|  |
| --- |
| Elmer Lukas, Heidt Christina, Steiner Diego, Treichler Delia, Waltenspül Remo  12. April 2011 |

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

****

# Dokumentinformationen

## Änderungsgeschichte

|  |  |  |  |
| --- | --- | --- | --- |
| Datum | Version | Änderung | Autor |
| 28.03.2011 | 1.0 | Erste Version des Dokuments | HC |
| 11.04.2011 | 1.1 | Vervollständigung des Inhaltsverzeichnisses | TD |

## Inhaltsverzeichnis

[1 Dokumentinformationen 1](#_Toc290375829)

[1.1 Änderungsgeschichte 1](#_Toc290375830)

[1.2 Inhaltsverzeichnis 1](#_Toc290375831)

[1.3 Abbildungsverzeichnis 2](#_Toc290375832)

[2 Einführung 3](#_Toc290375833)

[2.1 Zweck 3](#_Toc290375834)

[2.2 Gültigkeitsbereich 3](#_Toc290375835)

[2.3 Definitionen und Abkürzungen 3](#_Toc290375836)

[2.4 Referenzen 3](#_Toc290375837)

[3 Software Architektur 4](#_Toc290375838)

[3.1 Architektonische Ziele & Einschränkungen 4](#_Toc290375839)

[3.1.1 Ziele 4](#_Toc290375840)

[3.1.2 Einschränkungen 4](#_Toc290375841)

[3.2 Architekturübersicht 5](#_Toc290375842)

[3.3 Kernarchitektur Übertragung Zeiteintrag 6](#_Toc290375843)

[3.3.1 Sequenzdiagramm 6](#_Toc290375844)

[3.3.2 Zustandsdiagramm 7](#_Toc290375845)

[3.4 Systemstruktur 8](#_Toc290375846)

[3.4.1 Physische Sicht 8](#_Toc290375847)

[3.4.2 Logische Sicht Rails 8](#_Toc290375848)

[3.4.3 Logische Sicht Android 10](#_Toc290375849)

[3.5 Design Pakete Rails 10](#_Toc290375850)

[3.6 Design Pakete Android 10](#_Toc290375851)

[3.6.1 Übersicht über die Packages 10](#_Toc290375852)

[3.6.2 Package services 11](#_Toc290375853)

[3.6.3 Package gui/gen 12](#_Toc290375854)

[3.6.4 Package activities 13](#_Toc290375855)

[3.6.5 Package persistence 13](#_Toc290375856)

[3.6.6 Package network 16](#_Toc290375857)

[3.6.7 Package test 17](#_Toc290375858)

[4 Prozesse und Threads 18](#_Toc290375859)

[5 Datenspeicherung 18](#_Toc290375860)

[6 Grössen und Leistung 18](#_Toc290375861)

## Abbildungsverzeichnis

[Abbildung 1 - Architekturübersicht 5](file:///E:\EigeneDateien\Studium\WorkspaceEclipse\se2p_svn.elmermx.ch\doc\05_Design\Software%20Architektur.docx#_Toc290375862)

[Abbildung 2 - Sequenzdiagramm Übertragung TimeEntry 6](file:///E:\EigeneDateien\Studium\WorkspaceEclipse\se2p_svn.elmermx.ch\doc\05_Design\Software%20Architektur.docx#_Toc290375863)

[Abbildung 3 - Zustandsdiagramm Übertragung TimeEntry 7](#_Toc290375864)

[Abbildung 4 - Deployment Diagram MRT 8](#_Toc290375865)

[Abbildung 5 - Architektur von Ruby on Rails 9](#_Toc290375866)

[Abbildung 6 - Ablauf eines Requests 9](#_Toc290375867)

[Abbildung 7 - Übersicht über die Packages 11](#_Toc290375868)

[Abbildung 8 - Package services 12](#_Toc290375869)

[Abbildung 9 - Package gui/gen 12](#_Toc290375870)

[Abbildung 10 - Package activities 13](#_Toc290375871)

[Abbildung 11 - Package persistence 14](#_Toc290375872)

[Abbildung 12 - Package models 14](#_Toc290375873)

[Abbildung 13 - Package interfaces 15](#_Toc290375874)

[Abbildung 14 - Package database 16](#_Toc290375875)

[Abbildung 15 - Package network 16](#_Toc290375876)

[Abbildung 16 - Package test 17](#_Toc290375877)

# 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/templates/template.dotx
* doc/templates/java\_formatting.xml
* doc/templates/ruby\_formatting\_settings.zip
* doc/media/logo.png

# Software Architektur

## Architektonische Ziele & Einschränkungen

<Beschreibt die Softwareanforderungen und Objekte, welche einen Einfluss auf die Architektur haben [Bspl: Safety, Security, Privacy, Distribution, …] Beinhaltet auch eine Beschreibung von Design und Implementationsstrategie, Teamstruktur, Entwicklungstools, Zeitplan, etc…>

### Ziele

* Es soll möglichst einfach möglich sein, den Android Client durch einen anderen Client (z.B. iPhone, Windows7 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.

## Architekturübersicht

Wie anhand der nachstehenden Abbildung ersichtlich, 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 kleine Datenbank, welche verwendet wird falls ein Zeiteintrag nicht direkt übermittelt werden kann.

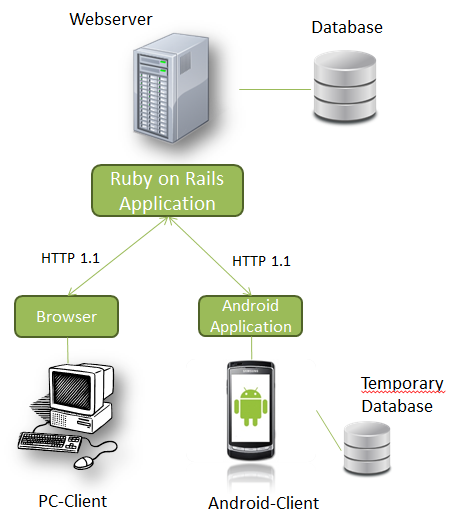


Abbildung - Architekturübersicht

## Kernarchitektur Übertragung Zeiteintrag

### Sequenzdiagramm

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



Abbildung - 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.

### Logische Sicht Rails

<Beschreibung mit Text und Diagramm der Architektur. Aufteilung in Packages (zum Beispiel: 3-Layer-Architektur mit GUI, Problem Domain und Datenhaltung)>

Die Architektur der Rails Applikation ist zu einem grossen Teil vorgegeben.

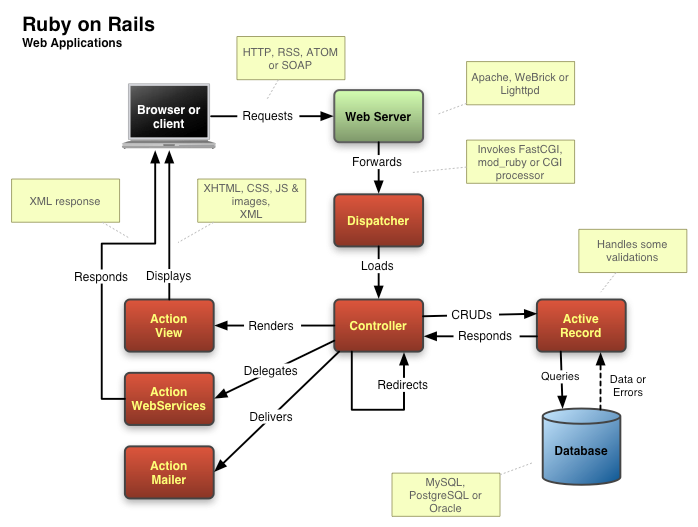


Abbildung - Architektur von Ruby on Rails

Abbildung - Ablauf eines Requests

### Logische Sicht Android

<Beschreibung mit Text und Diagramm der Architektur. Aufteilung in Packages (zum Beispiel: 3-Layer-Architektur mit GUI, Problem Domain und Datenhaltung)>

## Design Pakete Rails

<für jedes definierte Package erfolgt eine Beschreibung mit Diagramm >

Die Architektur in Ruby on Rails ist grundsätzlich bereits vorgegeben. Deshalb ist dieses Kapitel sehr kurz gehalten.

## Design Pakete Android

### Übersicht über die Packages



Abbildung - Übersicht über die Packages

### Package services

#### Beschreibung des Packages

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

#### Diagramme

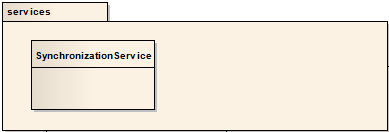


Abbildung - Package services

#### Schnittstellen

#### Operationen

##### Interne Operation 1

##### Interne Operation 2

### Package gui/gen

#### Beschreibung des Packages

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

#### Diagramme

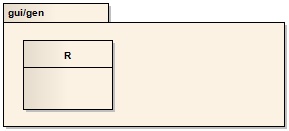


Abbildung - Package gui/gen

#### Schnittstellen

#### Operationen

##### Interne Operation 1

##### Interne Operation 2

### Package activities

#### Beschreibung des Packages

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

#### Diagramme

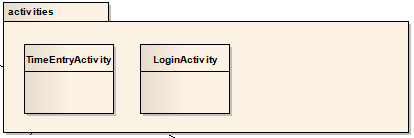


Abbildung - Package activities

#### Schnittstellen

Wie bereits beschrieben, bearbeiten „activities“ „models“ und starten „services“. Deshalb haben sie eine Abhängigkeit von den Packages „services“ und „persistence“ mit dem Unterpackage „models“. Zudem besteht auch eine Verbindung zum Package „gui/gen“.

#### Operationen

##### Interne Operation 1

##### Interne Operation 2

### Package persistence

#### Beschreibung des Packages

Das Package „persistence“ ist dafür zuständig, dass Daten persistent gespeichert werden, die Datenbank richtig initialisiert wird und automatisch Tabellen erzeugt werden und Daten korrekt in JSON serialisiert werden.

#### Diagramme

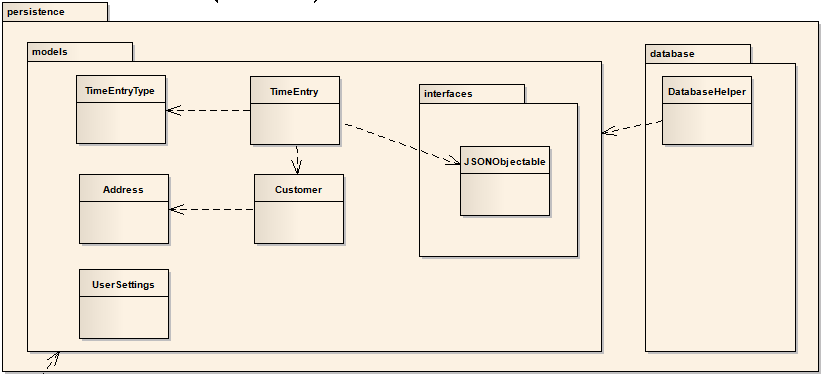


Abbildung - Package persistence

#### Unterpackages

„persistence“ beinhaltet zwei Unterpackages: „models“ und „database“.

##### Package models

###### Beschreibung des Packages

„models“ beinhaltet alle Daten, welche persistent gespeichert werden sollen. Es enthält ein Unterpackage „interfaces“.

###### Diagramme

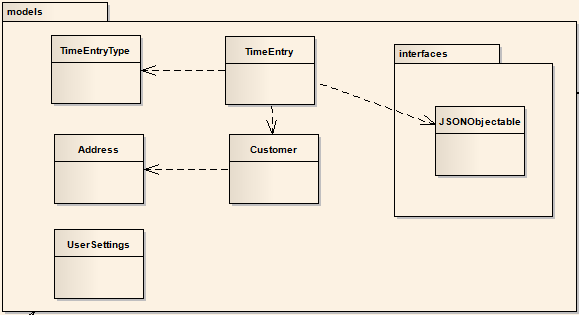


Abbildung - Package models

###### Schnittstellen

###### Operationen

Interne Operation 1

Interne Operation 2

###### Unterpackage interfaces

Beschreibung des Packages

Das Package „interfaces“ beinhaltet die JSON-Serialisierung, welche von den Klassen im Package „models“ benötigt werden.

Diagramme

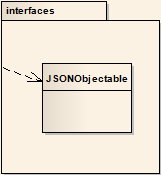


Abbildung - Package interfaces

Schnittstellen

Operationen

Interne Operation1

Interne Operation2

##### Package database

###### Beschreibung des Packages

„database“ ist dafür verantwortlich, dass eine Verbindung zur Datenbank hergestellt wird. Ist es zuständig für die Erstellung der Datenbank, mit den einzelnen Tabellen.

###### Diagramme

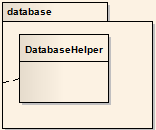


Abbildung - Package database

###### Schnittstellen

###### Operationen

Interne Operation 1

Interne Operation 2

### Package network

#### Beschreibung des Packages

Das Package „network“ ist dafür verantwortlich, dass die Daten übertragen werden.

#### Diagramme

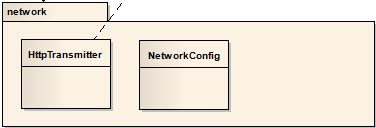


Abbildung - Package network

#### Schnittstellen

#### Operationen

##### Interne Operation 1

##### Interne Operation 2

### Package test

Das Package „test“ wird hier, obwohl es nicht in Abbildung 7 - Übersicht über die Packages (Kapitel 3.6.1 Übersicht über die Packages) ersichtlich ist, der Vollständigkeit halber aufgelistet.



Abbildung - Package test

Dieses Package enthält als Unterpackages nachfolgende Packages, welche bereits einzeln bestehen:

* „services“ (siehe 3.6.2 Package services)
* „activities“ (siehe 3.6.4 Package activities)
* „persistence“ (siehe 3.6.5 Package persistence)
* „network“ (siehe 3.6.6 Package network)

Die jeweiligen Packages beinhalten Klassen, die Testzwecken dienen.

Das Package „gui/gen“ läuft selbständig und benötigt keine Testklassen.

# Prozesse und Threads

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

# Datenspeicherung

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

# Grössen und Leistung

<Einschränkungen der Applikation bezüglich Speicher, Leistung, etc…. (zum Beispiel: Verwaltung unterstützt maximal 20'000 Einträge)>