in `index'

actionpack (3.0.5) lib/action\_controller/metal/implicit\_render.rb:4:in `send\_action'

actionpack (3.0.5) lib/action\_controller/metal/implicit\_render.rb:4:in `send\_action'

actionpack (3.0.5) lib/abstract\_controller/base.rb:150:in `process\_action'

actionpack (3.0.5) lib/action\_controller/metal/rendering.rb:11:in `process\_action'

actionpack (3.0.5) lib/abstract\_controller/callbacks.rb:18:in `process\_action'

activesupport (3.0.5) lib/active\_support/callbacks.rb:440:in `\_run\_\_1115460920\_\_process\_action\_\_199225275\_\_callbacks'

activesupport (3.0.5) lib/active\_support/callbacks.rb:409:in `send'

activesupport (3.0.5) lib/active\_support/callbacks.rb:409:in `\_run\_process\_action\_callbacks'

activesupport (3.0.5) lib/active\_support/callbacks.rb:93:in `send'

activesupport (3.0.5) lib/active\_support/callbacks.rb:93:in `run\_callbacks'

actionpack (3.0.5) lib/abstract\_controller/callbacks.rb:17:in `process\_action'

actionpack (3.0.5) lib/action\_controller/metal/instrumentation.rb:30:in `process\_action'

activesupport (3.0.5) lib/active\_support/notifications.rb:52:in `instrument'

activesupport (3.0.5) lib/active\_support/notifications/instrumenter.rb:21:in `instrument'

activesupport (3.0.5) lib/active\_support/notifications.rb:52:in `instrument'

actionpack (3.0.5) lib/action\_controller/metal/instrumentation.rb:29:in `process\_action'

actionpack (3.0.5) lib/action\_controller/metal/rescue.rb:17:in `process\_action'

actionpack (3.0.5) lib/abstract\_controller/base.rb:119:in `process'

actionpack (3.0.5) lib/abstract\_controller/rendering.rb:41:in `process'

actionpack (3.0.5) lib/action\_controller/metal.rb:138:in `dispatch'

actionpack (3.0.5) lib/action\_controller/metal/rack\_delegation.rb:14:in `dispatch'

actionpack (3.0.5) lib/action\_controller/metal.rb:178:in `action'

actionpack (3.0.5) lib/action\_dispatch/routing/route\_set.rb:62:in `call'

actionpack (3.0.5) lib/action\_dispatch/routing/route\_set.rb:62:in `dispatch'

actionpack (3.0.5) lib/action\_dispatch/routing/route\_set.rb:27:in `call'

rack-mount (0.6.14) lib/rack/mount/route\_set.rb:148:in `call'

rack-mount (0.6.14) lib/rack/mount/code\_generation.rb:93:in `recognize'

rack-mount (0.6.14) lib/rack/mount/code\_generation.rb:82:in `optimized\_each'

rack-mount (0.6.14) lib/rack/mount/code\_generation.rb:92:in `recognize'

rack-mount (0.6.14) lib/rack/mount/route\_set.rb:139:in `call'

actionpack (3.0.5) lib/action\_dispatch/routing/route\_set.rb:492:in `call'

exception\_notification (2.4.0) lib/exception\_notifier.rb:19:in `call'

warden (1.0.3) lib/warden/manager.rb:35:in `call'

warden (1.0.3) lib/warden/manager.rb:34:in `catch'

warden (1.0.3) lib/warden/manager.rb:34:in `call'

actionpack (3.0.5) lib/action\_dispatch/middleware/best\_standards\_support.rb:17:in `call'

actionpack (3.0.5) lib/action\_dispatch/middleware/head.rb:14:in `call'

rack (1.2.2) lib/rack/methodoverride.rb:24:in `call'

actionpack (3.0.5) lib/action\_dispatch/middleware/params\_parser.rb:21:in `call'

actionpack (3.0.5) lib/action\_dispatch/middleware/flash.rb:182:in `call'

actionpack (3.0.5) lib/action\_dispatch/middleware/session/abstract\_store.rb:149:in `call'

actionpack (3.0.5) lib/action\_dispatch/middleware/cookies.rb:302:in `call'

activerecord (3.0.5) lib/active\_record/query\_cache.rb:32:in `call'

activerecord (3.0.5) lib/active\_record/connection\_adapters/abstract/query\_cache.rb:28:in `cache'

activerecord (3.0.5) lib/active\_record/query\_cache.rb:12:in `cache'

activerecord (3.0.5) lib/active\_record/query\_cache.rb:31:in `call'

activerecord (3.0.5) lib/active\_record/connection\_adapters/abstract/connection\_pool.rb:354:in `call'

actionpack (3.0.5) lib/action\_dispatch/middleware/callbacks.rb:46:in `call'

activesupport (3.0.5) lib/active\_support/callbacks.rb:415:in `\_run\_call\_callbacks'

actionpack (3.0.5) lib/action\_dispatch/middleware/callbacks.rb:44:in `call'

rack (1.2.2) lib/rack/sendfile.rb:107:in `call'

actionpack (3.0.5) lib/action\_dispatch/middleware/remote\_ip.rb:48:in `call'

actionpack (3.0.5) lib/action\_dispatch/middleware/show\_exceptions.rb:47:in `call'

railties (3.0.5) lib/rails/rack/logger.rb:13:in `call'

rack (1.2.2) lib/rack/runtime.rb:17:in `call'

rack (1.2.2) lib/rack/lock.rb:11:in `call'

rack (1.2.2) lib/rack/lock.rb:11:in `synchronize'

rack (1.2.2) lib/rack/lock.rb:11:in `call'

railties (3.0.5) lib/rails/application.rb:168:in `call'

railties (3.0.5) lib/rails/application.rb:77:in `send'

railties (3.0.5) lib/rails/application.rb:77:in `method\_missing'

passenger (2.2.15) lib/phusion\_passenger/rack/request\_handler.rb:92:in `process\_request'

passenger (2.2.15) lib/phusion\_passenger/abstract\_request\_handler.rb:207:in `main\_loop'

passenger (2.2.15) lib/phusion\_passenger/rack/application\_spawner.rb:120:in `run'

passenger (2.2.15) lib/phusion\_passenger/rack/application\_spawner.rb:65:in `spawn\_application'

passenger (2.2.15) lib/phusion\_passenger/utils.rb:252:in `safe\_fork'

passenger (2.2.15) lib/phusion\_passenger/rack/application\_spawner.rb:58:in `spawn\_application'

passenger (2.2.15) lib/phusion\_passenger/rack/application\_spawner.rb:41:in `spawn\_application'

passenger (2.2.15) lib/phusion\_passenger/spawn\_manager.rb:150:in `spawn\_application'

passenger (2.2.15) lib/phusion\_passenger/spawn\_manager.rb:278:in `handle\_spawn\_application'

passenger (2.2.15) lib/phusion\_passenger/abstract\_server.rb:352:in `\_\_send\_\_'

passenger (2.2.15) lib/phusion\_passenger/abstract\_server.rb:352:in `main\_loop'

passenger (2.2.15) lib/phusion\_passenger/abstract\_server.rb:196:in `start\_synchronously'

passenger (2.2.15) bin/passenger-spawn-server:61