HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Bachelorarbeit, Abteilung Informatik

TourLive Server- und Aufnahmesystem

Hochschule für Technik Rapperswil

Frühlingssemester 2013
14. Juni 2013

Autoren:	Patrizia Heer, Simon Stäheli und Florian Bentele
Betreuer:	Prof. Dr. Peter Heinzmann
Projektpartner:	Lukas Frey, cnlab Software AG
Experte:	Dr. Th. Siegenthaler, CSI Consulting AG
Gegenleser:	Prof. Hansjörg Huser
Arbeitsperiode:	31.01.2013 - 14.06.2013
Arbeitsumfang:	1080 Stunden, 360h bzw. 12 ECTS pro Student
Link:	`http://tlng.cnlab.ch`

Abstract

Das cnlab TourLive System, kurz TourLive, ermöglicht die Aufzeichnung von Positions-, Bild- und Videodaten aus dem Fahrerfeld bei Radrennen. Die mittels Nokia Symbian Geräten gesammelten Daten werden an den TourLive Server übertragen und dort Radrennsportinteressierten präsentiert.

Im Rahmen dieser Bachelorarbeit wurde das über 10-jährige System analysiert, an die aktuellen Bedürfnisse angepasst und mit Hilfe moderner Technologien umgesetzt. Das TourLive System umfasst drei voneinander getrennte Komponenten. Diese sind zum einen die Aufnahmegeräte, welche durch Android Smartphones ersetzt wurden, zum zweiten der eigentliche TourLive Server (Java Spring Webservice) und als drittes der Device Management Server (Java Spring Webservice).

Die neu entwickelte Geräteverwaltung ermöglicht die vollumfängliche Fernverwaltung und Überwachung der Aufnahmesysteme, welche bisher nur in einem rudimentären Rahmen möglich war. Die Administrationsoberfläche des TourLive Servers ermöglicht zudem eine komfortable Renn- und Etappenverwaltung. Neben den bisherigen Elementen der Webseite ist es neu auch möglich, etappenspezifische Werbebanner zu platzieren. Die Verwendung moderner Webtechnologien stellt die korrekte Darstellung der Webseite auch auf mobilen Endgeräten sicher.

Neu ist auch die Art des Videostreams. Während bisher nur Einzelbilder übertragen und auf dem Server zu einem Videostream verarbeitet werden konnten, ermöglicht das neue System die effiziente Übertragung ganzer Videosequenzen. Die Kommunikation zwischen den verschiedenen Komponenten erfolgt über eine RESTful-JSON- Schnittstelle.

Der Prototyp des Systems wurde anhand eines Probelaufs an den 50. Radsporttagen Gippingen, sowie an der zweiten Etappe der Tour de Suisse Anfangs Juni 2013 erfolgreich getestet. Wichtige Erkenntnisse aus diesen beiden Feldtests wurden dokumentiert, konnten im Rahmen dieser Arbeit aber nur noch teilweise berücksichtigt werden. Die Weiterentwicklung und der allfällige Einsatz an Radrennen liegt in den Händen der cnlab Software AG und Swisscycling.

Aufgabenstellung

Studiengang:	Informatik (I)
Institut:	ITA: Internet-Technologien und Anwendungen
Gruppe:	Patrizia Heer, Simon Stäheli, Florian Bentele
Betreuer:	Prof. Dr. Peter Heinzmann
Koreferent:	Prof. H. Huser, HSR
Experte:	Dr. Th. Siegenthaler, CSI Consulting AG
Industriepartner:	Swiss Cycling / cnlab Software ag

Ausgangslage

Das cnlab TourLive-System[1] dient zur Renndatenerfassung an Sportanlässen. Es wird seit 2004 an der Tour de Suisse und bei den Rennsporttagen Gippingen eingesetzt. Die in Schiedsrichterfahrzeugen montierten mobilen Aufnahmesysteme (Nokia Mobiltelefone mit Symbian-Anwendung) erfassen die Position von den Spitzenfahrern, den Verfolgern und dem Feld. Ferner liefern die Aufnahmesysteme Live-Bilder aus Sicht dieser Schiedsrichterfahrzeuge. Mit Hilfe der RadioTour-Android Tablet Anwendung notiert der RadioTour-Speaker die Zeitabstände und Zusammensetzung von Fahrergruppen. All diese Daten werden auf dem cnlab TourLive-Server gesammelt, aufbereitet und für die Publikation auf Webservern zu Verfügung gestellt. Swiss Cycling möchte die TourLive-Anwendung nun auch kleineren Rennveranstaltern zu Verfügung stellen können. In diesem Zusammenhang sollen auch die Aufnahmesysteme auf Android portiert und in der Funktionalität optimiert werden.

Ziel

Das überarbeitete TourLive-System soll als Gesamtpaket für Rennveranstalter zur Verfügung stehen. Die Rennveranstalter sollen ihre Anlässe zusammen mit den Renninformationen auf der Webanwendungen (Datenerfassungs- und Aufbereitungssystem) präsentieren können. Die Aufnahmesysteme sollen auf Android Smartphones funktionieren und sie sollen über eine Management Anwendung überwacht und konfiguriert werden können.

1. Tourlive-System, `www.tourlive.ch`, aufgerufen am 30.04.2013

Teilaufgaben

- Analyse
  - Detailliertes Studium des aktuellen TourLive-Systems (Symbian Aufnahmesysteme und Webanwendung)
  - Studium verschiedener Webseiten zu Radrennen, Erstellung einer Übersicht und Beurteilung der verschiedenen Elemente auf solchen Webseiten
  - Festlegung der Funktionen der Aufnahme- und Darstellungssysteme
- Entwicklung Android Aufnahmesysteme
  - Positions- und Bildaufnahmen
  - Alarming-Funktionen
  - Logging-Funktionen
  - Kommunikation mit verschiedenen Datenservern
- Entwicklung Webanwendung (Webserver für Radrennen, bei welchen mit TourLive und RadioTour Renndaten aufgezeichnet und dargestellt werden)
  - Funktionsblöcke (Zeitabstände, Fahrergruppen, Bilder, LiveTicker, Kartendarstellung, Höhenprofil, Ranglisten, Marschtabellen, Fahrerinformationen, Abstände von Aufnahmesystemen, Distanz zum Werbe- und Renntross, Zeit bis dieser eine bestimmte Stelle passiert, . . .)
  - Kommunikationsschnittstelle mit alten und neuen TourLive-Aufnahmesystemen
  - Kommunikationsschnittstelle mit RadioTour-Anwendung
  - Schnittstelle zu Zeiterfassungssystemen (Matsport)
  - Rennadministration (Fahrerlisten)
- Testeinsatz am 50. GP des Kantons Aargau vom Donnerstag, 6. Juni 2013 im Rahmen der Radsporttage Gippingen 2013

Urheberschaft[3]

Die vorliegende Arbeit basiert auf Ideen, Arbeitsleistungen, Hilfestellungen und Beiträgen gemäss folgender Aufstellung:

Gegenstand, Leistung	**Person**	**Funktion**
TourLive Server	Florian Bentele	Programmierung & Dokumentation
Android Aufnahmesystem	Patrizia Heer & Simon Stäheli	Programmierung & Dokumentation
Device Management Server	Patrizia Heer & Simon Stäheli	Programmierung & Dokumentation
Lektorat	Ursina Bentele	
Lektorat	Doris Bentele	
Lektorat	Daniel Stucki	
Lektorat	Maura Weber	
Design Poster	Rahel Naef	
Idee, Aufgabenstellung, Betreuung während der Arbeit	Prof. Dr. Peter Heinzmann	Verantwortlicher Professor
Betreuung & zur Verfügungsstellung Infrastruktur	Lukas Frey	Industriepartner

Tabelle 1: Arbeitsaufteilung und Hilfestellungen

3. Diese Erklärung basiert auf der Muster-Erklärung in den Richtlinien der HSR zur Durchführung von Projekt-, Studien-, Diplom- oder Bachelorarbeiten [HSR12]

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,
- dass wir keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt habe.

Rapperswil, 13. Juni 2013

Simon Stäheli	Florian Bentele	Patrizia Heer

Vereinbarung zur weiteren Verwendung[5]

Gegenstand der Vereinbarung

Mit dieser Vereinbarung werden die Rechte über die Verwendung und die Weiterentwicklung der Ergebnisse der Bachelorarbeit TourLive von Simon Stäheli, Florian Bentele und Patrizia Heer unter der Betreuung von Peter Heinzmann geregelt.

Urheberrecht

Die Urheberrechte stehen den Studierenden zu.

Verwendung

Die Ergebnisse der Arbeit dürfen sowohl von allen an der Arbeit beteiligten Parteien, d.h. von den Studierenden, welche die Arbeit verfasst haben, vom verantwortlichen Professor, sowie vom Industriepartner verwendet und weiterentwickelt werden. Die Namensnennung der beteiligten Parteien ist bei der Weiterverwendung erwünscht, aber nicht Pflicht.

5. Die Vereinbarung zur weiteren Verwendung stammt aus den Richtlinien zur Bachelorarbeit von Prof. Dr. P. Heinzmann [DPPH13]

Rapperswil, 13. Juni 2013

Simon Stäheli

Rapperswil, 13. Juni 2013

Florian Bentele

Rapperswil, 13. Juni 2013

Patrizia Heer

Rapperswil, 13. Juni 2013

Prof. Dr. Peter Heinzmann

Rapperswil, 13. Juni 2013

Industriepartner, cnlab Software AG, Lukas Frey

Management Summary

Ausgangslage

Mit dem *cnlab TourLive System* werden an Radrennsportanlässen Live- Informationen (Positions-, Bild- und Videodaten) gesammelt, verarbeitet und auf einer Webseite präsentiert. Das gegenwärtig eingesetzte System aus dem Jahr 2001 umfasst Nokia Symbian Aufnahmegeräte sowie einen PHP Webservice. *TourLive* wird an den verschiedensten Radrennsportanlässen in der Schweiz, unter anderem an der Tour de Suisse[6], eingesetzt.

In Zukunft soll *TourLive* auch für kleinere Radrennsportanlässe zur Verfügung gestellt werden können. Dazu wurde im Rahmen dieser Bachelorarbeit das *TourLive* System erneuert und erweitert. Neben den bestehenden Funktionalitäten soll im *TourLive Next Generation*, kurz *TourLive NG*, der Fokus auf die Fernverwaltung der Aufnahmegeräte gelegt werden.

Als Partner und Auftraggeber dieses Projektes tritt die cnlab Software AG auf, die das bestehende System in Zusammenarbeit mit dem Radsportverband Swisscycling betreibt.

Vorgehen

Nach eingehender Analyse des bestehenden Systems wurden in Zusammenarbeit mit der cnlab Software AG die Anforderungen an *TourLive NG* spezifiziert. Als Basis diente die vorangegangene Analyse sowie die Erfahrungswerte der cnlab Software AG aus den vergangenen Einsätzen. Mit Prototypen wurden einzelne Funktionen auf ihre Machbarkeit überprüft. Durch regelmässige Sitzungen mit der cnlab Software AG konnte fortlaufend auf Änderungswünsche eingegangen werden. Als Projektvorgehensmodell wurde in groben Zügen der Rational Unified Process[7] angewendet.

6. Tour de Suisse, `http://www.tds.ch`, aufgerufen am 20.05.2013
7. IBM RUP, `http://www-01.ibm.com/software/rational/rup/`, aufgerufen am 25.04.2013

Ergebnisse

Das neu entwickelte System *TourLive NG* basiert auf drei Komponenten, wie Abbildung 1 zeigt.

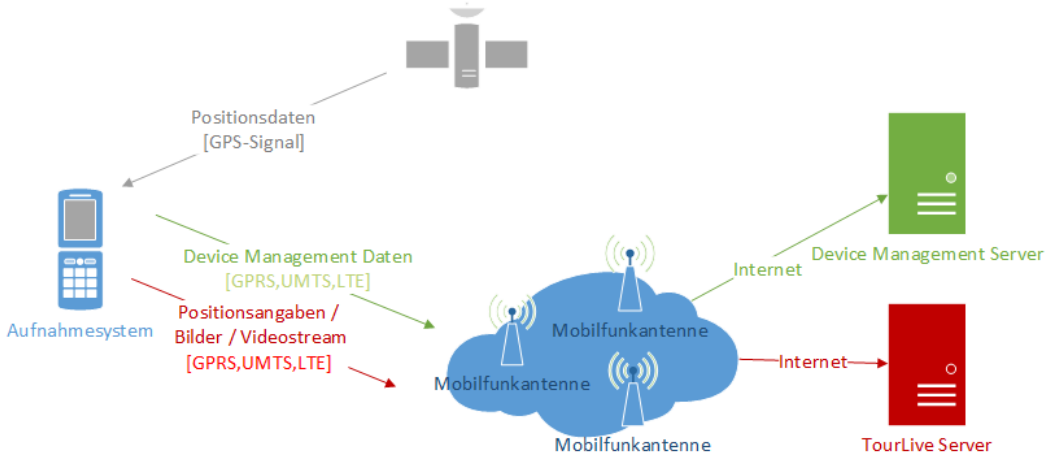


Abbildung 1: Big Picture TourLive NG

Das auf Android basierende Aufnahmesystem, der TourLive Server und der Device Management Server bilden die drei Komponenten. Der Device Management Server ermöglicht die Überwachung und Fernverwaltung der Aufnahmesysteme. Bei allfälligen Problemen ermöglicht ein Notfallwiederherstellungsdienst ein Neustart des Aufnahmesystems. Beide Serverkomponenten benutzen das Java Spring Framework. Sie verwenden Twitter Bootstrap als GUI Framework und erfüllen damit, unter Anwendung von *Responsive Web Design*, die Anforderung einer dynamischen Webseite, die auch auf Tablets und Smartphones korrekt angezeigt wird.

Rennen und Etappen können komfortabel über die mittels Passwort gesicherte Administrationsoberfläche des TourLive Servers verwaltet werden. Das Aufnahmesystem und die beiden Serverkomponenten kommunizieren über das *Hypertext Transfer Protocol (HTTP)* miteinander. Textdaten werden in einer JSON-Struktur übertragen, während Video- und Bilddaten binär an den Server gesendet werden. Der Videostream wird mit 10-sekündigen Videosequenzen realisiert, die vom TourLive Server in ein vom Browser kompatibles Format konvertiert und mit Hilfes des HTML5 <video> Tags[8] ausgeliefert werden. Aufgrund fehlender Standards und Browserinkompatibilitäten werden die Videosequenzen in verschiedenen Videoformaten zur Verfügung gestellt.

8. HTML 5 <video> Tag, `http://www.w3.org/wiki/HTML/Elements/video`, aufgerufen am 03.05.2013

Ausblick

TourLive NG wurde in der Schlussphase des Projektes unter anderem an den Radsporttagen in Gippingen und an der zweiten Etappe der Tour de Suisse ausgiebig getestet. An den beiden Rennanlässen wurde klar, wieso die Managementfunktionen so wichtig sind. Die Tests haben gezeigt, dass das System in seiner Grundfunktionalität, der Übertragung von Positionsdaten, eingesetzt werden kann. Es wurden aber auch Optimierungsmöglichkeiten eruiert und dokumentiert. Eine wichtige Erkenntnis dieser Testläufe ist die Problematik der Wärmeentwicklung der Geräte bei direkter Sonneneinstrahlung die zu Performanceeinbussen führt.

Die Weiterentwicklung des Projektes liegt in den Händen der cnlab Software AG. Der Einsatz an weiteren Radrennen wird die cnlab Software AG gemeinsam mit dem Radsportverband Swisscycling prüfen.

Inhaltsverzeichnis

1 Einleitung

Im folgenden Kapitel werden die Systemkomponenten und deren Zusammenspiel erläutert. Für das Verständnis wird das Aufgabenumfeld in einem abstrakten Kontext dargestellt. Die konkrete Implementierung und detaillierte Analyse werden in den folgenden Kapitel behandelt.

1.1 Big Picture

Zur Übersicht werden die verschiedenen Komponenten des Projektes in einem Big Picture zusammengefasst. Die untenstehende Grafik 1.1 ist in drei Abschnitte unterteilt. Im obersten Abschnitt befinden sich alle Geräte, welche Daten erfassen. Diese Aufnahmesysteme bestehen aus der Android TourLiveApp, als Teil dieser Arbeit, sowie dem Android RadioTourSpeaker, welcher im Rahmen einer Semesterarbeit [SB12] entwickelt wurde.

Im mittleren Teil befinden sich die Serversysteme. Diese bestehen aus dem TourLive Server, welcher die Renndaten empfängt und verarbeitet, sowie dem Device Management Server. Der Device Management Server bietet eine Übersicht über die registrierten Aufnahmesysteme und ermöglicht es, deren Einstellungen zu Verwalten sowie allfällige Fehlerquellen frühzeitig zu erkennen.

Der dritte Abschnitt zeigt die Anwendergruppen, die mit den Daten beliefert werden. Dies sind zum einen die Besucher der Webseite des TourLive Servers und zum anderen Drittentwickler, welche auf die öffentliche Schnittstelle zugreifen.

Teil dieser Arbeit sind die farblich hervorgehobenen Komponenten: Die Aufnahmesysteme TourLiveApp in Form einer Android App [blau], das Serversystem TourLive Server in Form einer Spring Webapplikation [rot] sowie das Serversystem Device Management Server ebenfalls in Form einer Spring Webapplikation [grün].

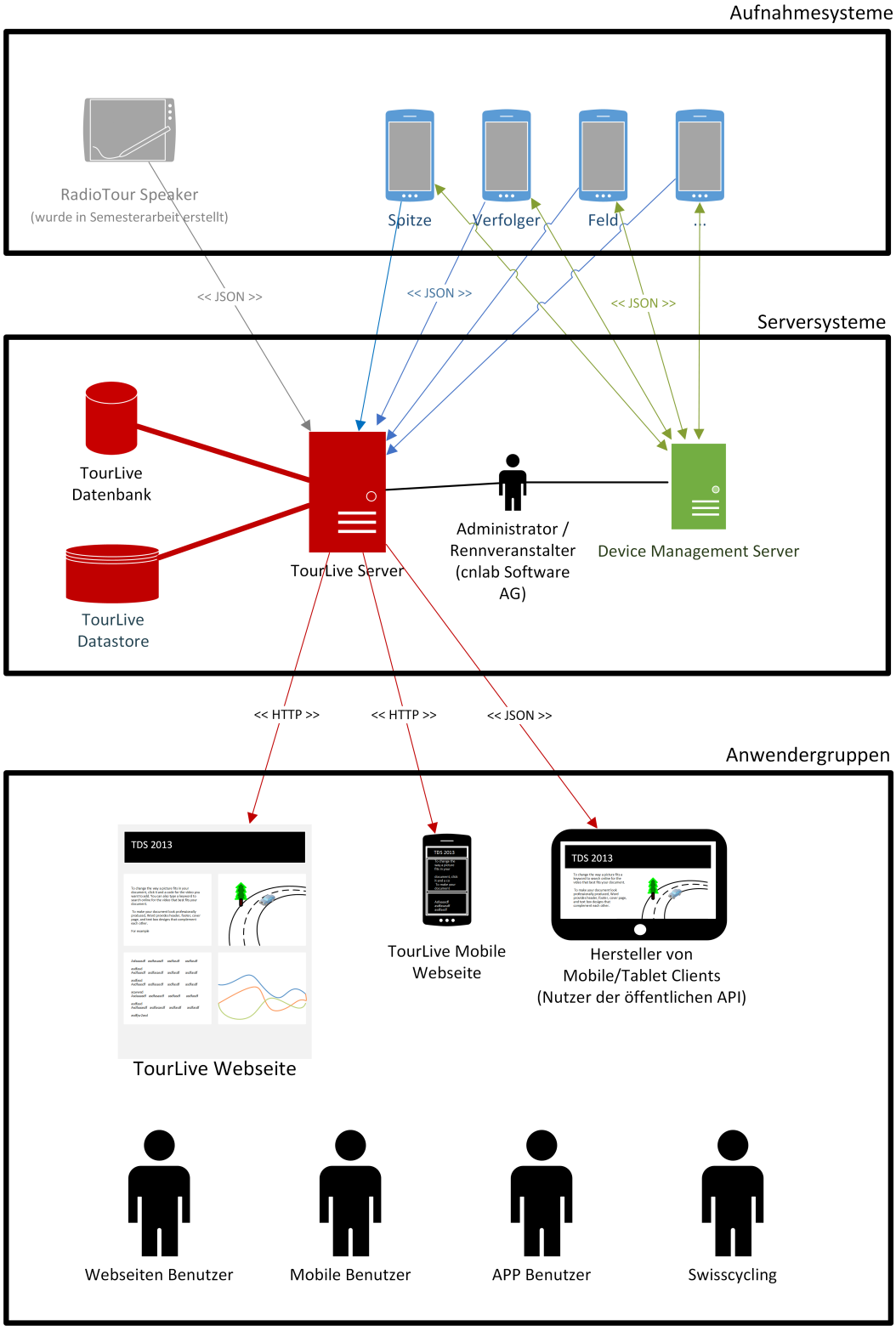


Abbildung 1.1: Big Picture

1.2 Kernelemente

1.2.1 TourLive Server

Auf der zentralen Webapplikation TourLive Server werden die von den Aufnahmesystemen eingehenden Daten gespeichert, verarbeitet, aufbereitet und weitergegeben. Bild- und Positionsdaten werden in Form einer Webseite für Radsportbegeisterte aufbereitet und in einer öffentlichen Schnittstelle für mögliche Drittentwickler zur Verfügung gestellt. Die Architektur wurde dabei so gewählt, dass das System bei hoher Last gut skaliert. Diese Komponente wird in Kapitel 2 genauer behandelt.

1.2.2 Device Management Server

Der Device Management Server hilft dabei, die aktiven Aufnahmegeräte zu verwalten. Es ersetzt das existierende Portal, welches nur wenig Funktionalität besitzt. Sollte der Device Management Server nicht verfügbar sein, sind die Aufnahmegeräte trotzdem einsatzfähig, da die Einstellungen auch lokal am Smartphone vorgenommen werden können. Auf den Device Management Server wird im Kapitel 3 eingegangen.

1.2.3 Aufnahmesystem

Ein weiterer Bestandteil des Endsystems ist das Aufnahmesystem. Es ersetzt die Symbian App auf den Nokia Handys. Periodisch werden alle gesammelten Daten vom Aufnahmesystem an den TourLive Server übermittelt. Ebenfalls periodisch werden die Einstellungen vom Device Management Server abgeholt und der Gerätestatus mitgeteilt. Werden die Einstellungen lokal verändert, so werden die veränderten Einstellungen an den Device Management Server übertragen. Im Kapitel 4 wird diese Komponente detailliert beschrieben.

2 TourLive Server

In den folgenden Abschnitten wird die Webapplikation TourLive Server erläutert. Dabei liegt der Fokus auf der Spezifikation und deren technischen Umsetzung. Dieses Kapitel richtet sich im Besonderen an Entwickler, welche an diesem Projekt weiterarbeiten möchten und an Personen, welche an den technischen Details interessiert sind.

2.1 Software Analyse

2.1.1 Spezifikation

Weite Teile der Anforderungen an die Webapplikation ergeben sich aus der Analyse der bestehenden Lösung. Im Einsatz ist zurzeit eine, von der cnlab Software AG entwickelte, *PHP* Webapplikation. Grundsätzlich soll die Funktionalität der bestehenden Lösung erweitert und in technologischer Sicht auf den aktuellsten Stand gebracht werden. In der folgenden Tabelle 2.1 wird dargestellt, ob es sich um eine neue Funktion handelt oder eine bestehende erweitert wurde.

Die Analyse der funktionalen Anforderungen sind im Anhang 7.4 umfangreich dokumentiert.

Spezifikation	**Altes System**	**Neues System**
Renn- und Etappenverwaltung	-	Benutzerfreundliche Renn- und Etappenverwaltung
Bildübertragung	1 Bild / Zeitpunkt (auch bei mehreren Aufnahmegeräten)	Bild- oder Videoübertragung
Streckenprofil	Alle Position statisch	Alle Positionen (HTML5, SVG[1])
Zeitliche Abstände	Distanz in Zeit und km zwischen Geräten	Rückstand relativ zur Spitze in Zeit und km, sowie Durchschnittsgeschwindigkeit und Höhenmeter

1. HTML5 und SVG sind zwei moderne Webtechnologien um Grafiken im Browser zu zeichnen

Rennsituation	Fahrer werden gruppiert dargestellt	Fahrer mit weiteren Informationen angereichert
Rangliste	-	Aktuelle (virtuelle) Rangliste, sortierbar
Marschtabelle	-	Marschtabelle mit Informationen und allen Positionen der Aufnahmegeräten
Kartenausschnitt	Position der Aufnahmesysteme	Positionen der Aufnahmesysteme (Farbe wählbar)
Replay	Vergangene Rennen abspielbar	Rennen vor und zurück spulen nach Zeit und Rennkilometer
Werbebanner	Statische Werbung	Einbetten von HTML Code für Werbeblock
Mobile Client	-	Webseite optimiert für alle Bildschirmgrössen und Geräte

Tabelle 2.1: Spezifikation TourLive Server

2.1.2 Evaluation Webframework

Wie aus der Aufgabenstellung zu entnehmen ist, wird keine spezifische Technologie für die Umsetzung des TourLive Servers festgelegt. Vielmehr ist es Teil der Arbeit, eine geeignete Lösung zu evaluieren und dabei auf ein *Webframework* zurückzugreifen. Die Anforderungen an das neue TourLive System bilden die Basis für die Evaluation eines dafür geeigneten Webframeworks.

Es wurden mehrere mögliche Lösungen gesucht und auf die obigen Anforderungen geprüft. Aktuell beliebte und verbreitete Frameworks wie Django (basierend auf der Programmiersprache Python) oder Ruby on Rails seien an dieser Stelle als Beispiele erwähnt. Für die detaillierte Evaluation und Gewichtung der Kriterien wird aber auf Kapitel 7.6 im Anhang verwiesen.

Entscheid

Zusammen mit dem Industriepartner fällt die Entscheidung auf das Java basierte Spring Framework[2]. Da die evaluierten Frameworks einen sehr ähnlichen Funktionsumfang bieten, waren für den Entscheid insbesondere die Vorkenntnisse der Studierenden in der Java Technologie ausschlaggebend.

2. Java Spring Framework Family, `http://springsource.org`, aufgerufen am 16.052013)

2.1.3 Weitere Technologien

Folgende weitere Technologien wurden für die Umsetzung des TourLive Servers verwendet. Im Kapitel 7.8 sind spezifische Tools und Entwicklungsumgebungen für die weitere Entwicklung aufgeführt.

ORM und Datenbank

Für die Persistierung sämtlicher Daten wird die MySQL ähnliche, quelloffene Datenbank MariaDB[3] verwendet. Dies ist eine Anforderung des Industriepartners cnlab Software AG.

Die Abbildung des Models auf der Datenbank übernimmt das Java ORM Framework Hibernate[4]. Dank unzähliger Datenbanktreiber für Hibernate, kann ein beliebiges Datenbanksystem, unter anderem auch *MariaDB*, verwendet werden.

Maven

Für die Verwaltung der externen Java Libraries wird Apache Maven[5] verwendet. Maven lädt die definierten Abhängigkeiten automatisch und kompiliert das Projekt. Weiter generiert Maven die Javadoc[6] Dokumentation zum Projekt und kann Auswertungen und statische Codeanalysen erzeugen.

Twitter Bootstrap

Die Daten werden mit dem Front-End Framework Twitter Bootstrap in Form einer HTML Webseite dargestellt. Twitter Bootstrap vereinfacht und beschleunigt die Entwicklung von Webseiten indem grundlegende Elemente angeboten werden. Es besteht aus einer komprimierten JavaScript und einer CSS Datei und kann durch viele Plugins erweitert oder verändert werden. Twitter Bootstrap ist OpenSource und in der Entwicklergemeinde sehr beliebt, da es unter anderem Webseiten für verschiedene Bildschirmgrössen (inkl. Smartphones und Tablets) optimal anpasst; diese Anpassung wird als *Responsive Web Design* bezeichnet.

2.1.4 Domain Model

In der folgenden Abbildung 2.1 wird die Problem Domain schematisch dargestellt. Die Renn- und insbesondere die Etappenklasse stehen im Zentrum der Abbildung, da die Informationen dort zusammengeführt werden. Darauf folgt die Umsetzung der einzelnen Elemente im Abschnitt 2.2.

3. MariaDB, `https://mariadb.org/`, aufgerufen am 16.05.2013
4. Hibernate ORM, `http://www.hibernate.org/`, aufgerufen am 16.05.2013
5. Apache Maven, `http://maven.apache.org/`, aufgerufen am 16.05.2013
6. Javadoc, `http://de.wikipedia.org/wiki/Javadoc`, aufgerufen am 16.05.2013

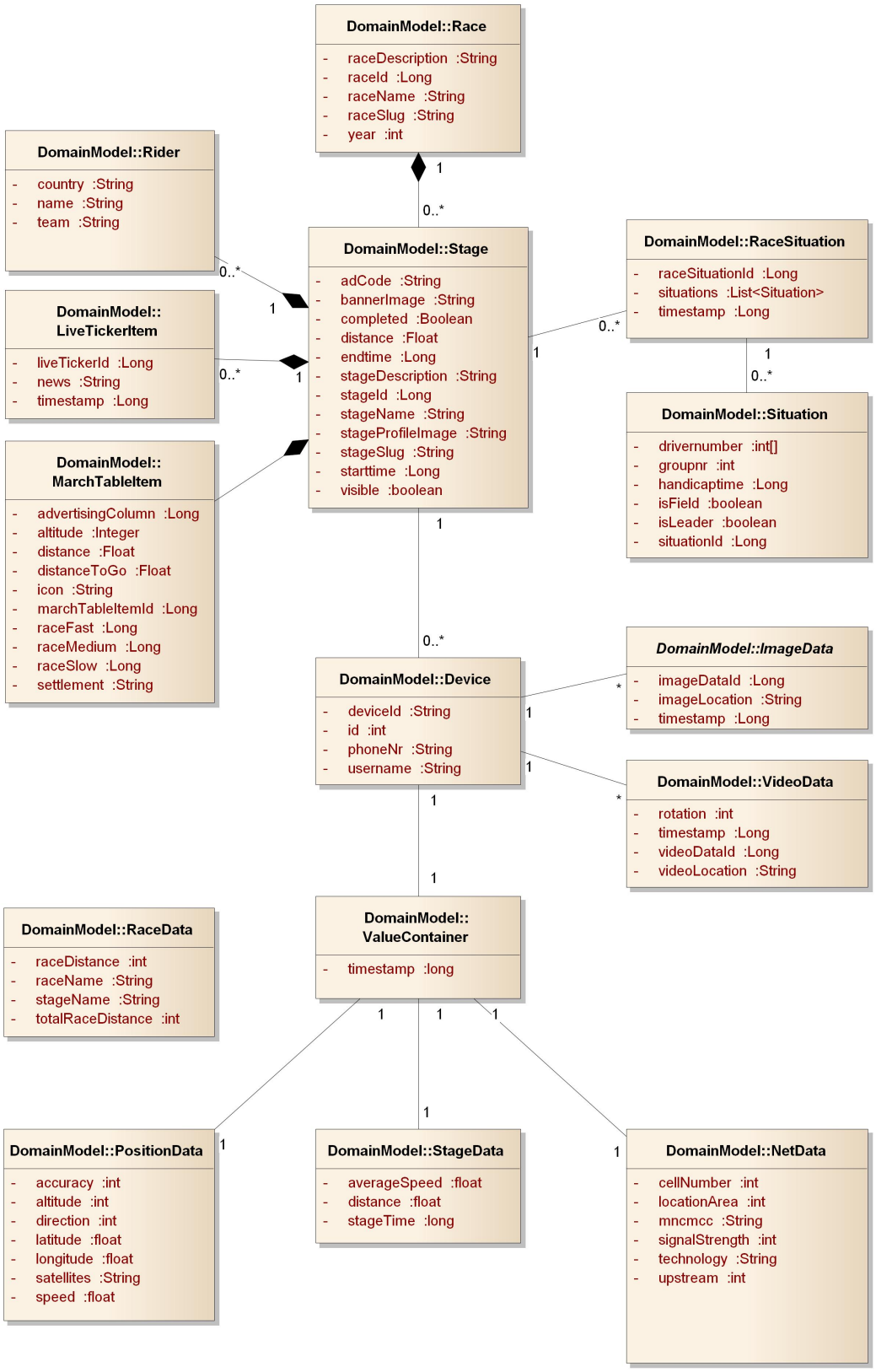


Abbildung 2.1: Domain Model des TourLive Servers

2.2 Software Design

Dieser Abschnitt behandelt die technische Umsetzung der Spezifikationen zum Endzustand. Nach einer kurzen Architekturübersicht wird jede Teilkomponente einzeln erläutert.

2.2.1 Architektur und Übersicht

Der TourLive Server übernimmt zwei grundsätzlich Funktionen, zum einen die Präsentation der Daten und zum anderen die Schnittstelle (*API*) für die Aufnahmegeräte und Drittentwickler.

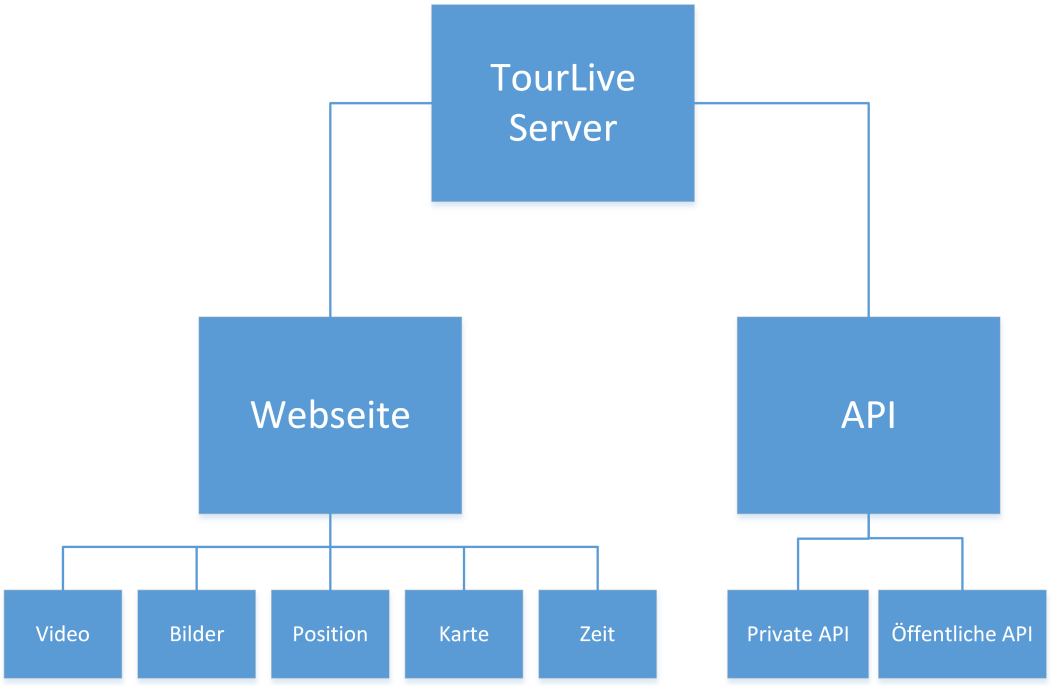


Abbildung 2.2: Grobstruktur des TourLive Servers

Diese Aufteilung wurde bewusst so gewählt, da es die Möglichkeit offen lässt, die Dienste auf mehrere Server aufzuteilen. Aus dem Requirements Engineering kam hervor, dass das System auch unter grosser Last für die Datenerfassung stets zur Verfügung stehen muss. In der Entwicklungsphase wurde der Einfachheit halber darauf verzichtet, das System auf mehrere Server aufzuteilen.

2.2.2 Schichtenmodell und Paketdiagramm

Ein Java Spring Projekt legt eine Struktur vor, wie eine Webapplikation aufgebaut werden soll. Dadurch werden gute Programmierpraktiken gefördert und Standards erzeugt. Auch beim TourLive Server wurden diese Vorgaben angewendet.

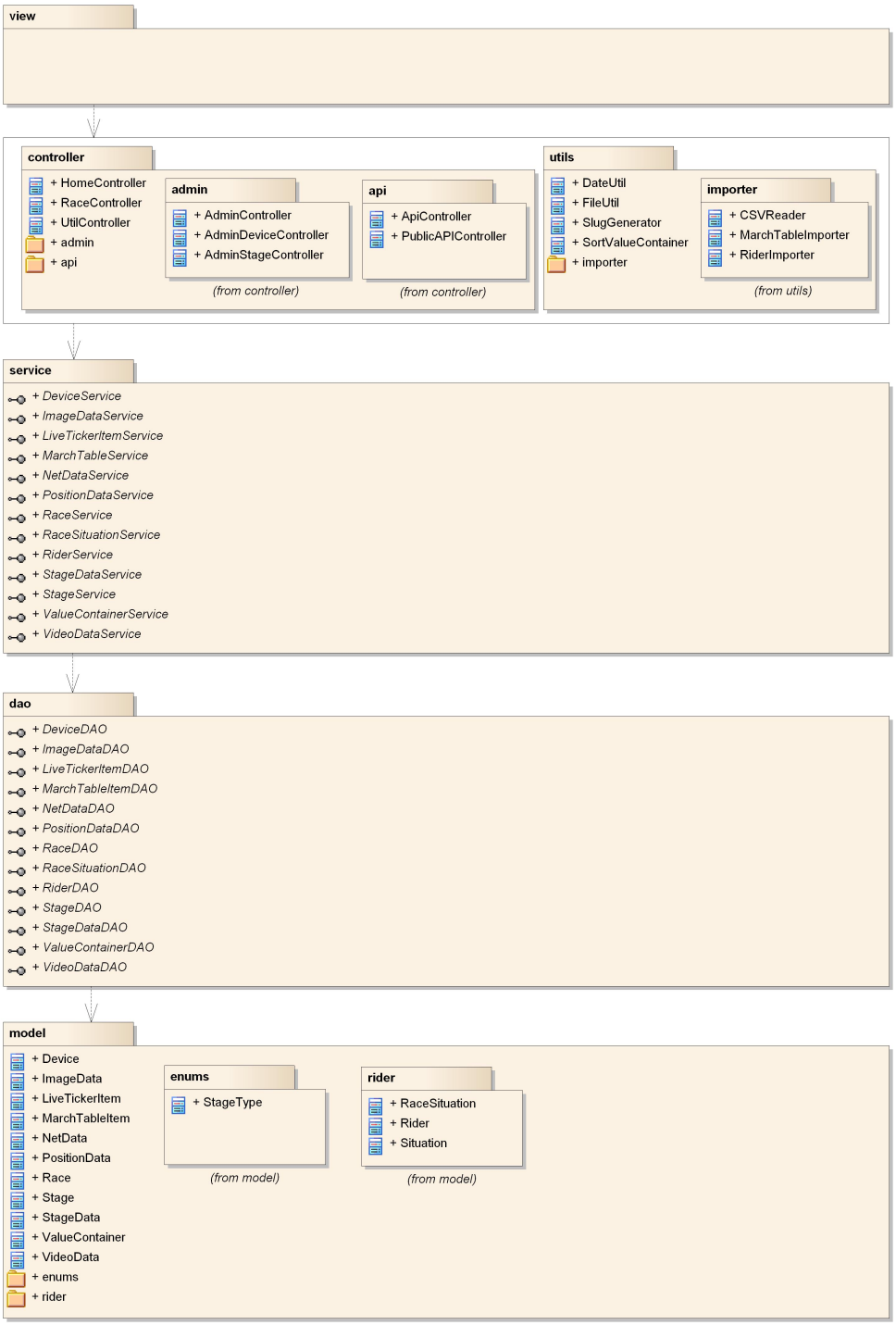


Abbildung 2.3: Paketdiagramm des TourLive Servers

Im Paketdiagramm 2.3 sind die Schichten der Applikation zu erkennen. So wurde das Domainmodel im Paket *Model* abgebildet, die Datenzugriffsobjekte im *DAO* Paket und die dazugehörige Service Schicht im Paket *Service*. Die Anfragen werden durch die Zugriffscontroller im Paket *Controller* bearbeitet. Um wiederkehrende Tasks zu zentralisieren, wurden diese im Paket *Utils* zusammengefasst, so z.B. das Formatieren von Zeiten und Daten.

2.2.3 Video

Eine Anforderung war es, eine Lösung zu erarbeiten, welche es ermöglicht, mit den Aufnahmegeräten Video-Streams aufzuzeichnen und diese an den Server zu übermitteln. Dazu gibt es verschiedene Ansätze mit entsprechenden Vor- und Nachteilen. Im TourLive System im Einsatz ist eine eigene Entwicklung, welche genau auf die Anforderungen des Industriepartners angepasst ist. Die Geräte nehmen Videosequenzen auf und übertragen diese an den Server. Der Server konvertiert diese Sequenzen vom mp4 in das ogg Format. Dazu wird die externe Library Xuggler[7] verwendet. Mit diesem Schritt sind die Browser Google Chrome, Mozilla Firefox und Microsoft Internet Explorer fähig, die Videosequenzen abzuspielen. Zusätzlich wird ein VideoData Objekt (vgl. Abbildung 2.1) in der Datenbank angelegt, welches den Aufnahmezeitpunkt, die Geräte ID, eine allfällige Rotation und den Pfad zur Videodatei enthält. Wenn nun ein Gerät einer Etappe zugewiesen ist und Videosequenzen vorhanden sind, wird die aktuellste Sequenz ausgeliefert. Der Videoplayer meldet, wenn die Videosequenz beendet ist und lädt, falls möglich, die nächste Sequenz via AJAX automatisch nach; dies wird im Quellcode 2.1 dargestellt. Falls kein neues Video verfügbar ist, wird nach 8 Sekunden erneut versucht ein Video zu laden. Dadurch wird auch bei einem temporären Ausfall der Aufnahmegeräte die Videowiedergabe fortgesetzt, sobald diese wieder verfügbar ist.

```
1  var videoPlayer1 = document.getElementById('video1');
2  videoPlayer1.addEventListener('ended', function(){
3    loadNext(videoPlayer1);
4  });
5
6  function loadNext(videoPlayer){
7    $.ajax({
8      type : "POST",
9      dataType: "json",
10     url : "/meineetappe/nextvideo",
11     data : {
12       deviceId : 'meinedeviceid',
13       afterId: videoPlayer.id,
14     },
15     success : function(data) {
16       if (data){
17         $('#mp4').attr('src', 'http://media.tourlive.ch/'+ data.
               videoLocation + '.mp4');
18         $('#ogg').attr('src', 'http://media.tourlive.ch/'+ data.
               videoLocation + '.ogg');
19         videoPlayer.id= "video" + data.videoDataId;
20         videoPlayer.load();
21       } else {
22       // try again in 8s
23         window.setTimeout(function(){loadNext(videoPlayer)},8000);
24       }
25     }
26   });
```

Quellcode 2.1: Automatisches Nachladen von Videosequenzen

7. Xuggler, `http://www.xuggle.com/xuggler`, aufgerufen am 04.06.2013

2.2.4 Bilder

Die Aufnahme der Bilder funktioniert nach einem ähnlichen Muster. Die aufgezeichneten Bilder werden auf dem Server abgelegt und ein ImageData Objekt in der Datenbank erzeugt. In der Klasse *ImageData.java* werden, wie schon beim Video, der Aufnahmezeitpunkt, die Geräte ID, sowie der Bildpfad gespeichert. Wie in Abbildung 2.4 zu sehen ist, wird beim Aufruf einer Seite das aktuellste verfügbare Bild pro Gerät angezeigt und mit dem Aufnahmezeitpunkt beschrieben.

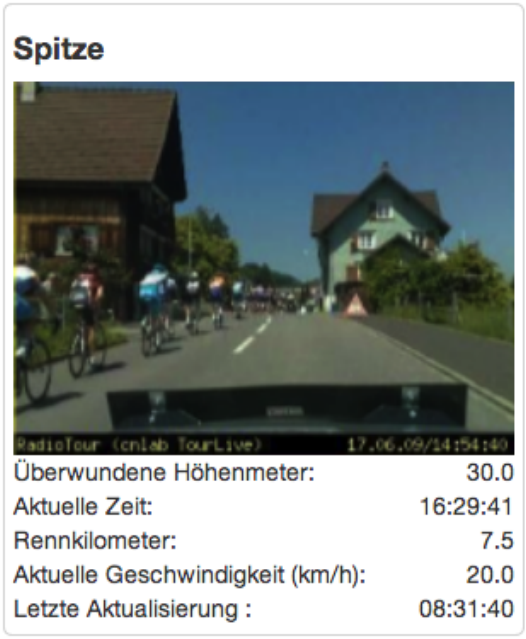


Abbildung 2.4: Aufgezeichnetes Bild mit Informationen ergänzt

2.2.5 Positionsdaten

Die Positionsdaten werden in Form eines ValueContainers gespeichert und übertragen. Ein beispielhafter ValueContainer ist im Kapitel 5.1.1 in der Abbildung 5.2 dargestellt.

Der Container setzt sich aus den drei Objekten *netData*, *positionData* und *stageData* (vgl. Abbildung 2.1) zusammen. Hinzu kommt zu jedem Container das *Device* Objekt sowie der aktuelle Zeitpunkt als Timestamp. Daher auch der Name ValueContainer, weil darin verschiedene Daten gesammelt und übermittelt werden.

Die ValueContainer bilden das Herz der Informationsquelle. Alleine durch Sortieren nach dem Feld *distance* im Objekt *stageData* wird festgestellt, welches Gerät an der Spitze mitfährt und wieviel Prozent der Gesamtstrecke zurückgelegt wurde. Da die ValueContainer nicht einer Etappe sondern einem Gerät zugeordnet sind, können auch nachträglich noch Geräte einer Etappe hinzugefügt oder entfernt werden. Um die Positionsdaten für eine Etappe zu erhalten, werden alle ValueContainer zwischen Startzeitpunkt und Endzeitpunkt dieser Etappe herausgefiltert und mit der Geräte ID eines Gerätes, welches der Etappe zugeordnet ist, ausgewählt. Um ein vergangenes Rennen wieder abzuspielen, wird die zeitliche Grenze, welche durch die Etappenendzeit gegeben war, durch einen beliebigen Zeitpunkt (> Startzeit) ersetzt. Dadurch werden immer nur diejenigen Container ausgewählt, welche bis zum angezeigten Zeitpunkt aufgezeichnet wurden.

Um die Daten zu visualisieren, werden verschiedene Elemente verwendet. Der geografische Verlauf der Strecke wird auf einer Karte eingezeichnet. Jedes Aufnahmegeräte (Klasse *Device.java*) hat zusätzlich ein Feld *color*, welches die Farbe auf der Karte bestimmt.

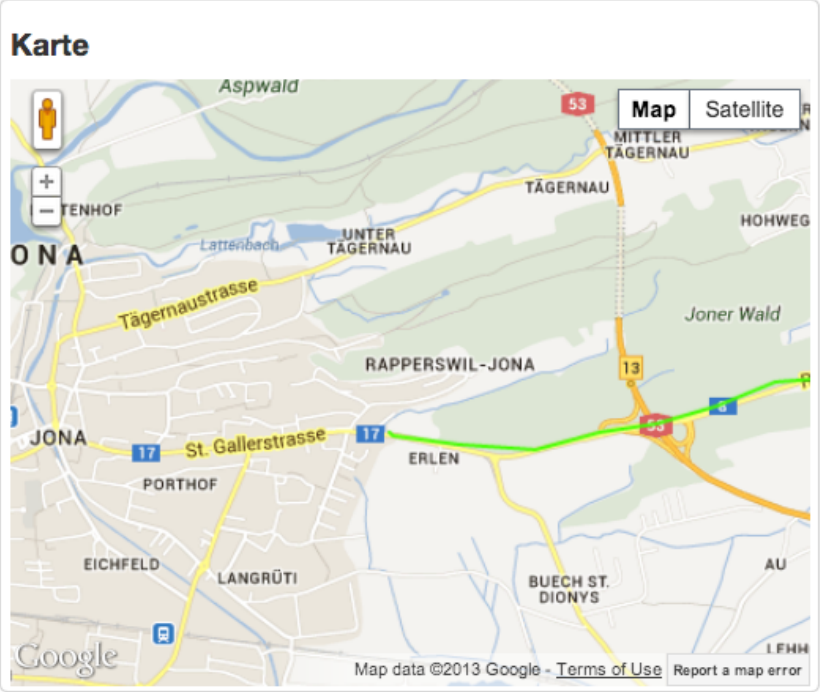


Abbildung 2.5: Kartenausschnitt mit zurückgelegter Strecke eines Gerätes

Ein Aufnahmegerät fährt in der Regel an der Spitze mit, ein zweites hinter dem Feld. Die Spitze setzt sich im Laufe des Rennens ab und der Abstand wird grösser. Die Entwicklung dieses Abstands ist für Radsportbegeisterte sehr spannend zu beobachten. Diese Berechnung wird ebenfalls durch die ValueContainers ermöglicht. Sobald mehr als ein Gerät einer Etappe zugeordnet ist, beginnt die Berechnung des Abstandes nach dem folgenden Algorithmus:

> *Für jeden ValueContainer in einer Etappe mache Folgendes: Suche pro Gerät den ValueContainer, welcher die nächsttiefere Etappendistanz im Vergleich zum obigen Container aufweist. Falls dieser Distanzunterschied kleiner als eine definierte Grösse ist (Standardwert 500m) so wähle den Container mit der kleinsten Zeit. Vergleiche diese Zeit mit dem ursprünglichen Container. Ist die Zeit des ursprünglichen Containers grösser, so macht die Differenz der Zeiten den Rückstand für diesen Container, andernfalls ist der ursprüngliche Container zu dem Zeitpunkt an der Spitze*

Diese Berechnung ist sehr aufwändig und wird mit steigender Anzahl ValueContainers immer langsamer. Im lokalen Testumfeld mit weniger als 1000 ValueContainer werden keine Performanceeinbussen verzeichnet. Beim ersten Testlauf stellte sich jedoch heraus, dass diese Werte zwischengespeichert werden müssen, da die Liveberechnung zu lange dauern würde. Bei jedem Seitenaufruf werden nur die neu hinzugekommenen ValueContainer berechnet. Dadurch wird das System schneller,

je mehr Benutzer auf der Seite sind, da die Anzahl neu zu berechnender Werte immer kleiner wird. Im Administrationsbereich kann die manuelle Berechnung aller ValueContainer neu gestartet werden. Dieser Vorgang ist notwendig, wenn nachträglich ein weiteres Gerät zur Etappe hinzugefügt wird.

Sämtliche Rückstände werden in einer Grafik (vgl. Abbildung 2.6) dargestellt. In der X-Achse sind die zurückgelegten Rennkilometer und in der Y-Achse die Rückstände in Sekunden. Für das Aufnahmegerät an der Spitze ist diese Kurve komplett flach, daher wird sie weggelassen.

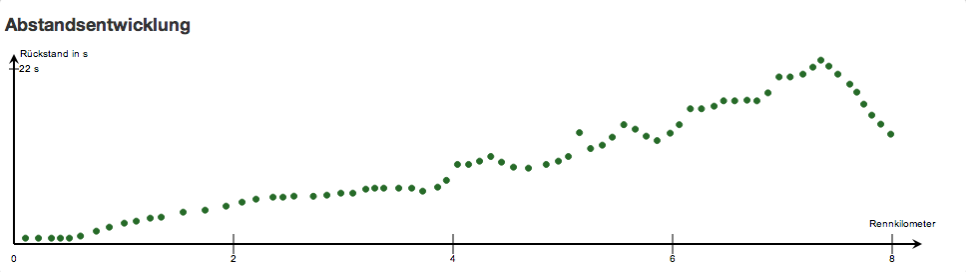


Abbildung 2.6: Abstandsentwicklung am Ende eines fiktiven Rennens

2.2.6 Private API

Die Schnittstelle zwischen den Aufnahmesystemen und dem TourLive Server wird als private *API* bezeichnet. Diese Schnittstelle wird genauer im Kapitel 5.1.1 erklärt.

2.2.7 Öffentliche API

Für die aufgezeichneten Daten wurde eine öffentliche Schnittstelle erstellt. Drittentwicklern wird es ermöglicht, die Positionsdaten in ihren Anwendungen zu verwenden und sowohl die Bilder als auch die Videosequenzen abzurufen. Die Schnittstelle ist zum jetzigen Zeitpunkt uneingeschränkt nutzbar. Für den produktiven Betrieb von TourLive wäre eine Authentifizierung für die Benutzung der Schnittstelle wünschenswert; eine solche ist aber nicht Teil dieser Arbeit. Die öffentliche Schnittstelle wird im Kapitel 5.2 detailliert erläutert.

2.2.8 XML basierte Konfiguration

Die Konfiguration in Spring wird einerseits über Java Annotationen vorgenommen, wie sie beim TourLive Server z.B. bei den Controllern zum Einsatz kommen, andererseits werden Servlet und Datenbankeinstellungen sowie sämtliche Abhängigkeiten zu anderen Libraries in XML Dateien konfiguriert. Die wichtigsten Dateien werden im Folgenden aufgezeigt.

web.xml

Im web.xml wird zugewiesen, welches Java Servlet welche Ressource ansteuert. Weiter werden die Filter definiert, welche angewendet werden sollen. Filter erlauben es, Anfragen zu bearbeiten, zu verändern oder gar umzuleiten. Im TourLive Server werden zwei Filter verwendet. Einerseits der Sitemesh Filter, welcher wiederverwendbare Elemente (z.B. Menu oder Footer) in den *JSP (JavaServer Pages)* Seiten auslagert. Andererseits schützt der zweite Filter den Administrationsbereich

vor nicht authentifiziertem Zugriff. Bei diesem Security Filter kann eine Ressource komplett geschützt werden. Die weitere Konfiguration findet in der security.xml Datei statt.

tourlive-servlet.xml

Damit die von den Controllern erstellten Models auch auf den *JSP* Seiten abgebildet werden können, wird ein ViewResolver verwendet. Dieser beschreibt, wo sich die für die Darstellung benötigten *JSP* Seiten befinden. Weiter werden die Internationalisierung und die Auslieferung von statischen Ressourcen - wie Bilder und StyleSheets - in der *tourlive-servlet.xml* Datei definiert.

hibernate.xml

Das Datenbankmapping geschieht mit Hibernate. In der hibernate.xml Datei werden die Datenbankeinstellungen vorgenommen. Zusätzlich wird hier die SessionBean deklariert - damit werden die Abfragen auf der Datenbank letztendlich ausgeführt.

root-context.xml

Im Root Context wird definiert, dass die weitere Konfiguration durch Java Annotationen interpretiert werden soll. Weiter werden die Konfigurationen der Datenbank, sowie umgebungsspezifische Einstellungen importiert. Das Pendant zum Root Context für die Testumgebung ist der Test Context.

security.xml

Die Sicherheitsparameter, welche von Spring geladen werden, sind in der security.xml Datei definiert. So kann eine Login- und eine Logout-Seite angegeben werden. An dieser Stelle wird auch definiert, dass sämtliche Zugriffe auf die Administrationsseiten nur nach Authentifizierung stattfinden können.

Die TourLive Server Anwendung verfügt nur über zwei Benutzergruppen: nicht authentifizierte Besucher und Administratoren. Daher wurde auf eine aufwändige Benutzerverwaltung verzichtet und nur ein Administrationsbenutzer fest in der XML Datei definiert.

2.3 Realisierung

Der für die Öffentlichkeit sichtbare Teil dieser Arbeit besteht aus den aufbereiteten Daten. Auf der Startseite wird jeweils die aktuellste Etappe angezeigt. Alle anderen Rennen können über die Navigation im Kopfbereich erreicht werden. In der Abbildung 2.7 ist ein fiktives Beispiel einer Etappe der Tour de Suisse 2013 dargestellt.

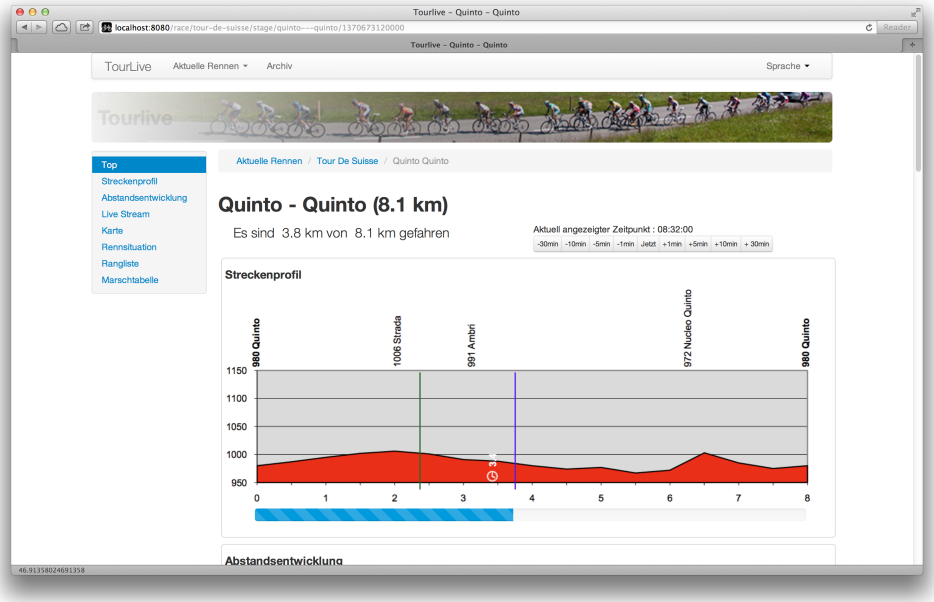


Abbildung 2.7: TourLive Server Webseite anhand eines Beispiels

2.3.1 Streckenprofil

Die aktuellen Positionen der Aufnahmegeräte werden mittels eines HTML5 Canvas direkt auf das Etappenprofil gezeichnet. Dazu wird die JavaScript Library *Raphaël JS*[8] verwendet. Zu jedem Gerät wird der aktuellste ValueContainer gesucht und daraus die Etappendistanz gelesen. Diese Distanz wird dann relativ zur Seitenbreite des Browserfensters als feine Linie mit der Farbe des Gerätes eingezeichnet, wie in der Abbildung 2.7 zu sehen ist. Gleich unterhalb dieser Grafik befindet sich eine Statusanzeige [blau]. Sie zeigt die zurückgelegten Kilometer in dieser Etappe an und dient gleichzeitig auch als Navigation. Durch klicken an einer beliebigen Stelle im blauen Balken kann man zu dieser Stelle (Rennkilometer) im Rennen zurückspringen. Das Pendant zu dieser Navigation sind die Schaltflächen im oberen Bereich, welche das Rennen über die Zeit navigieren lassen.

Gleich wie die Positionen der Aufnahmegeräte werden auch die Daten der Abstandsentwicklung mit der Raphaël JS Library dargestellt.

2.3.2 Marschtabelle

Für jede Etappe kann eine Marschtabelle importiert werden. Darin sind detaillierte Informationen über die Strecke der Etappe aufgeführt. Importiert wird eine *CSV (Comma Separated Values)* Datei über die Administrationsseite der Etappe. Folgende Spalten werden im *CSV* erwartet:

```
1 icon, altitude, distance, distanceToGo, settlement,
    advertisingColumn, raceSlow, raceMedium, raceFast
2 'icon' ist ein enum mit einem der folgenden Werte:
3 {<leer>, info, achtung, bahn, tunnel, verpflegung, rechts, links,
    bodenwelle}
```

Quellcode 2.2: Marschtabellen CSV Import

8. Raphaël JavaScript Library, `http://raphaeljs.com/`, aufgerufen am 01.06.2013

Die Klasse *MarchTableImporter.java* im Paket *utils* (siehe Paketdiagramm 2.3) importiert die Werte zeilenweise, setzt die Etappen-ID und speichert die Zeile als *MarchTableItem* in der Datenbank. Bei der Anzeige der Marschtabelle werden die Zeilen eingefärbt mit der Farbe des letzten Gerätes, welches an dieser Ortschaft vorbeigekommen ist. Dies wird aufgrund der Etappendistanz berechnet. Zudem wird die Tabelle zu dieser Zeile gescrollt, in welcher sich die Spitze aktuell befindet. Somit wird die Rennsituation auch in der Etappe farblich dargestellt, was das folgende Beispiel verdeutlicht.

Abbildung 2.8: Marschtabelle am Ende eines Rennens

2.3.3 Radrennfahrer

Wie schon bei der Marschtabelle können auch die Fahrer einer Etappe in Form einer *CSV* Datei über die Administrationsansicht importiert werden. Der TourLive Server erwartet die folgenden Spalten.

```
1 startNr, name, team, teamshort, country, official-time, deficite-
    time, virtual-time, neo
```

Quellcode 2.3: Fahrerliste CSV Import

Für den Import der Fahrer wird die selbe *CSV Reader Klasse* verwendet wie beim Import der Marschtabelle; die Zeilen werden aber mittels der RiderImporter Klasse interpretiert (siehe Paketdiagramm 2.3).

Alle importierten Fahrer können in der Fahrertabelle gefunden werden. Zusätzlich werden die Fahrer in Form einer schematischen Rennsituation (siehe Abbildung 2.9) dargestellt. Alle Fahrer die nicht einer Gruppe zugeordnet sind, werden automatisch dem Feld zugewiesen. Aufgrund ihrer Anzahl werden die Namen der Fahrer im Feld nicht einzeln aufgelistet, wie in Abbildung 2.9 zu erkennen ist.

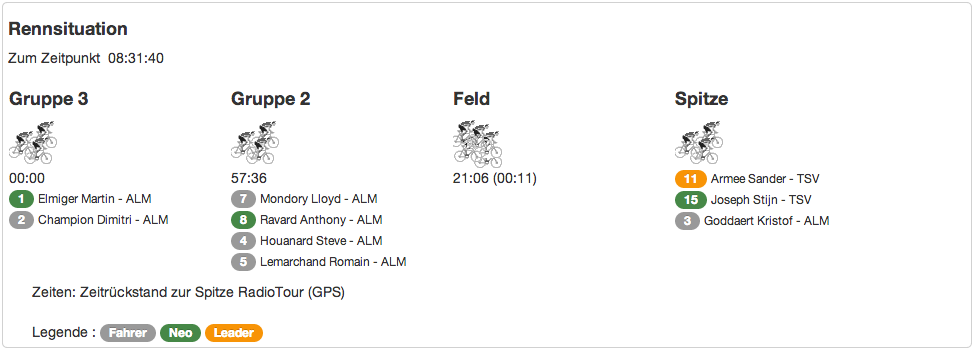


Abbildung 2.9: Rennsituation

3 Device Management Server

Im folgenden Kapitel werden die Systemkomponenten und deren Zusammenspiel erläutert. Für das Verständnis wird das Aufgabenumfeld in einem abstrakten Kontext dargestellt.

3.1 Software Analyse

3.1.1 Spezifikationen

Um die Anforderungen zu evaluieren wurde das bestehende Geräteverwaltungsportal analysiert. Da beim bestehenden System der Funktionsumfang stark eingeschränkt war, wurde vom Industriepartner zusätzliche Funktionalität gewünscht. Um eine kurze Übersicht über die vorhandenen Funktionen und erfassten Anforderungen zu geben folgt die Tabelle 3.1.

Spezifikation	**Altes System**	**Neues System**
Einstellungen anpassen	Nur Renndistanzkorrektur	Detaillierte Verwaltung der Geräteeinstellungen
Status der Geräte	Rudimentäre Anzeige des Gerätestatus	Detaillierte Anzeige des Gerätestatus
Alarming bei schlechtem Gerätezustand	Dauer seit letztem Positionsupdate / Bildempfang zu gross	Visuelle Hervorhebung bei Problemen mit dem Gerätestatus
Neustarten des Gerätes	Neustarten des Gerätes via Portal	nur Neustarten der App möglich
Gerätelog anzeigen	-	Gerätelog anzeigen
Versand von Nachrichten	-	Versand von Nachrichten möglich

Tabelle 3.1: Anforderungen Device Management Server

Eine detaillierte Beschreibung aller Anforderungen befindet sich im Anhang im Kapitel 7.5. Nachfolgend ein kurzer Überblick der wichtigsten Anforderungen.

3.1.2 Funktionale Anforderungen

Betriebsmodi

Die Geräte sollen sowohl über einen Management Server, analog zur bisherigen Verwaltungsseite, fernverwaltet als auch über ein „Einstellungen“-Menü direkt am Gerät konfiguriert werden können.

Daraus resultieren zwei Betriebsmodi: «managed» (TourLive App wird über den Device Management Server verwaltet) und «unmanaged» (TourLive App wird in den lokalen App Einstellungen verwaltet). Die Standardkonfiguration sieht den Modus «managed» vor. Die beiden Modi lassen sich am Gerät wie auch auf dem Device Management Server jederzeit ändern.

Alarming Funktionen

Treten Probleme auf, so soll auf dem Telefon sowie in der Management Konsole darüber informiert werden. Als zu meldende Probleme gelten folgende:

- Smartphone Akkustand liegt unter 30%
- Smartphone wird nicht mehr geladen (Stromzufuhr unterbrochen)
- Keine Medien-Daten seit mehr als 3 Minuten
- Keine Positions-Daten seit mehr als 3 Minuten
- Keine Status Updates seit mehr als 3 Minuten
- Weniger als 30% freier Speicher intern und extern
- Keine GPS-Satelliten verfügbar sind

3.1.3 Technologien

Für die Entwicklung des Device Management Servers wurden die selben Technologien verwendet, die bereits für den TourLive Server evaluiert wurden. Mehr Informationen dazu im Kapitel 2.1.2 und 2.1.3.

3.1.4 Domain Model

Abbildung 3.1 zeigt das Domain Model des Device Management Servers. Das Domain Model kann grundsätzlich in vier Kategorien unterteilt werden. Jede dieser vier Kategorien wird jeweils einem Gerät (der Klasse *Device.java*) assoziiert. Die vier Kategorien sind:

- Die Klasse *TourLiveLog.java* (1) mit sämtlichen Gerätelogs in Form der Klasse LogEntry.
- Die Klasse *DeviceManagementContainer.java* (2), die die Geräteeinstellungen auf die Klassen *DeviceSettings.java*, *RecordingSettings.java* und *AdditionalSettings.java*, aufteilt.
- Die Klasse *StatusData.java* (3), die den aktuellen Gesundheitszustand eines Aufnahmegerätes widerspiegelt.
- Die Klasse *Message.java* (4), die allfällige Nachrichten vom Device Management Server an ein Aufnahmegerät speichert.

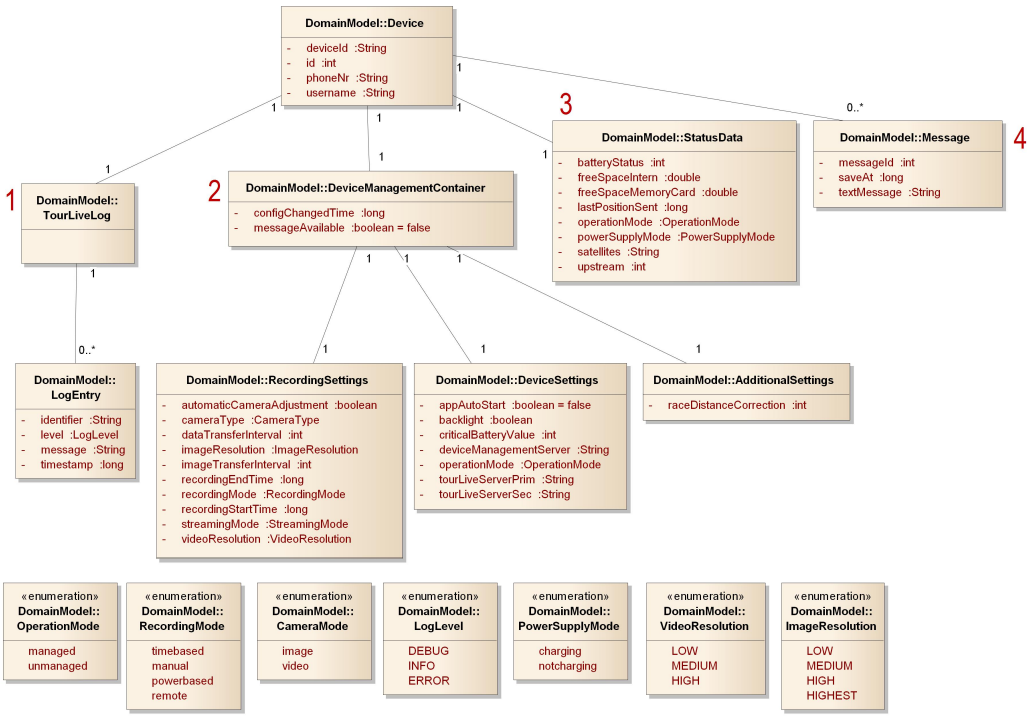


Abbildung 3.1: Domain Model des Device Management Servers

3.2 Software Design

3.2.1 Architektur und Übersicht

Der Device Management Server unterteilt sich in Webseite und API. Die API wird von den Aufnahmesystemen angesprochen wohingegen die Webseite hauptsächlich von Administratoren genutzt wird.

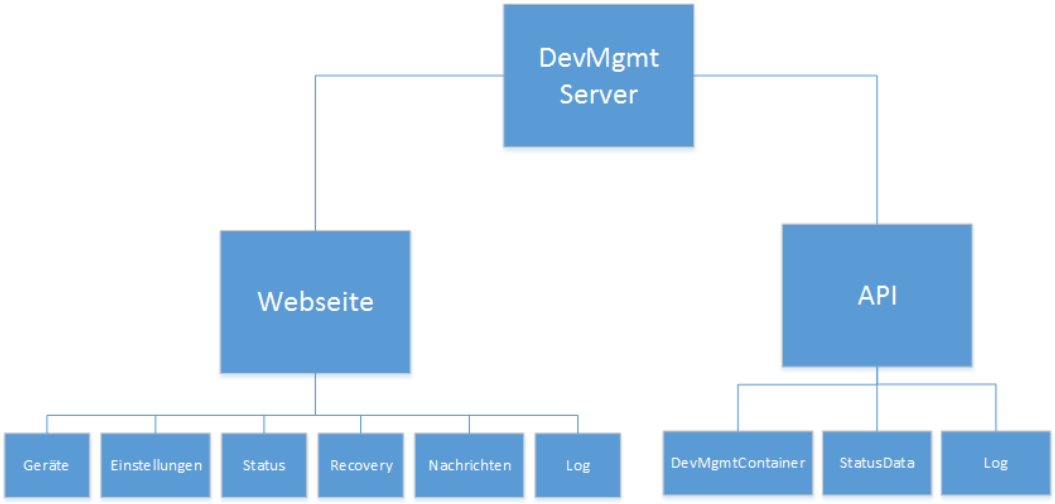


Abbildung 3.2: Grobstruktur des Device Management Servers

Wie beim TourLive Server besteht dank der gewählten Struktur die Möglichkeit, die

Webseite und die API auf verschiedene Server aufzuteilen. Da die Auslastung des Device Management Servers gering sein wird, wird darauf verzichtet.

3.2.2 Schichtenmodell und Paketdiagramm

Der Device Management Server ist in vier Schichten unterteilt. Die Schichten wurden an die vorgegebene Struktur des Spring Frameworks angepasst. Abbildung 3.3 zeigt diese vier Schichten auf. Jede Schicht kann nur auf die darunterliegenden Schichten zugreifen.

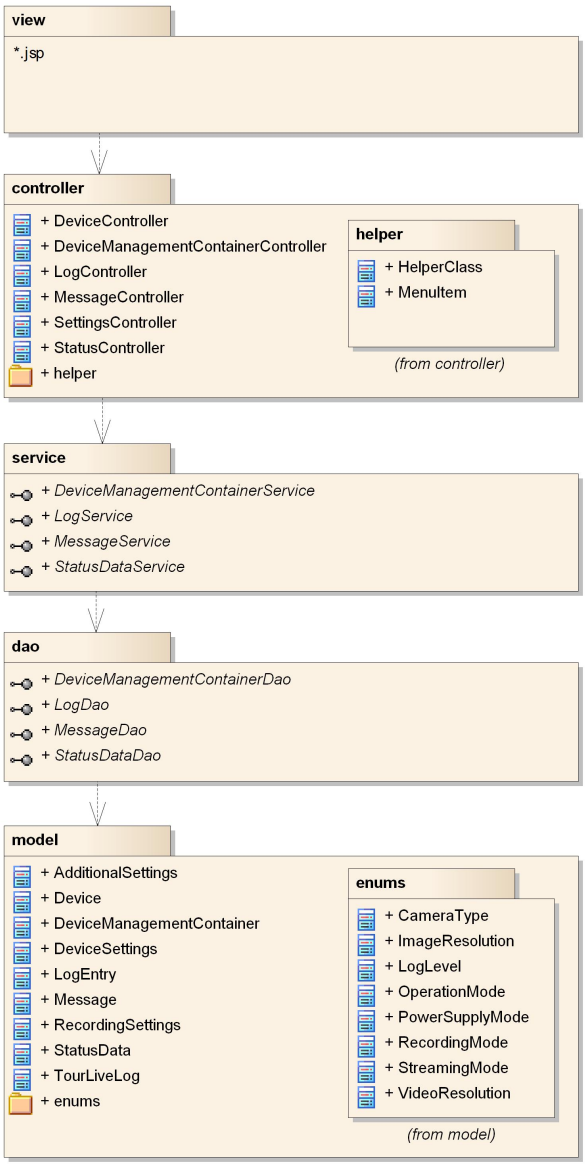


Abbildung 3.3: Schichtendiagramm des Device Management Servers

View

Die View Schicht enthält alle *JSP* Dateien, die für die Darstellung der Webseite benötigt werden.

Controller

Die Controller widerspiegeln die HTTP-Methoden (GET, POST,...), die für die Anzeige der Webseite und die RESTful JSON-Schnittstelle benötigt werden. Die Controller besitzen zwei verschiedene Methodentypen. Die einen werden für die Webseite zur Verfügung gestellt und die anderen für die Aufnahmesysteme. An den Methoden sieht man lediglich den Unterschied, dass bei den Webseitenmethoden ein ViewModel vorhanden ist, welches für die Anzeige gebraucht wird.

Service

Die Service Klassen sind für die Businesslogik auf den von den DAO's gelieferten Objekten verantwortlich.

Device Management Container Service

Der Device Management Container Service speichert und liefert die Container mit den Geräteeinstellungen sowie das Flag *'messageAvailable'* falls auf dem Device Management Portal eine Nachricht für das Gerät vorhanden ist.

DAO

Die DAO Schicht enthält die Datenzugriffsobjekte. Diese dient zur Entkopplung der Geschäftslogik vom Datenzugriff. Die Interfaces bilden die Schnittstelle. Die Implementierung kann so je nach Persistenztechnologie unterschiedlich sein, ohne dass die Geschäftslogik geändert werden muss.

Model

Die Domäne des Device Management Servers wird in der Model Schicht abgebildet. In den Objektinstanzen der Model Klassen sind die eigentlichen Daten gespeichert. Diese Objektinstanzen werden über den OR-Mapper in der SQL-Datenbank gespeichert.

3.3 Realisierung

Folgendes Kapitel beschreibt die Umsetzung des Device Management Servers.

3.3.1 Definition Requests

Die Requests werden mittels Spring Annotations definiert.

```
1 @RequestMapping(value = "/api/getdevmgmtcontainer", method =
    RequestMethod.POST)
2 @ResponseBody
3 public DeviceManagementContainer getDeviceManagementContainer(
    @RequestBody final StatusData request)
```

Quellcode 3.1: Spring Annotation

- ***@RequestMapping*:** gibt an, welche URL auf diese Methode gemappt wird und welche HTTP Request Methode erlaubt ist.

- ***@ResponseBody*:** definiert, dass die Methode einen HTTP Response Body zurück gibt und nicht eine *JSP* Seite rendert.
- ***@RequestBody*:** der Parameter der Methode wird als Body des Requests definiert.

Diese annotierten Methoden müssen in einer Klasse enthalten sein, welche als ***@Controller*** annotiert ist.

3.3.2 Webseite

Die Webseiten wurden mit *JSP* umgesetzt und mit der JavaScript Library JQuery angereichert. Um die grafische Oberfläche möglichst einfach zu gestalten, wurde Twitter Bootstrap[1] benutzt.

Die Webseite wird in drei Teile unterteilt: Header, Footer, sowie den Hauptbereich.

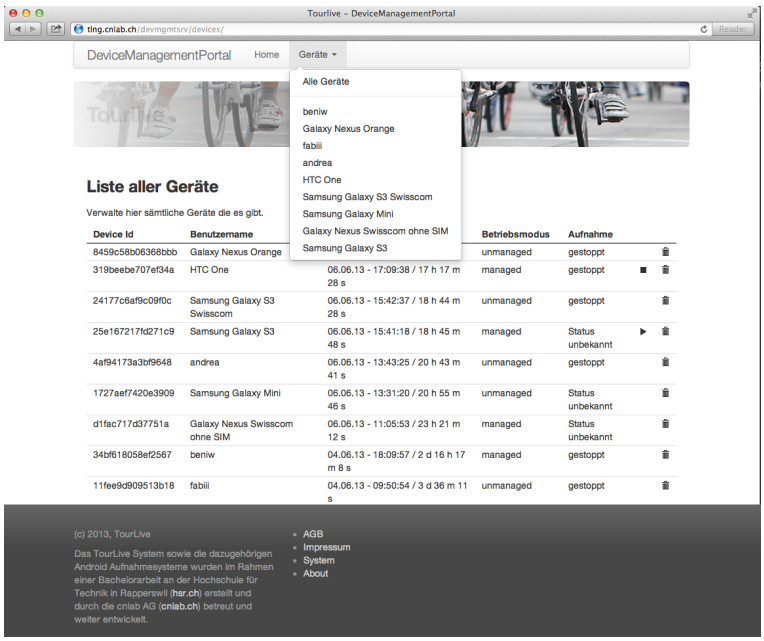


Abbildung 3.4: Übersicht der Device Management Server Webseite

Header

Der Header bietet die Möglichkeit, über eine Schnellnavigation auf ein Gerät zuzugreifen.

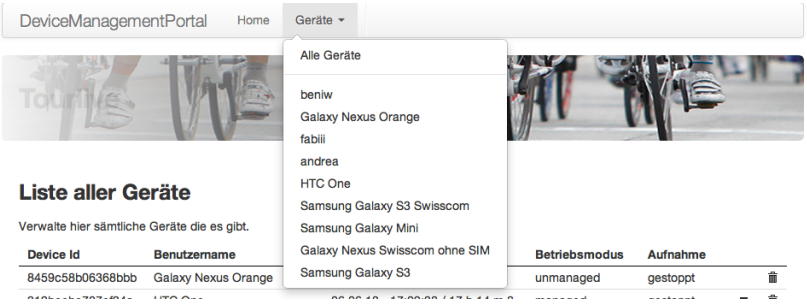


Abbildung 3.5: Device Management Server Webseite - Header

1. Twitter Bootstrap, `http://twitter.github.io/bootstrap/`, aufgerufen am 25. Mai 2013

Footer

Der Footer enthält detaillierte Informationen zum Projekt TourLive.

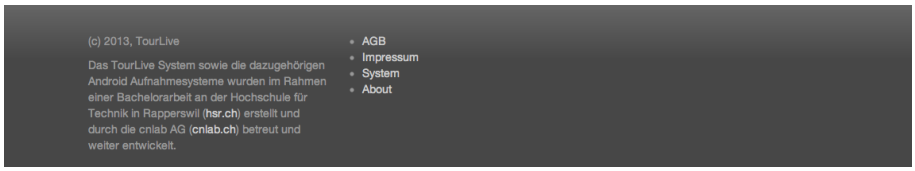


Abbildung 3.6: Device Management Server Webseite - Footer

Hauptbereich

Der Hauptbereich ist in zwei Teile unterteilt. So befindet sich Links eine Navigation, mit welcher zwischen den verschiedenen Funktionen navigiert werden kann.

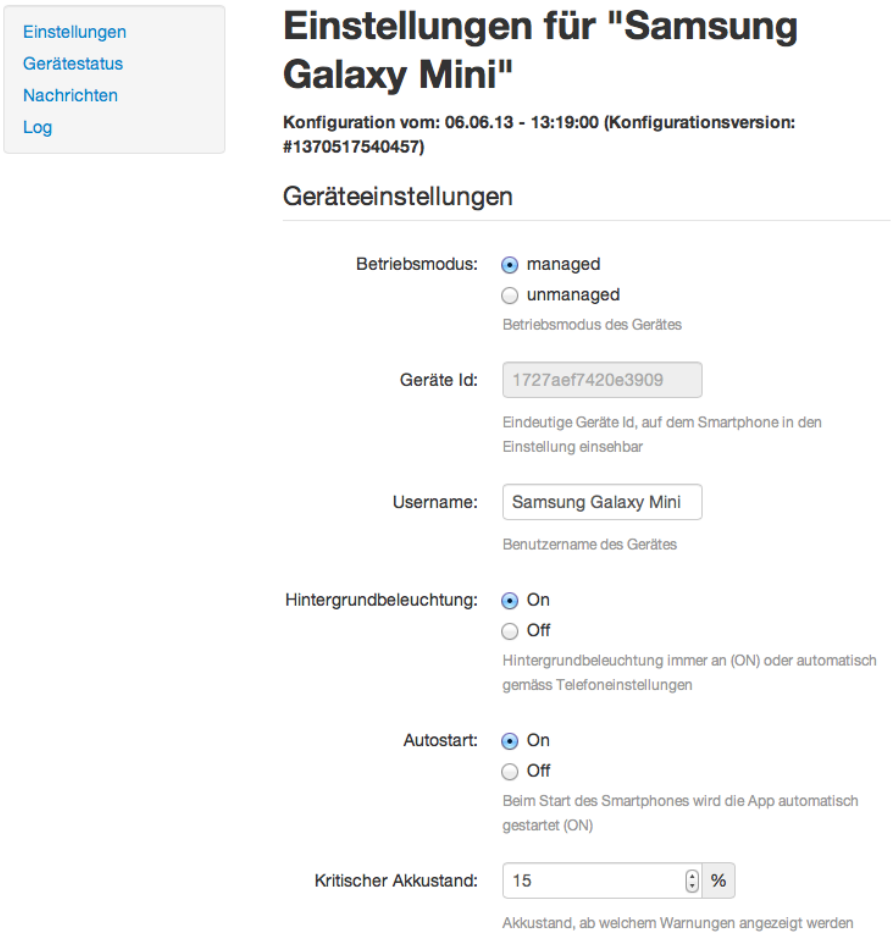


Abbildung 3.7: Device Management Server Webseite - Hauptbereich

4 Android Aufnahmesystem

Folgendes Kapitel behandelt das auf Google Android basierende Aufnahmesystem. Das Aufnahmesystem besteht aus zwei separaten Android Applikationen mit der Bezeichnung *TourLive.apk* sowie *TourLiveRecoveryService.apk*. Das Augenmerk folgender Unterkapitel liegt auf den Spezifikationen und der technischen Umsetzung des Aufnahmesystems und ist für Entwickler gedacht, die das Aufnahmesystem erweitern möchten.

4.1 Software Analyse

Dieses Kapitel beschreibt die Anforderungen an das Aufnahmesystem und gibt eine grobe Übersicht über die vorhandene Funktionalität.

4.1.1 Spezifikation

Um die Anforderungen zu evaluieren wurde das bestehende Aufnahmesystem auf Basis von Nokia Symbian analysiert. Alle vorhandenen Funktionen wurden erfasst und gemeinsam mit dem Industriepartner um weitere ergänzt. Eine kurze Übersicht über die bereits bestehenden Funktionen und den neu erfassten Anforderungen liegt in tabellarischer Form vor.

Spezifikation	**Altes System**	**Neues System**
Positionsdaten übertragen	Geschwindigkeit, Höhe, Richtung/Beschleunigung, Steigung, Longitude, Latitude	Geschwindigkeit, Höhe, Richtung, Steigung, Longitude, Latitude
Etappendaten	Zeit, Höhe, Distanz, Durchschnittliche Geschwindigkeit, UTC Zeit, Datum	Zeit, Höhe, Distanz, Durchschnittliche Geschwindigkeit
Tour Total	Zeit Total, Zeit Tour, Distanz Total, Distanz Tour, Höhe Total, Höhe Tour	-
Netzdaten übertragen	Zellen ID, Location Area, Signal, Akku, Netzwerk, Netzwerk ID	Zellen ID, Area, Signal, Netzwerk ID, Technologie, Datenrate

Bilder übertragen	wird unterstützt	wird unterstützt
Videostream übertragen	Einzelne Bilder wurden übertragen und serverseitig zu einem Stream zusammengefügt	Videosequenzen werden übertragen und serverseitig zu einem Stream zusammengesetzt
Lokales Caching	nur Bilder wurden gecacht	Sämtliche aufgenommene Daten werden lokal gespeichert
Aufnahmegerät Systemstatus übertragen	nur der Akkustand wurde an das Device Management Portal übertragen	Detaillierte Statusinformation an das Device Management Portal
Betriebsmodi	-	Managed - Einstellungen über das Portal / Unmanaged - Einstellungen am Gerät
Auto-Start der App	-	App wird beim Gerätestart automatisch gestartet
Externe Geräte	ODB (Onboard Diagnose Bus) und Pulsinfo Geräte wurden angesteuert	-
Betriebssystem	Symbian App	Android App
Log	Position und Bilder gesendet	Daten, Bilder, Status und Einstellungen gesendet / Exceptions
Aufnahmestart Modi	Aufnahme hat automatisch bei App Start gestartet	Manuell, Zeitbasiert, Fernverwaltet oder bei Aktivierung einer externen Stromquelle
Power Management	-	Bei niedrigem Akkustand wird ein Energiesparmodus aktiviert
Alarming Funktion	-	Treten Probleme auf, so wird darauf hingewiesen (z.B. keine GPS Daten während x Minuten)
Mehrsprachigkeit	-	Deutsch und Englisch, einfach erweiterbar

Fehlerkorrektur der GPS Daten	-	Ausreisser bei den GPS Daten werden herausgefiltert
Notfallwiederherstellung per SMS	wird unterstützt	wird unterstützt

Tabelle 4.1: Anforderungen Android Aufnahmesystem

Eine detaillierte Beschreibung aller Anforderungen befindet sich im Anhang im Kapitel 7.5.

4.2 Software Design

Dieser Abschnitt dokumentiert das Software Design, verwendete Technologien und die Architektur des Aufnahmesystems.

4.2.1 Verwendete Technologien

Android

Eine Voraussetzung für das Aufnahmesystem ist, dass dieses in Form einer Android Applikation entwickelt wird. Die native Programmiersprache für Android Applikationen ist Android Java, ein der Java Standard Edition sehr ähnliches Java Derivat. Die Programmierung in Java bringt den Vorteil, dass die gesamte Android-API benutzt werden kann. Da sämtliche Komponenten des TourLive Projektes in Java geschrieben sind, kann teilweise auf Server- wie auch Clientseite auf dieselben Klassen-Bibliotheken zurückgegriffen werden, was mögliche Inkompatibilitäten vermindert.

Eine Anwendung wird jeweils für eine konkrete Android Version (Minimum Required SDK) entwickelt. Im Rahmen dieses Projektes wird als minimale SDK Version Android 4.0 (Versionsname: Ice Cream Sandwich[1], API-Level: 14) verwendet. Die Applikation sollte gemäss Android Richtlinien kompatibel mit allen darauffolgenden Android Versionen sein.

4.2.2 Externe Libraries

Um den geforderten Funktionsumfang umzusetzen, wird auf folgende zwei externe Bibliotheken zurückgegriffen.

Spring for Android[2]

Ein Rest Client für Android von den Entwicklern des Java Spring Frameworks. Der Rest Client bietet die einfache Möglichkeit der Serialisierung / Deserialisierung von Java Objekten in JSON-Strings.

Verwendete Version: spring-android-1.0.1

1. Android Ice Cream Sandwich, `http://www.android.com/about/ice-cream-sandwich/`, aufgerufen am 29.04.2013
2. Spring for Android, `http://www.springsource.org/spring-android`, aufgerufen am 29.04.2013

ORMLite[3]

ORMLite ist eine OpenSource Java Library, welche das Object-Relational Mapping (ORM) übernimmt. ORMLite bietet eine speziell auf Android angepasste Distribution an.

Verwendete Version: ORMLite-4.45

4.2.3 Architektur des Aufnahmesystems

Die Android Applikation zeichnet Daten für den TourLive Server sowie den Device Management Server auf. In der Abbildung 4.1 wird veranschaulicht, welche Daten an welchen Server übertragen werden.

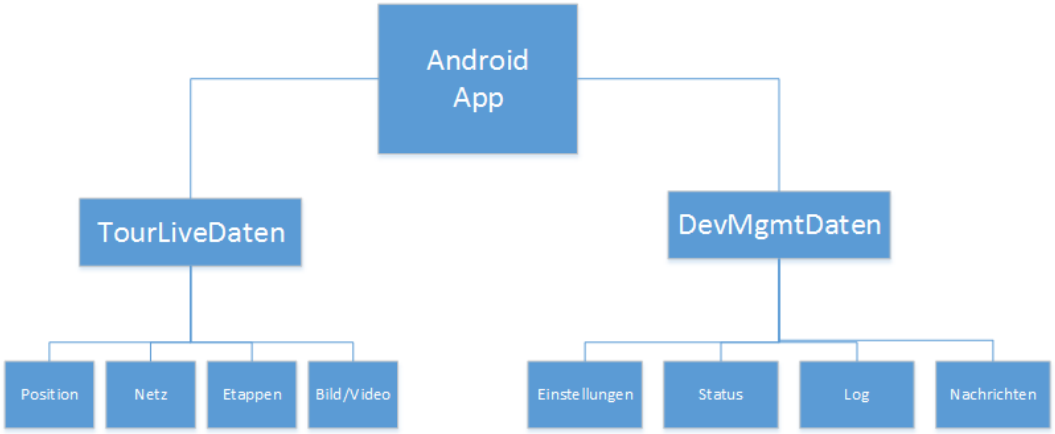


Abbildung 4.1: Grobstruktur der Android App

3. ORMLite, `ormlite.com`, aufgerufen am 02.05.2013

Schichtenmodell

Die Android Applikation teilt sich in 3 Schichten auf. Die oberste Schicht stellt die Schnittstelle zum Benutzer dar. Die Klassen in dieser Schicht sind für die Darstellung zuständig. Die mittlere Schicht enthält die ganze Businesslogik. Dazu gehören Services, Timer, Listener, etc. Die unterste Schicht enthält die Objekte, die die effektiven Daten (Positionsdaten, Konfigurationen, ...) enthalten.

Abbildung 4.2: Paketdiagramm des Aufnahmesystems

Domain Model

Das Domain Model des Aufnahmesystems setzt sich aus den Domain Models des TourLive Servers und des Device Management Servers zusammen. Im Aufnahmesystem kommen keine neuen Objekte hinzu.

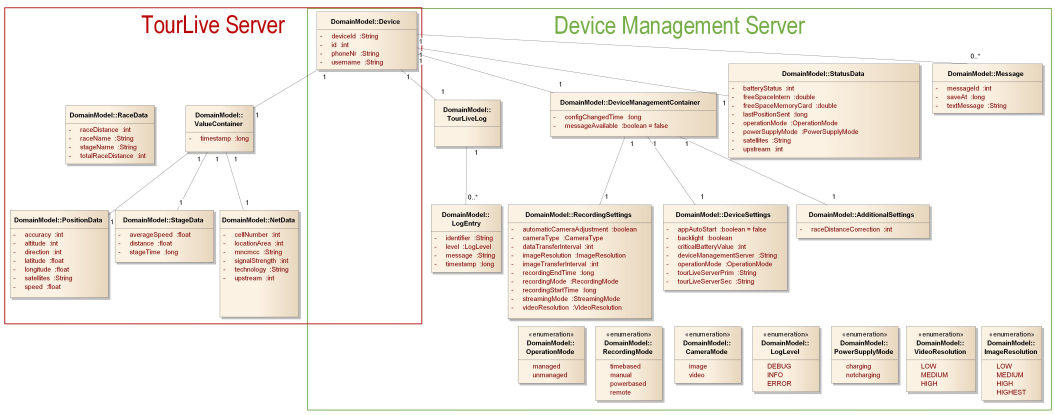


Abbildung 4.3: Aufnahmesystem Domain Model

Paketdiagramm

Die TourLive Applikation hat folgende Pakete. Als Grundstruktur wurde analog zum TourLive Server und dem Device Management Server *ch.hsr.ba.tourlive* verwendet. Die TourLive Applikation verwendet zusätzlich die Subkategorie *.android* sowie folgende applikationsspezifische Struktur:

Paket	**Inhalt**
.	die initialisierende TourLiveApplication.java Klasse sowie Config.java
.adapter	Daten-Adapter um Listen darzustellen
.adapter.row	individuelle Listen Einträge
.domain	projektübergreifende Domain Klassen
.domain.devmgmtsrv	Device Management Server spezifische Domain Klassen
.domain.devmgmtsrv.enums	Device Management Service spezifische Enums
.domain.tourliveserver	TourLive Server spezifische Domain Klassen
.domain.tourliveserver.enums	TourLive Server spezifische Enums
.helper	Hilfsklassen zur Berechnung von Werten
.listener	System Event-Listener
.receiver	Broadcast Receiver
.service	sämtliche Services
.task	sämtliche AsyncTasks
.task.timer	sämtliche TimerTasks
.view	Activities und Fragments
.view.helper	individuelle Dialoge
.view.helptext	die Hilfstext Klassen

Tabelle 4.2: Pakete in der Android Applikation

4.2.4 Architekturentscheide

Globaler Daten Provider - Repository.java

Das Repository.java implementiert das Singleton-Pattern und fungiert als globaler Daten Provider. Damit werden lange Aufrufsketten verhindert. Das Repository bietet auf folgende Daten Zugriff:

- diverse Services
- das TourLiveLog
- Liste mit ValueContainers
- diverse globale Domänen Objekte

Start der Applikation - TourLiveApplication.java

Beim Start der TourLive Applikation müssen diverse Daten initialisiert werden. Um diese Initialisierung vor der ersten View durchzuführen, wurde eine projektspezifische Klasse *TourLiveApplication.java* geschrieben, die von der Android Klasse *Application* ableitet. Eine von *Application* abgeleitete Klasse darf nur einmalig vorkommen und wird als erste Klasse bei einem App-Start initialisiert.

> *"An Application class is a base class for those who need to maintain global application state."*[4]

Über die Klasse TourLiveApplication kann ausserdem auf den globalen Application-Context zugegriffen werden, der zur Initialisierung diverser Objekte benötigt wird und wie folgt in den Android Developer Reference beschrieben ist.

> *"Interface to global information about an application environment. This is an abstract class whose implementation is provided by the Android system. It allows access to application-specific resources and classes, as well as up-calls for application-level operations such as launching activities, broadcasting and receiving intents, etc."* [5]

Caching der Daten

Um einen lokalen Cache zu realisieren wird der auf SQLite basierende OR-Mapper ORMLite verwendet. Der Cache verhindert Datenverlust bei Applikationsabstürzen. In der SQLite-Datenbank werden folgende Objekte gespeichert:

- Bildinformationen (Binärdatei liegt im Dateisystem)
- Videoinformationen (Binrädatei leigt im Dateisystem)
- Positions-, Etappen- & Netzdaten (ValueContainer)
- Geräteinformationen (Device)
- Log-Einträge

Um Objekte und ihre Attribute zu persistieren, werden diese in den Modelklassen mit Java Annotationen markiert.

4. Application class, `http://developer.android.com/reference/android/app/Application.html`, aufgerufen am 10.04.2013

5. Context class, `http://developer.android.com/reference/android/content/Context.html`, aufgerufen am 13.04.2013

Factory Pattern - DataCreator.java

Das Erstellen der ValueContainer, DeviceManagementContainer und anderen Datenstrukturen unterliegt einem relativ komplexen Vorgang, da unzählige Werte aus den verschiedensten Android Listener, Android Provider und Android Services zusammengezogen werden müssen. Durch das Factory Pattern ist es möglich, dies über ein einzelnes Objekt, den DataCreator, zu realisieren.

Primärer / Sekundärer TourLive Server - URLProvider.java

Eine Anforderung an das TourLive Aufnahmesystem ist die Implementation eines sekundären TourLive Servers, der bei Ausfall des primären Servers die Daten empfangen und verarbeiten kann. Dies wurde über die Klasse URLProvider.java realisiert. Schlägt die Verbindung zum aktuell ausgewählten Server, in der Regel ist dies der primäre Server, mehrmals fehl, stellt der URLProvider den Zweitserver zur Verfügung. Die Anzahl möglicher Fehlschläge vor einem Wechsel ist in der Config.java definiert. Sämtliche genutzten URLs werden über den URLProvider bezogen.

Unveränderbare Applikationskonfiguration - Config.java

In der *Config.java* werden unveränderbare Applikationseinstellungen gespeichert. Diese sind direkt über *public static final* Attribute aufrufbar. Die Config.java enthält folgende Konfigurationen:

- sämtliche API-URLs des TourLive Servers und des Device Management Servers
- Speicherort im Dateisystem der Bilddateien
- Speicherort im Dateisystem der Videodateien
- Intervall wie oft der Gerätezustand dem Device Management Server übertragen werden soll
- Angaben zur Genauigkeit die eine GPS Location haben muss, damit sie nicht verworfen wird

4.3 Realisierung

Das Kapitel Realisation gibt Hinweise zur Umsetzung des TourLive Aufnahmesystems. Es werden interne Abläufe und Funktionen erläutert und bietet eine kurze Einführung in die Funktionen des Ausnahmesystems.

4.3.1 Grafische Oberfläche

Einführung in die grafische Oberfläche von Android

Die grafische Oberfläche einer Android Applikation besteht in erster Linie aus Activities. Gemäss Android Developer Reference wird eine Activity wie folgt definiert.

> *"An activity is a single, focused thing that the user can do. Almost all activities interact with the user, so the Activity class takes care of creating a window for you in which you can place your UI."* [6]

6. Activity, `http://developer.android.com/reference/android/app/Activity.html`„ aufgerufen am 25.03.2013

Die TourLive Applikation wurde mit folgenden drei Activities realisiert.

- MainActivity.java
- SettingsActivity.java
- AboutActivity.java

Zu einer Activity gehört jeweils eine *.java Datei sowie eine *.xml Datei die das Layout definiert. Innerhalb einer Activity wird mit Fragments gearbeitet, welche wie folgt definiert sind:

> *"A Fragment represents a behavior or a portion of user interface in an Activity."* [7]

Fragments werden für die verschiedenen Tabs innerhalb der MainActivity verwendet, wie nachfolgende Screenshots zeigen.

Konkrete Umsetzung der grafischen Oberfläche

Die MainActivity, bestehend aus *MainActivity.java* und *main_activity.xml*, ist die Hauptansicht der Applikation. Sie ist in drei Teile unterteilt, Header, Footer und Hauptbereich. Header und Footer sind in allen Tabs sichtbar.

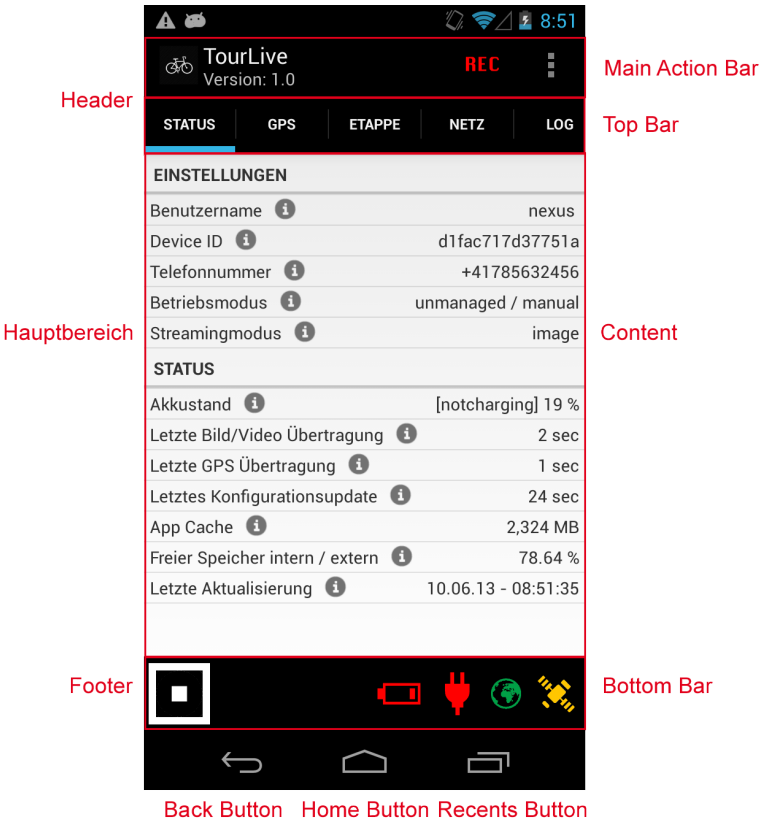


Abbildung 4.4: Aufnahmesystem MainActivity

7. Fragment, `http://developer.android.com/reference/android/app/Fragment.html`, aufgerufen am 25.03.2013

Hauptbereich

Der Inhalt des Hauptbereichs hängt vom selektierten Tab ab. Die einzelnen Ansichten Status, GPS, Etappe, Netz und Log wurden mit Fragments realisiert und basieren jeweils auf einer ListView. Eine ListView ist wie folgt definiert:

> *"ListView is a view group that displays a list of scrollable items. The list items are automatically inserted to the list using an Adapter that pulls content from a source such as an array or database query and converts each item result into a view that's placed into the list."* [8]

Header

Abbildung 4.5: Aufnahmesystem Header

Der Header beinhaltet in der *Action Bar* die aktuelle Versionsnummer, den Namen der Applikation sowie den Settings Button. Befindet sich die Applikation im Aufnahmemodus wird in roter Farbe der Text *REC* eingeblendet. Unterhalb der *Action Bar* befindet sich die *Top Bar*, in der die verschiedenen Tabs angezeigt werden. Innerhalb des Content Bereiches kann mit einem Swipe zwischen den verschiedenen Views hin und hergewechselt werden.

Settings Button

Über den Settings Button wird folgendes Menü eingeblendet:

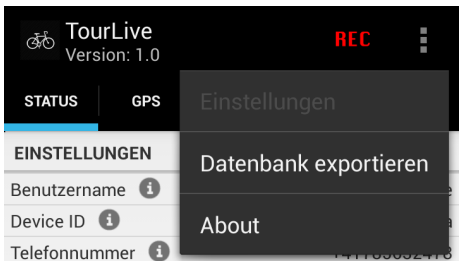


Abbildung 4.6: Aufnahmesystem Settings Button Clicked

Befindet sich die Applikation im Aufnahmemodus ist der Menüeintrag *Einstellungen* deaktiviert. Die restlichen Aktionen sind trotzdem verfügbar.

8. ListView, `http://developer.android.com/guide/topics/ui/layout/listview.html`,, aufgerufen am 25.03.2013

Footer

Der Footer zeigt den Status der App. Die Symbole werden je nach Gesundheitszustand der Applikation rot, gelb oder grün dargestellt. Welche Farbe welchem Gesundheitszustand entspricht, ist in den Anforderungen im Kapitel 7.5.2 beschrieben.

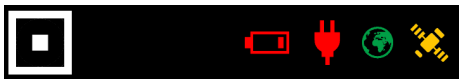


Abbildung 4.7: Aufnahmesystem Footer

Mit dem Recording Button links kann die Aufnahme gestartet und gestoppt werden. Der Button ist nur aktiviert, wenn der Aufnahmemodus auf manuell gesetzt ist.

Einstellungen

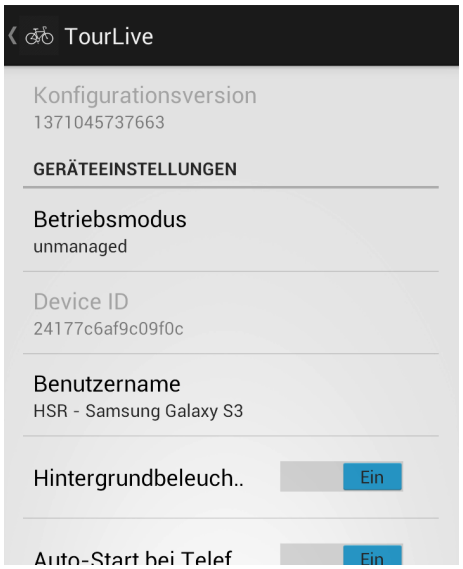


Abbildung 4.8: Aufnahmesystem Einstellungen

Der oben abgebildete Screenshot zeigt einen Ausschnitt der Aufnahmesystemeinstellungen. Sämtliche Einstellungen können auch am Device Management Server verwaltet werden und werden regelmässig synchronisiert. Folgende Einstellungen sind möglich:

- Konfigurationsversion (read-only)
- Betriebsmodus (managed / unmanaged)
- Device ID (read-only)
- Benutername (String)
- Hintergrundbeleichtung (on / off)
- Auto-Start bei Gerätestart (on / off)
- Kritischer Akku-Stand (0 - 100 %)
- TourLive Server primär (String)
- TourLive Server sekundär (String)

- Device Management Server primär (String)
- Aufnamemodus (manuell, strombasiert, ferngesteuert, zeitbasiert)
- Startzeit (DateTimePicker - nur bei Aufnahmemodus zeitbasiert)
- Endzeit (DateTimePicker - nur bei Aufnahmemodus zeitbasiert)
- Datenübertragunsintervall (Integer)
- Kamera automatisch ausrichten (on / off)
- Kamera (front / back)
- Streamingmodus (Bilder / Videos / Nichts)
- Bildauflösung (320x240, 640x480, 1280x960, 1600x1200)
- Bildübertragunsintervall (Integer)
- Videoauflösung (176x144, 720x480, 1280x720)
- Renndistanzkorrektur (Integer)

Es ist ausserdem möglich sämtliche lokal gespeicherten Etappen und Positionsdaten zu löschen, sämtliche Bilder und Videodaten zu löschen und die Telefonnummer beim Wechsel der SIM-Karte anzupassen.

4.3.2 Berechtigungen

Das Starten einer Android Applikation erfordert je nach Funktionalität der Applikation bestimmte Systemberechtigungen, die bei der Installation *Akzeptiert* werden müssen.

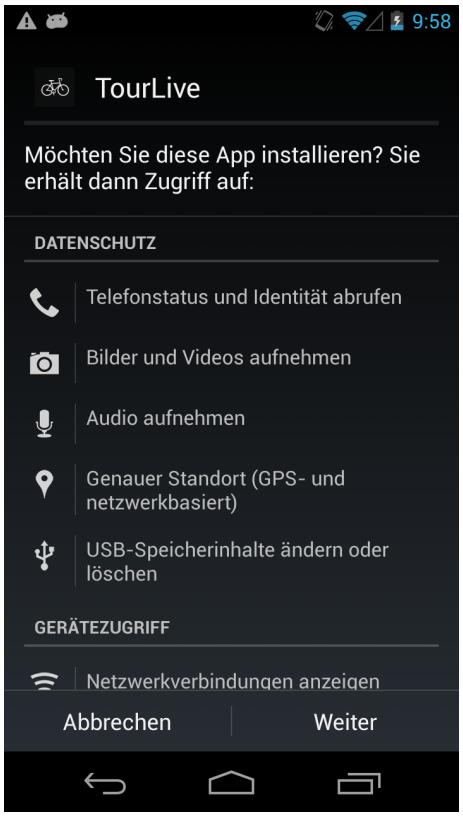


Abbildung 4.9: Aufnahmesystem Berechtigungen

Datenschutz

- **Telefonstatus und Identität abrufen:** Berechtigt das Auslesen der Telefonnummer
- **Bilder und Videos aufnehmen:** Berechtigt die Aufnahme von Bildens und Videos
- **Audio aufnehmen:** Berechtigt die Aufnahme von Audio
- **Genauer Standort:** Berechtigt die Nutzung des GPS Providers
- **USB Speicherinhalte ändern und löschen:** Berechtigt den Zugriff auf das Dateisystem (Bilder und Videos)

Gerätezugriff

- **Netzwerkverbindungen:** Berechtigt zur Nutzung des mobilen Netzwerks
- **Beim Start ausführen:** Berechtigt die Nutzung des Auto-Starts
- **Zugriff auf geschützten Speicher testen:** Berechtigt den Zugriff auf das Dateisystem (SQLite)

4.3.3 I18N

Die Applikation ist in deutscher sowie englischer Sprache verfügbar. Weitere Sprachen lassen sich einfach durch Erstellen und Übersetzen der Ordnerstruktur TourLive/res/values-SPRACHE hinzufügen.

4.3.4 Services, Tasks & TimerTasks

Die TourLive Applikation arbeitet mit mehreren Services. Diese implementieren in der Regel das eigens für diese Applikation entworfene Interface *TourLiveService.java*. Die Services werden in der Regel in Verbindung mit einem *Timer* genutzt, der wiederum einen *TimerTask* voraussetzt.

> *"Each timer has one thread on which tasks are executed sequentially. When this thread is busy running a task, runnable tasks may be subject to delays."* [9]

> *"The TimerTask class represents a task to run at a specified time. The task may be run once or repeatedly."* [10]

Folgende Services, Timer und TimerTask werden in der TourLive Applikation verwendet.

RecordingService

RecordingService.java ist verantwortlich für die Aufnahme von Live Informationen. Dieser wird entweder manuell über den Start-Button oder via Device Management Server über den Fernstart, strom- oder zeitbasiert gestartet. Direkt mit dem RecordingService verknüpft sind folgende Services: ValueContainerService, ImageService und der VideoService.

9. Timer, `http://developer.android.com/reference/java/util/Timer.html`„ aufgerufen am 11.04.2013
10. TimerTask, `http://developer.android.com/reference/java/util/TimerTask.html`„ aufgerufen am 11.04.2013

ValueContainerService

ValueContainerService.java ist für die Übertragung von Positionsdaten (ValueContainer) verantwortlich. Der ValueContainerService registriert einen LocationListener, der GPS-Updates meldet und diese dann mit Hilfe des DataCreators in ValueContainer-Objekt erstellt und per ValueContainerTask an den TourLive Server überträgt.

ImageService

ImageService.java ist für die Übertragung von Bilddaten verantwortlich. Der ImageService terminiert mit der Unterstützung eines Timers den TimerTask *ImageTimerTask.java*, der in regelmässigem Intervall ein Einzelbild aufnimmt und dieses per *UploadMediaTask.java* an den TourLive Server sendet.

VideoService

VideoService.java ist für die Übertragung von Videodaten verantwortlich. Analog zum ImageService nutzt der VideoService den *VideoTimerTask.java* zur Aufnahme von Videosequenzen und den gemeinsamen UploadMediaTask.java zur Übertragung der Videosequenzen an den TourLive Server.

StartUpService

StartUpService.java ist für den Auto-Start der Applikation nach dem Gerätestart verantwortlich, sofern dieser in den Einstellungen aktiviert ist. [Vog11]

StatusDataService

StatusDataService.java ist für die Übertragung des Gerätezustandes verantwortlich. Mit Hilfe eines Timers und des *StatusDataTimerTask.java* wird in regelmässigem Intervall der Gerätezustand an den Device Management Server übertragen.

4.3.5 Kommunikation mit den Servern

Für die Übertragung der Daten an die beiden Server wurden AsyncTasks verwendet. Diese ermöglichen eine einfache Anwendung von Threads und sind in der Android Developer Reference wie folgt definiert:

> *"AsyncTask enables proper and easy use of the UI thread. This class allows to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers. An asynchronous task is defined by a computation that runs on a background thread and whose result is published on the UI thread. An asynchronous task is defined by 3 generic types, called Params, Progress and Result, and 4 steps, called onPreExecute, doInBackground, onProgressUpdate and onPostExecute."* [11]

Es folgt eine kurze Auflistung der implementieren AsyncTasks.

11. AsyncTask, `http://developer.android.com/reference/android/os/AsyncTask.html`,, aufgerufen am 13.04.2013

- *DeviceManagementTask.java* überträgt die Einstellungen an den Device Management Server und registriert das Gerät beim Applikationsstart.
- *LogReaderTask.java* aktualisiert die LogView alle 5 Sekunden.
- *MessageTask.java* bezieht Nachrichten vom Device Management Server, sofern welche vorhanden sind.
- *StageInfoTask.java* bezieht Renn- und Etappeninformationen vom TourLive Server.
- *StatusDataTask.java* sendet den Gerätezustand an den Device Management Server und empfängt allfällige Konfigurationsänderungen vom Device Management Server.
- *UploadMediaTask.java* sendet Bild- und Videodaten an den TourLive Server.
- *ValueContainerTask.java* sendet Positionsdaten an den TourLive Server.

4.3.6 Listener & BroadcastReceiver

Listener und BroadcastReceiver empfangen System- und andere Events.

- *LocationChangedListener.java* meldet Positionsupdates. [Vog10]
- *PreferenceChangedListener.java* meldet Einstellungsänderungen.
- *SignalStrengthListener.java* meldet Signalstärkenänderungen.
- *BootUpReceiver.java* meldet einen Gerätestart.
- *PowerConnectionReceiver.java* meldet Änderungen in der Stromversorgung. [Vog13]

4.3.7 Aufnahme von Bildern und Videosequenzen

Die Aufnahme von Bild- und Videosequenzen kann über die Android API grundsätzlich ziemlich komfortabel realisiert werden. Vorausgesetzt wird lediglich eine SurfaceView (Bereich auf dem Bildschirm), die als Kamera Vorschau genutzt werden kann.

> *"Sets the Surface to be used for live preview. Either a surface or surface texture is necessary for preview, and preview is necessary to take pictures."* [12]

Eine View wiederum muss mit einer Activity verknüpft werden und eine Activity wiederum kennt verschiedene Status (*active*, *paused*, *stopped*, *destroyed*). Welche Activity sich in welchem Status befindet wird vom Android Betriebssystem verwaltet. Grundsätzlich kann aber davon ausgegangen werden, dass eine Activity sich nicht mehr im Status *active* befindet, wenn die Applikation sich im Hintergrund befindet oder der Bildschirm sich abdunkelt. Befindet sich eine Activity nicht mehr im Status *active* kommt es vor, dass der Garbage Collector View-Objekte aufräumt die beim Reaktivieren der Activity neu initialisiert werden.

Mit diesem Verhalten der Android API gibt es zwei Konflikte mit den Anforderungen an die TourLive Applikation.

Das erste Problem mit der geforderten View, die als Vorschau benötigt wird, kann mit einer 1x1 Pixel grossen View die "unsichtbaren" in einer Ecke angezeigt wird, gelöst werden.

12. AsyncTask, `http://developer.android.com/reference/android/hardware/Camera.html#setPreviewDisplay%28android.view.SurfaceHolder%29`,, aufgerufen am 17.04.2013

Das zweite, etwas grössere Problem besteht darin, dass Bilder und Videosequenzen auch aufgenommen werden sollen, wenn die Applikation im Hintergrund oder mit abgedunkelten Bildschirm läuft. Da in diesen beiden Fällen die Möglichkeit besteht, dass die MainActivity in einen anderen Status als *active* wechselt, kann es vorkommen, dass die View für die Kamera Vorschau vom Garbage Collector aufgeräumt wird. In diesem Fall muss die MainActivity neu gestartet und die View für die Kamera Vorschau neu initialisiert werden. Dies wird über einen Intent realisiert, der die MainActivity aufweckt, wenn diese sich nicht mehr im Status *active* befindet.

Da die Android Kamera Design Prinzipien nicht für einen solchen Verwendungszweck ausgelegt ist (Aufnahme ohne Vorschau, Aufnahme bei abgedunkeltem Bildschirm / Applikation im Hintergrund), kann es während dem Betrieb zu unvorhergesehenen Problemen (Exceptions) kommen.

4.3.8 Datenbank

Für das lokale Caching werden Daten mittels ORMLite in die Datenbank geschrieben. Eine Klasse wird mittels Annotation als Datenbankklasse definiert.

```
1  @DatabaseTable
2  public class Device {
3    @DatabaseField(generatedId = true)
4    private int id;
5    @DatabaseField
6    private String deviceId;
7    @DatabaseField
8    private String username;
9    @DatabaseField
10   private String phoneNr;
11 }
```

Quellcode 4.1: ORMLite Annotations

- ***@DatabaseTable*:** mit dieser Annotation wird angegeben, dass die annotierte Klasse eine gemappte Datenbank Klasse ist.
- ***@DatabaseField*:** das folgende Attribut ist eine Spalte in der Datenbank. Mittels generatedID = true wird für das Attribut eine eindeutige ID generiert.

Die Klasse *DatabaseHelper.java* verwaltet die Zugriffe auf die Datenbank. Beim ersten Start der App werden die Tabellen generiert. Werden später neue Attribute hinzugefügt, so muss die Datenbankversion erhöht und in der *onUpgrade* - Methode ein Update-Script zur Verfügung gestellt werden.

Der *DatenbankManager.java*, eine Singleton Klasse, ermöglicht den Zugriff über den *DatabaseHelper.java* auf die Datenbank. Im *DatenbankManager.java* sind sämtliche *CRUD*-Operationen enthalten.

4.3.9 Systemsequenzdiagramme

Dieses Kapitel beschreibt die Abläufe innerhalb der Applikation in Form von Systemsequenzdiagrammen. Sie geben eine grobe Übersicht wie die bisher erwähnten Klassen zusammenarbeiten.

Applikation im Ruhezustand

Diagramm 4.10 beschreibt die Applikation im Ruhezustand. In regelmässigem Intervall (wird in der *Config.java* definiert) wird der Gesundheitszustand des Gerätes an den Device Management Server übertragen. Der Device Management Server antwortet mit der gespeicherten Gerätekonfiguration des Gerätes sowie einem Message-Flag, das kennzeichnet, wenn eine Nachricht für das Gerät vorhanden ist.

Sind keine Renn- und Etappeninformationen vorhanden oder sind diese veraltet, werden diese nach Beendigung des ersten Requests beim TourLive Server abgerufen.

Wurde zudem das Message-Flag gesetzt, wird in einem weiteren Request die Nachricht abgerufen.

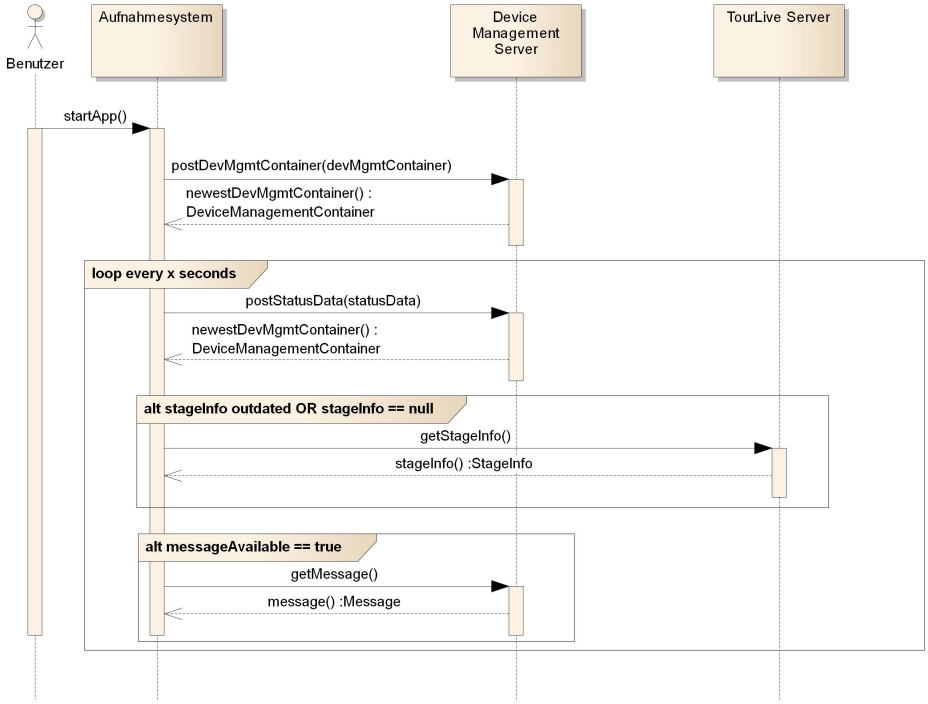


Abbildung 4.10: Aufnahmesystem permanente Tasks

Applikation im Aufnahmemodus

Diagramm 4.11 zeigt die Abläufe beim Starten des Aufnahmemodus (*RecordingService.java*). Die Operation *startRecording()* entspricht dem *RecordingService.java*. Der Start des RecordingServices beinhaltet auf jeden Fall den Start des *ValueContainerService.java*. Je nach Konfiguration wird zudem der *VideoService.java*, der *ImageService.java* oder keiner der beiden gestartet.

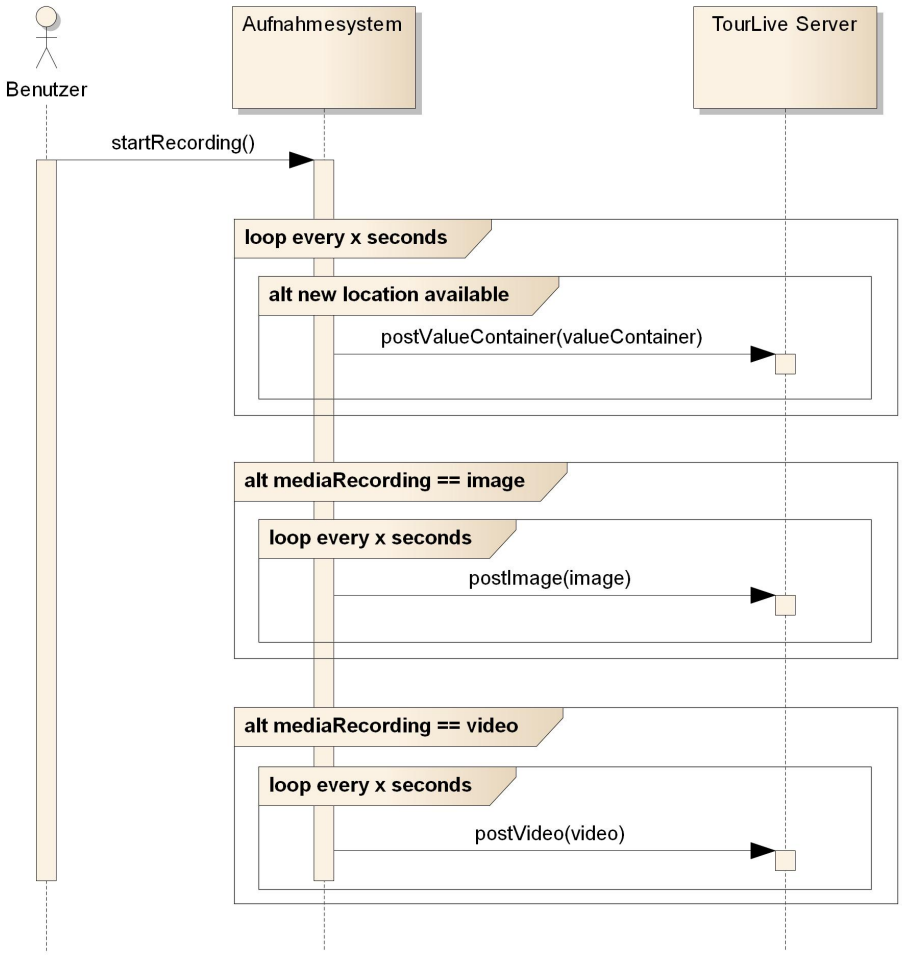


Abbildung 4.11: Aufnahmesystem Aufnahme

4.3.10 Algorithmen

Das Aufnahmesystem enthält einige Algorithmen zur Berechnung von Distanzen, Steigungen und Zeit. Diese werden im folgenden Abschnitt kurz beschrieben. Damit eine Berechnung durchgeführt werden kann, wird ein *StageInfo*-Objekt vorausgesetzt, das beschreibt, in welchem Rennen beziehungsweise in welcher Etappe sich das Aufnahmegerät aktuell befindet.

Berechnung der Renndauer

Die Renndauer ist in der TourLive Applikation im Tab Etappe (eng. Stage) zu finden und beschreibt die Renndauer seit dem offiziellen Etappenstart gemäss *StageInfo*. Die Berechnung der Renndauer erfolgt durch eine einfache Subtraktion.

Algorithmus 1 Berechnung der Renndauer

```
1: if StageInfo ≠ null then
2:     return now - StageInfo.StageStartTime;
3: else
4:     return 0;
5: end if
```

Die Renndauer wird bei der Erstellung eines *ValueContainers* im assoziierten *StageData* im Attribut *stageTime* gespeichert.

Berechnung der überwundenen Höhenmeter

Die überwundenen Höhenmeter repräsentieren das Total der positiv überwundenen Höhenmeter während einer Etappe. Der Wert wird in der TourLive Applikation im Tab Etappe (eng. Stage) angezeigt. Die Berechnung erfolgt nach folgendem Algorithmus:

Algorithmus 2 Berechnung der überwundenen Höhenmeter

```
1: Altitude ← 0;
2: CurrentAltitude ← null;
3: NextAltitude ← null;
4: for all ValueContainers since StageInfo.StageStartTime do
5:     if NextAltitude ≠ null then
6:         CurrentAltitude ← ValueContainer.Altitude;
7:         if CurrentAltitude ≤ NextAltitude then
8:             Altitude ← Altitude + (NextAltitude − CurrentAltitude)
9:         end if
10:    else
11:        NextAltitude ← ValueContainer.Altitude;
12:    end if
13: end for
```

Die überwundenen Höhenmeter werden bei der Erstellung eines *ValueContainers* im assoziierten *StageData* im Attribut *stageUpAltitude* gespeichert.

Berechnung der Steigung

Die Angabe zur Steigung beziehen sich auf die letzten 100m Horizontaldistanz. Die Steigung wird im Tab Etappe (eng. Stage) angezeigt und bei der Erstellung eines *ValueContainers* im assoziierten *StageData* als Attribut *incline* gespeichert. Die Berechnung erfolgt nach folgendem Algorithmus:

Algorithmus 3 Berechnung der Steigung

```
1: Altitude ← 0;
2: Distance ← 0;
3: LastAltitude ← null;
4: LastDistance ← null;
5: for all ValueContainers.Reverse do
6:     if valueContainer is in currentStage then
7:         CurrAltitude ← ValueContainer.Altitude;
8:         CurrDistance ← ValueContainer.Distance;
9:         if LastAltitude ≠ null and LastDistance ≠ null then
10:            Altitude ← Altitude + (LastAltitude − CurrAltitude);
11:            Distance ← Distance + (LastDistance − CurrDistance);
12:        end if
13:        LastAltitude ← CurrAltitude;
14:        LastDistance ← CurrDistance;
15:        if Distance ≥ 100 then
16:            BREAK;
17:        end if
18:    else
19:        BREAK;
20:    end if
21: end for
22: Incline ← altitude/distance;
23: return arctan(incline.toDegrees);
```

Allgemeine Algorithmen zur Vermeidung von Berechnungsfehlern

Die Erstellung eines ValueContainers erfolgt sobald das Aufnahmegerät eine neue Position meldet. Es kann vorkommen, dass die GPS-API von Android fehlerhafte Positionsdaten liefert. Aus diesem Grund werden erstellte ValueContainer auf ihre Datenkonsistenz überprüft und gegebenenfalls wieder verworfen. Der ValueContainer wird nach folgenden Kriterien überprüft:

- longitude != 0
- latitude != 0
- accuracy < 100m
- returned location from GPS_PROVIDER != null

4.3.11 TourLive Recovery Service

Die Notfallwiederherstellung, wie sie in den Anforderungen ans Aufnahmesystem formuliert ist, wurde aus Gründen der Zuverlässigkeit und Stabilität als separate Android Applikation (*TourLiveRecoveryService.apk*) realisiert. Der TourLiveRecoveryService startet beim Gerätestart automatisch und besteht aus zwei Komponenten. Der SmsReceiver scannt empfangene SMS nach dem Stichwort *RESTART* und sendet, falls dieses Stichwort gefunden wurde, eine Nachricht (Intent) an die TourLive Applikation, um diese nach einem Absturz erneut zu starten.

5 Schnittstellen

Ein wichtiger Bestandteil des TourLive Gesamtsystems ist die Kommunikation zwischen den verschiedenen Komponenten.

Die Schnittstellen werden in private und öffentliche Schnittstellen unterteilt. Über die privaten Schnittstellen kommunizieren die TourLive Geräte (Aufnahmegeräte und der RadioTourSpeaker) mit dem TourLive Server und dem Device Management Server. Die öffentlichen Schnittstellen dienen Drittentwicklern zur Abfrage von TourLive Daten.

5.1 Private Schnittstellen

Folgende Abbildung bietet eine grobe Übersicht über alle verfügbaren privaten Schnittstellen.

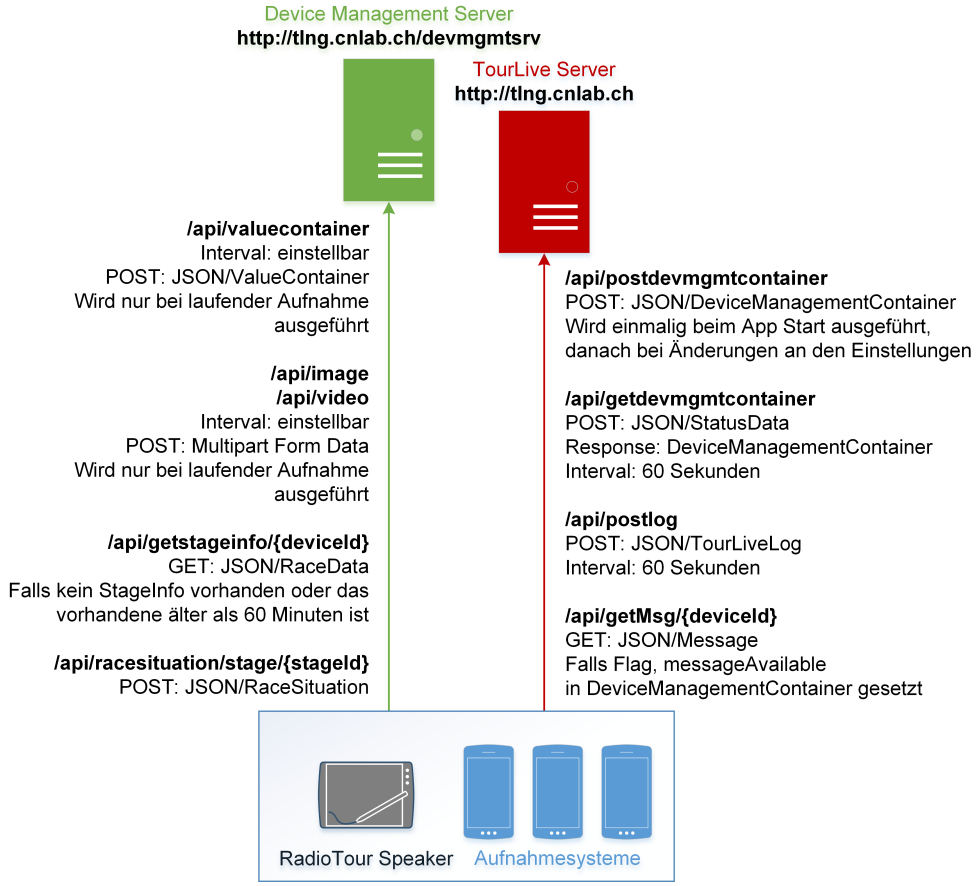


Abbildung 5.1: Übersicht private Schnittstellen

5.1.1 Schnittstellen TourLive Server

Die Aufnahmegeräte liefern über diese Schnittstellen ValueContainer, Bilder und Videos an den TourLive Server.

ValueContainer an den TourLive Server senden

URL:	/api/valuecontainer	
Request:	Method: Content-Type: Body:	POST application/json JSON-String *ValueContainer*
Response:	Header: Body:	HTTP/1.1 200 OK <empty>

Tabelle 5.1: Schnittstelle ValueContainer

JSON Beispiel - ValueContainer

Der TourLive Server erwartet die folgende JSON-Struktur für die Darstellung eines ValueContainers.

```
{
    "device": {
    ...
    },
    "stageData": {
        "stageUpAltitude": 0,
        "stageTime": 0,
        "incline": 0,
        "distance": 0,
        "averageSpeed": 0
    },
    "netData": {
        "technology": "EDGE",
        "mncmcc": "22803",
        "locationArea": 7018,
        "signalStrength": -85,
        "cellNumber": 27432,
        "upstream": 0
    },
    "positionData": {
        "satellites": "0",
        "altitude": 0,
        "direction": 0,
        "latitude": 47.223472595214844,
        "longitude": 8.817152976989746,
        "accuracy": 20,
        "speed": 0
    },
    "timestamp": 1370621236854
}
```

Abbildung 5.2: JSON-Struktur: ValueContainer

Bild an den TourLive Server senden

URL:	/api/image	
Request:	Method:	POST
	Content-Type:	multipart/form-data [3 Boundaries]
	Body-Boundary 1:	text/plain (timestamp)
	Body-Boundary 2:	text/plain (deviceId)
	Body-Boundary 3:	application/octet-stream (Bild)
Response:	Header:	HTTP/1.1 200 OK
	Body:	<empty>

Tabelle 5.2: Schnittstelle Bild senden

Video an den TourLive Server senden

URL:	/api/video	
Request:	Method:	POST
	Content-Type:	multipart/form-data [3 Boundaries]
	Body-Boundary 1:	text/plain (timestamp)
	Body-Boundary 2:	text/plain (deviceId)
	Body-Boundary 3:	application/octet-stream (Video)
Response:	Header:	HTTP/1.1 200 OK
	Body:	<empty>

Tabelle 5.3: Schnittstelle Video senden

Rennsituation an den TourLive Server senden

URL:	/api/racesituation/stage/{stageId}	
Request:	Method: Body:	POST JSON-String 'RaceSituation'
Response:	Header: Body:	HTTP/1.1 200 OK <empty>
Bemerkung:	Der RadioTourSpeaker sendet (unabhängig von den TourLive Aufnahmegeräten) periodisch die Rennsituation an den Server	

Tabelle 5.4: Schnittstelle Rennsituation senden

JSON Beispiel - Rennsituation

Der TourLive Server erwartet die folgende JSON-Struktur für die Darstellung der Rennsituation.

```
{
  "timestamp":1338284032350,
  "situation":
    [
      {
        "riderNumber":[11, 15, 3],
        "isLeader":true,
        "groupnr":0,
        "handicaptime":-1234000,
        "isField":false
      },
      {
        "isLeader":false,
        "groupnr":1,
        "handicaptime":-2334000,
        "isField":true
      },
      {
        ...
      }
    ]
}
```

Abbildung 5.3: JSON-Struktur: Rennsituation

5.1.2 Renninformation vom TourLive Server abfragen

URL:	/api/getstageinfo/{deviceId}	
Request:	Method: Body:	GET <empty>
Response:	Header: Content-Type: Body:	HTTP/1.1 200 OK application/json Custom JSON-String
Bemerkung:	Im Body der Response wird eine eigens kreierte HashMap ausgeliefert, darin sind aktuelle Etappeninformationen für das angegebene Gerät. Diese Informationen werden für den Rennbegleiter auf dem Gerät dargestellt	

Tabelle 5.5: Schnittstelle Renninformation abfragen

JSON Beispiel - Renninformation

```
{
  raceName: "Tour de Suisse",
  stageName: "Quinto - Airolo",
  currentStageStarttime: 1370693400000,
  completedStagesDistance: 0,
  currentRaceTotalDistance: 832,
  currentStageTotalDistance: 183.4
  currentStageEndtime: 1370700600000,
}
```

Abbildung 5.4: JSON-Struktur: Renninformation

5.1.3 Device Management Server Schnittstellen

Einstellungen an den Device Management Server senden

Sobald Einstellungen im TourLive Aufnahmesystem geändert werden, müssen die Einstellungen über diese Schnittstelle mit dem Device Management Server synchronisiert werden.

URL:	/devmgmtsrv/api/postdevmgmtsrv	
Request:	Method: Content-Type: Body:	POST application/json JSON-String 'DeviceManagementContainer'
Response:	Header: Body:	HTTP/1.1 200 OK <empty>
Eigenschaft:	Wird einmalig beim App Start ausgeführt sowie bei lokalen Änderungen in den Einstellungen	
Interval:	unregelmässig	

Tabelle 5.6: Schnittstelle Einstellungen

JSON - DeviceManagementContainer

```
{
    "additionalSettings": {
        "raceDistanceCorrection": 0
    },
    "recordingSettings": {
        "videoResolution": "LOW",
        "cameraType": "Back",
        "streamingMode": "video",
        "imageResolution": "LOW",
        "recordingMode": "manual",
        "automaticCameraAdjustment": true,
        "recordingEndTime": 0,
        "recordingStartTime": 0,
        "dataTransferInterval": 27,
        "imageTransferInterval": 27
    },
    "device": {
        "deviceId": "8459c58b06368bbb",
        "username": "Galaxy Nexus Orange",
        "phoneNr": "41789645164"
    },
    "deviceSettings": {
        "tourLiveServerSec": "tlng.cnlab.ch",
        "tourLiveServerPrim": "tlng.cnlab.ch",
        "operationMode": "unmanaged",
        "deviceManagementServer": "tlng.cnlab.ch/devmgmtsrv",
        "criticalBatteryValue": 15,
        "backlight": true,
        "appAutoStart": true
    },
    "configChangedTime": 1370610867542,
    "messageAvailable": false,
    "startRecording": false
}
```

Abbildung 5.5: JSON-Struktur DeviceManagementContainer

Gerätezustand dem Device Management Server mitteilen

Um den aktuellen Gesundheitszustand des Aufnahmesystems dem Device Management Server mitzuteilen, wird diese Schnittstelle benutzt. Vom Device Management Server wird mit einem DeviceSettingsContainer geantwortet.

URL:	/devmgmtsrv/api/getdevmgmtsrv	
Request:	Content-Type: Method: Body:	application/json POST JSON-String 'StatusData'
Response:	Method: Content-Type: Body:	POST application/json JSON-String 'DeviceManagementContainer'
Eigenschaft:	Wird regelmässig als Service ausgeführt.	
Interval:	regelmässig - alle 60 Sekunden	

Tabelle 5.7: Schnittstelle Status

JSON Beispiel - StatusData

Das JSON eines Status Posts sieht folgendermassen aus:

```
{
    "satellites": "0",
    "device": {
        "deviceId": "d1fac717d37751a",
        "username": "username",
        "phoneNr": ""
    },
    "powerSupplyMode": "charging",
    "operationMode": "managed",
    "lastConfigSent": 1370621254591,
    "lastMediaSent": 0,
    "lastPositionSent": 0,
    "freeSpaceMemoryCard": 78.61,
    "freeSpaceIntern": 78.61,
    "tourLiveDataSize": 2437092836,
    "batteryStatus": 86,
    "recordingRunning": false,
    "upstream": 0
}
```

Abbildung 5.6: JSON-Struktur: Status

Log an den Device Management Server übertragen

Über diese Schnittstelle werden die Log-Einträge des Aufnahmesystems an den Device Management Server übertragen.

URL:	/devmgmtsrv/api/postlog	
Request:	Method: Content-Type: Body:	POST application/json JSON-String 'TourLiveLog'
Response:	Header: Body:	HTTP/1.1 200 OK <empty>
Eigenschaft:	Wird regelmässig als Service ausgeführt.	
Interval:	regelmässig - alle 60 Sekunden	

Tabelle 5.8: Schnittstelle Log

JSON Beispiel - Log

Das JSON eines Log Posts sieht folgendermassen aus:

```
{
    "device": {
        "deviceId": "8459c58b06368bbb",
        "username": "Galaxy Nexus Orange",
        "phoneNr": "41789645164"
    },
    "entries": [
        {
            "message": "start image",
            "identifier": "ch.hsr.ba.tourlive.service.StatusDataService",
            "level": "DEBUG",
            "date": 1366893166692
        },
        {
            "message": "onReceive",
            "identifier": "ch.hsr.ba.tourlive.receiver.PowerConnectionReceiver",
            "level": "DEBUG",
            "date": 1366893166725
        },
        {
            "message": "GPS Initalized",
            "identifier": "ch.hsr.ba.tourlive.view.StartFragment",
            "level": "DEBUG",
            "date": 1366893167076
        }
    ]
}
```

Abbildung 5.7: JSON-Struktur: Log

Nachricht vom Device Management Portal abholen

Ob eine neue Nachricht für ein Gerät verfügbar ist, wird dem Gerät im DeviceManagementContainer über das Flag *messageAvailable* bekannt gegeben. Über diese Schnittstelle kann das Gerät die Nachricht dann abrufen. Ist keine Nachricht verfügbar, wird ein leerer String zurückgegeben.

URL:	/devmgmtsrv/api/getMsg/{deviceId}	
Request:	Method: Body:	GET <empty>
Response:	Method: Content-Type: Body:	POST application/json JSON-String 'Message'
Eigenschaft:	Wenn das 'messageAvailable'-Flag gesetzt ist in einem empfangenen DeviceManagementContainer.	
Interval:	unregelmässig	

Tabelle 5.9: Schnittstelle Nachricht abholen

JSON Beispiel - Message

Das JSON für die Nachrichten ist das Folgende:

```
{
    "device": {
        "deviceId": "8459c58b06368bbb",
        "username": "Galaxy Nexus Orange",
        "phoneNr": "41789645164"
    },
    "messageId": 2,
    "textMessage": "Hallo Simon",
    "readAt": 0
}
```

Abbildung 5.8: JSON-Struktur: Nachricht

5.2 Öffentliche Schnittstellen

Die Renn- und Etappeninformationen stehen auch für Drittentwickler zur Verfügung. Zu jeder Etappe können zusätzlich die ValueContainer und Bilddaten abgefragt werden. Folgende Abbildung gibt eine grobe Übersicht über die öffentlichen Schnittstellen.

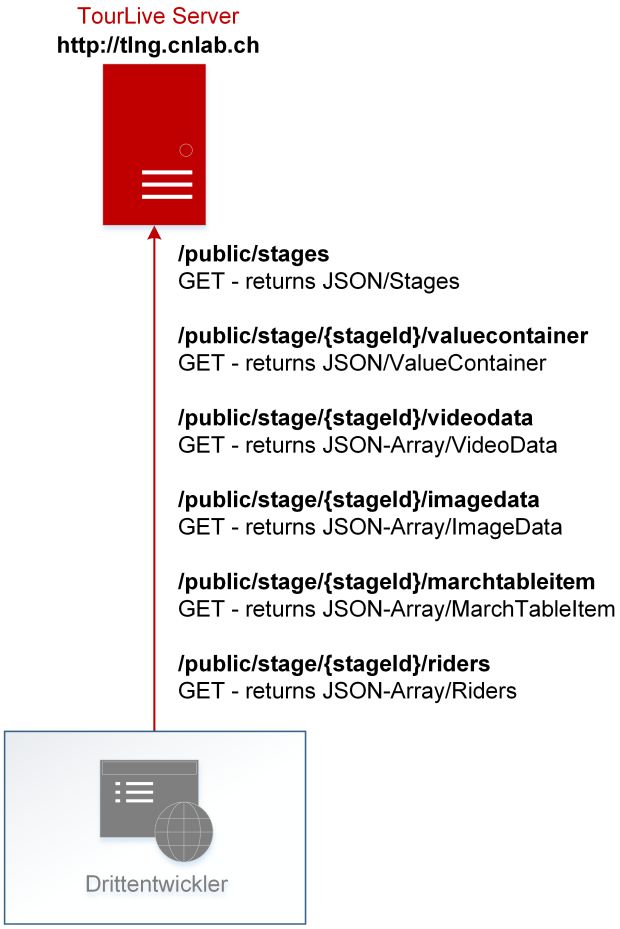


Abbildung 5.9: Übersicht öffentliche Schnittstellen

Etappen beim TourLive Server abfragen

URL:	/public/stages	
Request:	Method: Body:	GET <empty>
Response:	Header: Content-Type: Body:	HTTP/1.1 200 OK application/json JSON-Array 'Stages'
Bemerkung:	Alle sichtbaren Etappen	

Tabelle 5.10: Schnittstelle Etappen abfragen

Beispiel JSON - Array Stages

```
[
  {
    stageId: 1,
    stageName: "Tour de Rappi",
    stageSlug: "tour-de-rappi",
    stageDescription: "Tour um Rapperswil",
    starttime: "25.05.2013 - 17:00",
    endtime: "25.05.2013 - 19:30",
    offsettime: "00:00:00",
    distance: 22,
    bannerImage: "stage1/stageBanner.png",
    stageProfileImage: "stage1/stageProfile.png",
  ...
]
```

Abbildung 5.10: JSON-Struktur: Etappen

ValueContainer beim TourLive Server abfragen

URL:	/public/stage/{stageId}/valuecontainer	
Request:	Method: Body:	GET <empty>
Response:	Header: Content-Type: Body:	HTTP/1.1 200 OK application/json JSON-Array 'ValueContainers'
Bemerkung:	Alle ValueContainers einer Etappe	

Tabelle 5.11: Schnittstelle ValueContainer abfragen

Bilddaten beim TourLive Server abfragen

URL:	/public/stage/{stageId}/imagedata	
Request:	Method: Body:	GET <empty>
Response:	Header: Content-Type: Body:	HTTP/1.1 200 OK application/json JSON-Array 'ImageData'
Bemerkung:	Alle ImageData Objekte zu dieser Etappe, nicht aber die eigentlichen Bilder. Die Bilder können aber mit dem Feld *imageLocation* entweder direkt verlinkt oder heruntergeladen werden	

Tabelle 5.12: Schnittstelle Bilddaten abfragen

Beispiel JSON - Bilddaten

```
[
  {
    imageDataId: 136,
    device: {
      ...
    },
    imageLocation: "images/awesome_image.png",
    realTimestamp: 1369494910108
  },
  {
    imageDataId: 137,
...
]
```

Abbildung 5.11: JSON-Struktur: Bilddaten

Videodaten beim TourLive Server abfragen

URL:	/public/stage/{stageId}/videodata	
Request:	Method: Body:	GET <empty>
Response:	Header: Content-Type: Body:	HTTP/1.1 200 OK application/json JSON-Array 'VideoData'
Bemerkung:	Alle VideoData Objekte zu dieser Etappe, nicht aber die eigentlichen Videosequenzen. Die Videos können aber mit dem Feld *videoLocation* entweder direkt verlinkt oder heruntergeladen werden	

Tabelle 5.13: Schnittstelle Videodaten abfragen

Beispiel JSON - Videodaten

```
[
  {
    videoDataId: 136,
    device: {
      ...
    },
    videoLocation: "videos/awesome_video.mp4",
    realTimestamp: 1369494910108
  },
  {
    videoDataId: 137,
...
]
```

Abbildung 5.12: JSON-Struktur: Videodaten

Marschtabelle beim TourLive Server abfragen

URL:	/public/stage/{stageId}/marchtableitem	
Request:	Method: Body:	GET <empty>
Response:	Header: Content-Type: Body:	HTTP/1.1 200 OK application/json JSON-Array 'MarchTableItem'
Bemerkung:	Die Marschtabelle ist aufgeteilt in Einheiten. Jede Reihe bedeutet eine Einheit. Pro Etappe können alle Marschtabelleneinheiten abgerufen werden.	

Tabelle 5.14: Schnittstelle Marschtabelle abfragen

JSON Beispiel - Array Marschtabelle

```
[
  {
    marchTableItemId: 25,
    icon: "verpflegung",
    altitude: 319,
    distance: 0,
    distanceToGo: 181.5,
    settlement: "Gippingen, Start",
    advertisingColumn: "12:30:00",
    raceSlow: "13:30:00",
    raceMedium: "13:30:00",
    raceFast: "13:30:00"
  },
  {
    marchTableItemId: 26,
    icon: "tunnel",
...
]
```

Abbildung 5.13: JSON-Struktur: Marschtabelle

5.2.1 Rennfahrer beim TourLive Server abfragen

URL:	/public/stage/{stageId}/riders	
Request:	Method: Body:	GET <empty>
Response:	Header: Content-Type: Body:	HTTP/1.1 200 OK application/json JSON-Array 'Riders'
Bemerkung:	Sämtliche Fahrer, welche dieser Etappe zugeordnet sind	

Tabelle 5.15: Schnittstelle Rennfahrer abfragen

JSON Beispiel - Array Rennfahrer

```
[
  {
    riderId: 321,
    startNr: 1,
    name: "Keizer Martijn",
    team: "Vacansoleil - DCM",
    teamShort: "VCD",
    country: "NED",
    note: null,
    neo: false,
    timeOff: "00:00",
    timeRueck: "00:00",
    timeVirt: "00:00"
  },
  {
    riderId: 322,
    startNr: 2,
    name: "Kreder Wesley",
...
]
```

Abbildung 5.14: JSON-Struktur: Rennfahrer

6 Ergebnisse und Schlussfolgerungen

Alle Komponenten dieser Arbeit wurden gemäss den Anforderungen erarbeitet und getestet. In den folgenden Abschnitten werden die Schlussfolgerungen erläutert und einen Ausblick über die weitere Entwicklung und den möglichen Einsatz an Radrennen gegeben.

6.1 Endprodukt

Die zu Beginn der Arbeit definierte Spezifikation sämtlicher Systeme konnten umgesetzt und an den Radsporttagen in Gippingen einem Test unterzogen werden. Ein ausführlicher Testbericht befindet sich im Anhang 7.2.1. Die drei Komponenten, TourLive Server, die Aufnahmegeräte und der Device Management Server arbeiten wie erwartet zusammen.

6.2 Ausblick

Nach den positiven Erfahrungen an den Radsporttagen in Gippingen konnte das System auch an der Tour de Suisse getestet werden. In einem zweiten Live Test wurde der Einsatz parallel zum bestehenden System von der cnlab Software AG durchgeführt. Allfällige Verbesserungsmöglichkeiten und Ideen zur Weiterentwicklung von TourLive wurden im Kapitel 7.2.2 im Anhang zusammengefasst. Allfällige Anpassungen am System werden durch die cnlab Software AG vorgenommen.

Für die Dauer der Entwicklung wurde der TourLive Server auf einem virtuellen Server betrieben. Beim Einsatz an einem Radrennen wie der Tour de Suisse wird empfohlen, die Dienste auf mehrere Server aufzuteilen. Weiter hat sich herausgestellt, dass sich nicht alle Androidgeräte gleich gut als Aufnahmegeräte eignen. Während der Entwicklung wurde primär mit dem Gerät Samsung Galaxy Nexus[1] gearbeitet, welches auch die besten Testresultate erzielte. Ebenfalls zu empfehlen, und auch etwas aktuellere bezüglich Hardware, ist das Samsung Galaxy S3[2].

1. Samsung Galaxy Nexus, `http://www.gsmarena.com/samsung_galaxy_nexus_i9250-4219.php` aufgerufen am 12.06.2013
2. Samsung Galaxy S3, `http://www.samsung.com/global/galaxys3/`, aufgerufen am 02.06.2013

Abbildungsverzeichnis

Glossar

API
Die API (Application Programming Interface), beschreibt die Schnittstelle zu einem System oder einer Technologie. 21, 26, 39, 102

CRUD
Das Akronym CRUD umfasst die grundlegenden Datenbankoperationen Create (Datensatz anlegen), Read (Datensatz lesen), Update (Datensatz aktualisieren) und Delete (Datensatz löschen). . 52

CSV
Comma Separated Values, eine Tabellendarstellungsform und maschinenlesbares Format zur Darstellung von Daten. 28, 29

git
Das Versionscontrollsystem git ist ein dezentrales OpenSource Sourcecodeversionierungssystem, welches sich für die Entwicklung von Software in Teams sehr gut eignet. 79

HTTP
Hypertext Transfer Protocol. 10, 100

JSON
JSON (JavaScript Opject Notation) ist eine Darstellungsform eines Objektes welches menschliche Lesbarkeit und maschinelle Verarbeitung zulässt. 100

JSP
Die JavaServer Pages stellen dynamische Daten eines Servers in Form einer HTML Seite dar.. 26, 27, 33, 35

MariaDB
MariaDB ist eine Weiterentwicklung der berühmten MySQL Datenbank. Sie ist Quelloffen und verfügt über hohe Kompatibilität zu MySQL. Für weitere Informationen sei auf den ausführlichen Artikel auf Wikipedia verwiesen: `http://en.wikipedia.org/wiki/Mariadb`, aufgerufen am 07.05.2013. 19, 102

PHP
PHP (Hypertext Preprocessor) ist eine Webprogrammiersprache, welche auf dem Server ausgeführt wird und in der Regel eine dynamische HTML Webseite generiert. 17

Responsive Web Design
Responsive Web Design beschreibt den Ansatz an das Endgerät anpassungsfähiger Webseiten. Eine Webseite, welche die Regeln des Responsive Web

Designs einhält, wird auf allen möglichen Endgeräten, egal ob Smartphone, Tablet oder Notebook korrekt dargestellt. 10, 19, 89

RUP

Der Rational Unified Process (RUP) ist ein Vorgehensmodell zur Softwareentwicklung. RUP benutzt die Unified Modeling Language (UML) als Notationssprache.. 78

Webframework

Ein Webframework ist eine Entwicklungsbasis für die Softwareentwicklung von Webapplikationen, also serverbasierten Applikationen, welche über Webbbrowser aufgerufen werden. Das Framework übernimmt dabei grundlegende Funktionen und beschleunigt die Entwicklung.. 18

Literaturverzeichnis

[DPPH13] cnlab Software AG Dr. Prof. Peter Heinzmann. Administratives zu studien- und bachelorarbeiten. http://www.cnlab.ch/kurse/SABA/, 2013. letzter Zugriff am 16.05.2013.

[Fow04] Martin Fowler. Inversion of control containers and the dependency injection pattern. 2004. letzter Zugriff am 16.05.2013.

[HSR12] Rapperswil HSR, Hochschule für Technik. Muster einer erklärung. https://www.hsr.ch/Allgemeine-Infos-Diplom-Bach.4418.0.html?&no_cache=1&cid=15019&did=8096&sechash=0423f652, 2012. geschützte Ressource, letzter Zugriff am 09.06.2013.

[Mur09] Mark L. Murphy. Beginning android, 2009.

[Sam11] Colin Sampaleanu. Green beans: Getting started with spring mvc. http://blog.springsource.com/2011/01/04/green-beans-getting-started-with-spring-mvc/, 2011. letzter Zugriff am 16.05.2013.

[SB12] D. Stucki and F. Bentele. Android applikation radiotour. http://eprints.hsr.ch/id/eprint/204, 2012. letzter Zugriff am 01.06.2013.

[Tho10] Mark Thomas. One should always use apache httpd in front of apache tomcat to improve performance? 2010. letzter Zugriff am 05.06.2013.

[Vog10] Lars Vogel. Android location api tutorial. 2010. letzter Zugriff am 08.03.2013.

[Vog11] Lars Vogel. Android service tutorial. 2011. letzter Zugriff am 02.03.2013.

[Vog13] Lars Vogel. Android broadcastreceiver tutorial. 2013. letzter Zugriff am 12.05.2013.

[Yon11] Mook Kim Yong. Spring 3 mvc hello world example. http://www.mkyong.com/spring3/spring-3-mvc-hello-world-example/, 2011. letzter Zugriff am 02.06.2013.

7 Anhang

7.1 Projektmanagement

Für die Projektplanung und -verwaltung wurde auf verschiedene Hilfsmittel zurückgegriffen. Als Projektvorgehensmodell wurde in groben Zügen RUP (Rational Unified Process) angewendet. Zum Projektbeginn wurde ein Grobzeitplan erstellt und folgende Meilensteine definiert:

#	Beschreibung	Datum
0	Kickoff Meeting	31.01.2013
1	Requirements definiert	15.03.2013
2	Schnittstellen definiert	22.03.2013
3	Erster Prototyp bei der cnlab Software AG präsentiert	28.03.2013
4	Erster Feldtest Stäheli	31.03.2013
5	Zwischenpräsentation mit Gegenleser und zweiter Prototyp vorgestellt	26.04.2013
6	Komplettsystem Test	02.05.2013
7	Feature Freeze	24.05.2013
8	Code Freeze	31.05.2013
9	Abstract / Broschürentext einreichen	07.06.2013
10	Abgabe und Präsentation Poster	14.06.2013

Tabelle 7.1: Übersicht aller Meilensteine

Die Gesamtplanung mit den An- und Abwesenheiten der Projektmitgliedern und den Meilensteinen wurde in einer Excel Datei erfasst. Diese Datei befindet sich auf der CD. Die aufgewendete Arbeitszeit zu den jeweiligen Arbeitspaketen wurde im Projektverwaltungstool Redmine aufgezeichnet und kann unter `http://ita.cnlab.ch/redmine/projects/ba-tourlive` eingesehen werden.

7.1.1 Zeitauswertung

Abbildung 7.1 gibt einen Überblick in die Anzahl aufgewendeter Arbeitsstunden pro Person und Semesterwoche. Geplant wurde mit einem Projektaufwand von 1080 Stunden. Die Ist-Zeit beträgt 1326 Stunden und übersteigt die Soll-Zeit um 246 Stunden. Eine personenbezogene Zeitauswertung befindet sich auf der beiliegenden CD.

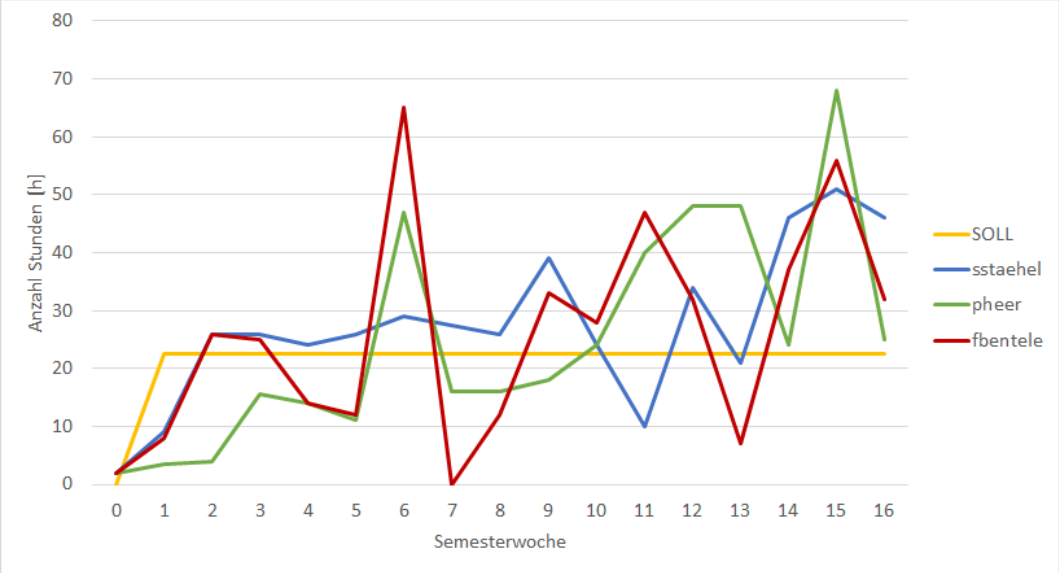


Abbildung 7.1: Zeitauswertung über alle Projektmitarbeiter

7.1.2 Dokumentation

Für diese Dokumentation wurde das Textsatzprogramm LaTeX verwendet. Es lässt sich optimal mit dem Versionierungssystem *git* kombinieren, was die Zusammenarbeit an der Dokumentation vereinfachte.

Zu jedem Meeting mit dem Industriepartner wurde ein Protokoll geschrieben. Sämtliche Protokolle und zusätzliche Dokumente befinden sich im Word Format auf der CD.

7.2 Testing

Das Testing des Gesamtsystems und der einzelnen Komponenten wurde mit verschiedenen Testverfahren abgedeckt. Funktionen und Algorithmen innerhalb der einzelnen Komponenten wurden mit Unit Tests (Kapitel 7.2.3) getestet. Systemtests (Kapitel 7.2.1) wurden durchgeführt um das Zusammenspiel der verschiedenen Komponenten und ihre Funktionen als Ganzes zu testen. Auf Usability Tests wurde absichtlich verzichtet, da sich die Komplexität der grafischen Oberflächen in Grenzen hält.

7.2.1 Systemtests

Übersicht Systemtests

#	Datum	Systemtest
1	25.05.2013	Rapperswiler Rundfahrt
2	01.06.2013	Langzeittest über Nacht
3	02.06.2013	Basel-Zürich
4	06.06.2013	Gippingen
5	09.06.2013	Tour de Suisse

Tabelle 7.2: Übersicht aller durchgeführten Systemtests

Nachfolgend werden sämtlich durchgeführten Systemtests beschrieben und ausgewertet. Im Kapitel Resultate werden die Erkenntnisse aus den Systemtests zusammengefasst und wie folgt klassifiziert:

+ *positive Erkenntnis*
- *negative Erkenntnis*

Zu jeder negativen Erkenntnis wird eine Massnahme vorgeschlagen. Massnahmen, die nicht mehr im Rahmen dieser Arbeit umgesetzt werden konnten, werden mit einem * markiert.

#1 - Systemtest Rapperswiler Rundfahrt

Ein erster Praxistest wurde durchgeführt, um das Zusammenspiel aller drei Komponenten zu testen. Auf einer Rundfahrt durch Rapperswil mit einem einzelnen Auto konnte der eine oder andere Fehler eruiert werden. Folgende Geräte wurden zu Testzwecken eingesetzt:

- Google Galaxy Nexus
- Samsung Galaxy S3

Resultate

Folgende Erkenntnisse wurden aus dem Systemtest gewonnen:

- App: Fehlberechnung bei der Distanz
- App: Fehlberechnung bei der Renndauer
- App: Reproduzierbare Abstürze (NullPointerExceptions,...)
- App: NullPointerException wenn Gerät im Landscape-Modus
- TourLive Server: DataIntegrityDataViolation Exception
- TourLive Server: Inkonsistenzen I18N
- TourLive Server: durchschnittliche Geschwindigkeit fehlt
- Fehlerhafte Werte im übertragenen ValueContainer
+ Funktionierendes Zusammenspiel der Komponenten

Sämtliche negativen Erkenntnisse wurden behoben und werden an dieser Stelle daher nicht weiter erläutert.

#2 - Systemtest Langzeittest über Nacht

Aufgrund wiederkehrender Abstürze der TourLive App bei laufender Aufnahme wurde über Nacht ein Langzeittest durchgeführt. Beide Geräte wurden einer 8 stündigen Etappe zugewiesen und sollten über Nacht Positionsdaten und Bilder übertragen.

- Google Galaxy Nexus
- Samsung Galaxy S3

Resultate

Folgende Erkenntnisse wurden aus dem Systemtest gewonnen:

- \- Samsung Galaxy S3: App ist nach 2 Stunden aufgrund eines SIG-9 FATAL ERROR's abgestürzt. Grund für solche Abstürze sind Schreib-/Lesekonflikte im Speicherbereich und treten auf, wenn zwei Threads gleichzeitig eine gemeinsame Variabel lesen / schreiben. Aufgrund dieser Erkenntnis wurden die diversen AsyncTasks, die parallel ausgeführt werden, optimiert.
- \+ Google Galaxy Nexus: Hat den Test erfolgreich bestanden. Im Log konnten keine verdächtigen Einträge festgestellt werden.

Sämtliche negativen Erkenntnisse wurden behoben und werden daher an dieser Stelle nicht weiter erläutert.

#3 - Systemtest Basel - Zürich

Während einer längeren Autofahrt wurde das System als Ganzes einem grundlegenden Test unterzogen. Bei beiden Geräten wurde als Streamingmodus die Aufnahme von Bildern konfiguriert.

- Google Galaxy Nexus [Orange SIM-Karte]
- Samsung Galaxy S3 [Swisscom SIM-Karte]

Resultate

Beide Geräte sendeten Bilder und Positionsdaten an den TourLive Server. Beim Google Galaxy Nexus mit der Orange SIM-Karte wurden aufgrund der schlechteren Netzabdeckung weniger Übertragungen registriert. Die TourLive Applikation auf dem Samsung Galaxy S3 musste 2x aufgrund eines Crashes (NullPointerException) neugestartet werden.

Sämtliche negativen Erkenntnisse wurden behoben und werden daher an dieser Stelle nicht weiter erläutert.

#4 - Systemtest Radsporttage Gippingen

Im Rahmen der Radsporttage Gippingen konnte das Gesamtsystem ein erstes Mal praxisnah an einem Radrennen getestet werden. Für den Testlauf an den Radsporttagen in Gippingen wurden drei verschiedene Androidgeräte mit der aktuellsten Version der TourLive App ausgerüstet. Auf dem TourLive Server wurde ein Rennen

und eine Etappe erstellt und die Fahrerliste sowie die Marschtabelle importiert. Im Einsatz waren folgende Gerätetypen:

- Google Galaxy Nexus [Orange SIM-Karte] im VIP-Auto
- Samsung Galaxy S3 [Swisscom SIM-Karte] im VIP-Auto
- HTC One [Sunrise SIM-Karte] im Feld
- Samsung Galaxy S3 mini [Swisscom SIM-Karte] im TourSpeaker Auto
- Samsung Galaxy S3 [Orange SIM-Karte] an der Spitze

Resultate

Nach wenigen Minuten konnte kein Kontakt mehr zum HTC One Aufnahmegerät hergestellt werden. Das Gerät konnte durch die Notfallwiederherstellung per SMS neu gestartet werden, jedoch war nach kurzem Kontakt die Verbindung zum Gerät wieder abgebrochen. Die erste Analyse ergab, dass das Gerät stark überhitzt war und sich deshalb selbst ausschaltete. Ein weiteres Gerät versuchte sich mit dem naheliegenden Deutschen Mobilfunknetz zu verbinden, was aber daran scheiterte, dass das Datenroaming deaktiviert war. Das dritte Gerät sowie die zusätzlichen Testgeräten hielten den Test durch, obwohl alle Geräte extrem warm geworden sind. Die eingelieferten Daten wurden vom TourLive Server empfangen und verarbeitet. Die Kommunikation zwischen Aufnahmegeräten und Server verlief problemlos. Zu Bemerken ist, dass zwei der eingesetzten Geräte, mit denen es auch am meisten Probleme gab, zuvor nicht getestet werden konnten. Zusammenfassend können folgende Erkenntnisse aus diesem Test mitgenommen werden:

- Grundsätzliche Probleme (Exceptions) aufgrund Programmierfehler mit den beiden Gerätetypen HTC One und Samsung Galaxy S3 mini. Exceptions, die nachvollzogen werden konnten, wurden behoben.
- Starke Erhitzung der Geräte aufgrund direkter Sonneneinstrahlung, was zu starken Performanceeinbussen, beispielsweise bei der Bedienung des GUI's, führt. Massnahme:* Direkte Sonneneinstrahlung wenn möglich vermeiden. Die Intervalle zur Übertragung von Daten heruntersetzen um die Auslastung der Geräte zu vermindern.
- Performanceeinbussen bei wachsender Grösse des lokalen Caches (ORMlite Datenbank). Massnahme:* sämtliche Daten, die bereits übertragen wurden, müssen nicht mehr zwingend lokal gespeichert werden. Dies würde die Grösse der lokalen Datenbank vermindern. Eine andere Möglichkeit wäre der Einsatz eines Alternativproduktes.
- Optimierungsmöglichkeiten bei der Darstellung der Geräte im Device Management Portal aufgrund der praktischen Anwendung. Hierbei handelt es sich primär um die Sortierung entsprechender Listen, was bereits umgesetzt wurde.
+ Wichtigkeit der Fernwartungsmöglichkeiten erkannt und mit Erfolg eingesetzt.
+ Zuverlässig funktionierende Aufnahme mit dem Gerätetyp Samsung Galaxy S3
+ Weniger als 1 km Abweichung am Etappenende bei der Berechnung der Tour-Distanz im Vergleich zum bestehenden TourLive System

#5 - Systemtest Tour de Suisse - 2. Etappe

Ein letzter Systemtest konnte an der 2. Etappe der Tour de Suisse durchgeführt werden. Im Einsatz waren folgende Gerätetypen:

- HTC One [Sunrise SIM-Karte]
- Samsung Galaxy S3 mini [Swisscom SIM-Karte]
- Samsung Galaxy S3 [Orange SIM-Karte]

Resultate

- Absturz der TourLive Android Applikation auf dem Samsung Galaxy S3 während laufender Aufnahme von Positionsdaten beim Ändern des Streamingmodus per Device Management Server. Wahrscheinlich gab es Probleme beim Neustart des RecordingService. Als Massnahme* wird empfohlen, das Ändern von Einstellungen während laufender Aufnahme experimentell zu testen um das genaue Problem zu eruieren.
- Bei schlechten Lichtverhältnissen wird nicht erkannt, wenn der Aufnahme-Button aufgrund eines falschen Aufnahmemodus deaktiviert ist. Als Massnahme wurde die Farbe des deaktivierten Start-Button in rot geändert.
- Probleme gab es der Berechnung der Abstandsentwicklung auf dem TourLive Server. Grund waren falsche Rennkilometerkorrekturen auf den Aufnahmegeräten. Massnahme:* Der Algorithmus zur Berechnung der Abstandsentwicklung auf dem TourLive Server sollte zur Berechnung der Abstandsentwicklung nicht die berechneten Distanzwerte der Aufnahmegeräte verwenden, sondern die Koordinaten im PositionDate Container.
+ Das Aufnahmegerät Samsung Galaxy S3 lief mit Ausnahme des Absturzes aufgrund Konfigurationsänderungen während der gesamten Etappe stabil.
+ Das Senden von Nachrichten an die Aufnahmegeräte funktionierte
+ Die Rennkilometerkorrektur via Device Management Server funktionierte.

7.2.2 Fazit der Systemtests

Aufgrund der Systemtests und weiterführenden Ideen während der Entwicklung und dem Testing wurden folgende Verbesserungsmöglichkeiten aller drei Komponenten zusammengestellt.

TourLive Server

- Der Algorithmus für die Abstandsentwicklung basiert aktuell auf Basis der berechneten Etappendistanzen der Aufnahmegeräte (siehe Kap 2.2.5). Anstelle dieser berechneten Distanzen könnte die Abstandsentwicklung auch auf Basis der Koordinaten im ValueContainer berechnet werden. Dies würde verhindern, dass allfällige Rechnungsfehler der Aufnahmegeräte in die Abstandsentwicklung einflössen.
- Nach dem Erstellen einer Etappe sind keine Informationen der Geräte vorhanden. Dies führt dazu, dass die Darstellung der Etappe aus Sicht der Öffentlichkeit (Webseite) unfertig wirkt.
- Für den produktiven Betrieb sollte die Webapplikation auf verschiedenen Servern verteilt werden. So kann ein System die API Aufgaben übernehmen und ein weiteres System die Webseite den Besuchern ausliefern.

- In der Etappenansicht könnte die Navigation auch via Rennkilometer im Streckenprofil ermöglicht werden.
- Aus der Administrationsansicht sollte es einen Link direkt zu der jeweiligen Etappe geben.
- Bei der automatische Aktualisierung der Seite sollte man den Zeitintervall in Sekunden angeben können.

Android Aufnahmesystem

- Der Start / Stop des Aufnahmemodus (RecordingService) scheint, sofern Bilder- oder Videostreaming aktiviert ist, sporadisch Probleme zu bereiten. Dies zeigt sich vor allem wenn der Aufnahmemodus per Fernsteuerung (Device Management Server) gestartet / gestoppt wird. Die genaue Fehlerquelle konnte bisher nicht eruiert werden.
- Entwicklung eines eigene Thread-Pools zur Verwaltung und Abarbeitung der Threads (inbesondere der AsyncTasks). Tests haben ergeben, dass die Anzahl möglicher paralleler Threads von Gerät zu Gerät variiert und auch der Android interne Thread-Pool teilweise unterschiedlich arbeitet.
- Die Verwendung von SQLite in Verbindung mit ORMlite führt bei grösseren Datenmengen zu erheblichen Performanceeinbussen. Die Datenmenge könnte reduziert werden, indem bereits übertragene Daten im lokalen Cache gelöscht werden oder indem ein alternatives, performanteres Produkt eingesetzt würde.
- Das Ausschalten des Bildschirms / "Schliessen" der Applikation führt beim Einsatz von Bilder- / Videostreaming sporadisch dazu, dass die Fragments innerhalb der MainActivity nicht mehr korrekt angezeigt werden. Eine exakte Fehlerquelle konnte bisher nicht eruiert werden.
- Befindet sich die TourLive Applikation im Aufnahmemodus "ferngesteuert" und in einer laufenden Aufnahme, kann das Aufnahmegerät ohne Löschung der gesamten Einstellungen nicht mehr verwaltet werden. Dies liegt daran, dass die Einstellungen während einer laufenden Aufnahme gesperrt sind und die Aufnahme nicht gestoppt werden kann, da der Aufnahmemodus auf "ferngesteuert" gesetzt ist.
- Deaktivierung der Audioaufnahme bei Videoaufzeichnungen zur Einsparung von Bandbreite.
- Die Telefonnummer wird nicht an den Device Management Server übermittelt, nachdem die Telefonnummer eingegeben wurde. Dies liesse sich mit dem sofortigen Ausführen des StatusDataTasks nach dem Ändern der Telefonnummer beheben.
- Automatische Ausrichtung der App funktioniert auf dem Samsung S2 nicht erwartungsgemäss.
- Ist ein Gerät im Standby Modus, kann die Applikation mit dem TourLiveRecoveryService nicht neu gestartet werden.

Device Management Server

- Implementation von I18N.
- Diverse Enums auf Deutsch übersetzen.

- Die Bild- und Videoauflösungen sind bei jedem Gerät unterschiedlich. Eine Synchronisierung der Auflösungen beim Gerätestart würde ermöglichen, dass eine gerätespezifische Auflösung ausgewählt werden könnte. Aktuell können nur drei vordefinierte Auflösungen ausgewählt werden.
- Möglichkeit bieten, die Anzahl der anzuzeigenden Logeinträge auszuwählen.
- Standardeinstellungen definieren und auf ein Gerät übertragen.
- Geräteeinstellungen auf ein anderes Gerät kopieren.
- Geräteeinstellungen gleichzeitig für mehrere Geräte vornehmen.
- Positions- und Etappendaten per Device Management Server löschen.
- Bild- und Videodateien per Device Management Server löschen.
- Spring Security implementieren. Momentan ist die Seite öffentlich zugänglich.

7.2.3 Unit Tests

Für den TourLive Server existieren grundlegende JUnit Tests. Diese dienen in erster Linie dazu, die Serverkomponente zu testen und die Entwicklung von weiteren Tests beispielhaft zu dokumentieren.

7.3 Persönliche Berichte

7.3.1 Simon Stäheli

Als besonders interessant an dieser Bachelorarbeit empfand ich den engen Praxisbezug. Zu Radrennen hatte ich zuvor keinen grossen Bezug. So musste erst einmal ein gewisses Grundverständnis für die Abläufe während eines solchen Rennens erarbeiten werden.

Die Aufgabenteilung dieses doch ziemlich umfangreichen Projektes war von Anfang an gegeben. Zusammen mit Patrizia Heer war ich für das Android Aufnahmesystem (Android Applikation) und denn Device Management Server (Java Spring Framework Werbservice) verantwortlich. Da ich mit keiner der beiden verwendeten Plattformen Praxiserfahrung vorzuweisen habe, bestanden die ersten Wochen primär aus dem Erarbeiten der Grundkenntnisse dieser beiden Technologien [Mur09]. Interessant wurde es, als die ersten Prototypen getestet werden konnten und die ersten Bilder und Positionsdaten vom Aufnahmegerät an den TourLive Server übertragen wurden.

Das Highlight dieses Projektes war sicherlich der Feldtest an den Radsporttagen in Gippingen und dass wir die Aufnahmegeräte an die zweite Etappe der Tour de Suisse mitgeben durften.

7.3.2 Patrizia Heer

Als wir mit der Bachelorarbeit begannen, wusste ich nocht nicht viel über Radrennen. So führte uns Herr Heinzmann zuerst mal in die Problemstellung ein. Nach dem Abschluss der Spezifikation war das Wissen über Radrennen komplett und wir konnten ohne Mühe mit dem Prototyp beginnen.

Mit Simon Stäheli zusammen war ich für das Aufnahmesystem und den Device Management Server zuständig. Da ich bereits Android Applikationen erstellt habe, musste ich mich nur in das Spring Framework einlesen. Als der erste Prototyp mit dem TourLive Server und dem Device Management Server kommunizierte, war die Freude gross. Doch es wurde schnell klar, dass der Funktionsumfang des Aufnahmesystems gross ist und uns viel Arbeit bevorstand.

Es war sehr spannend, die Bachelorarbeit in einem mir unbekannten Themengebiet zu absolvieren. Mein Interesse an Radrennen ist gestiegen; dies hat auch mit dem Test am Radrennen in Gippingen zu tun. Dieser Tag wird mir bestimmt noch lange in Erinnerung bleiben!

7.3.3 Florian Bentele

Da es sich für mich bei dieser Bachelorarbeit um eine Art Fortsetzungsarbeit handelt, hatte ich bereits ein gutes Verständnis für das Aufgabenumfeld. Dennoch war die Aufgabenstellung, eine Webapplikation zu entwickeln, eine neue Herausforderung.

In den ersten Wochen musste ich sehr viel Zeit in das Erlernen des mir unbekannten Spring Framework investieren. Bei Fragen konnte ich wohl auf ein fundiertes Wissen bei Mitstudenten zurückgreifen, dennoch blieb der Einstieg eher schwierig, da mir die verwendeten Konzepte nicht geläufig waren. Erst nachdem der erste Prototyp erfolgreich getestet wurde, bekam ich das Gefühl, auf das richtige Framework gesetzt zu haben.

Die Aufgabenteilung im Team war im Ansatz gegeben. Unsere Aufgabe bestand darin, die Schnittstellen zu den Systemen zu definieren und dann die Systemkomponenten umzusetzen. Dies fordert eine gewisse Flexibilität, da manche Probleme erst während der Entwicklung auftauchen und die Schnittstellen danach angepasst werden müssen. Weiter war auch nicht immer klar, auf welcher Komponente ein Task ausgeführt werden soll. Die Drehung von aufgenommen Bildern sei an dieser Stelle als Beispiel erwähnt. Gemeinsam haben wir jedoch die Probleme besprochen und immer eine geeignete Lösung gefunden.

Ich habe in diesem Projekt sehr viel über die Entwicklung von Webapplikationen gelernt. Mir ist im besonderen die Wichtigkeit für den Einsatz eines Frameworks klar geworden. Für die professionelle Entwicklung ist die Verwendung eines Frameworks unabdingbar. Persönlich bleiben mir besonders die Erlebnisse am Testlauf an den Radsporttagen in Gippingen in guter Erinnerung.

7.4 Anforderungen TourLive Server

7.4.1 Live Stream

Auf der Webseite werden Bilder von den Aufnahmegeräten zum aktuellen Zeitpunkt angezeigt. Auf den Bildern ist zu erkennen, wann dieses erstellt wurde. Das Bild kann zusätzlich vergrössert werden. Je nach Einstellung werden stehende Bilder (wie in Abbildung 1) oder fliessende Bilder (Videos) übertragen.

Abbildung 7.2: Aufnahme aus dem Fahrzeug

7.4.2 Streckenprofil

Der Verlauf der Strecke wird in einem Streckenprofil dargestellt. Die Position der Aufnahmegeräte wird mit verschiedenen Farben aufgezeichnet, das Streckenprofil an sich ist statisch und beinhaltet zusätzliche Renn-Informationen wie z.B. Sprints oder Passhöhen. Die Zusatzinformationen und Profildaten werden von Bildern, GPS-Daten und/oder der Marschtabelle übernommen.

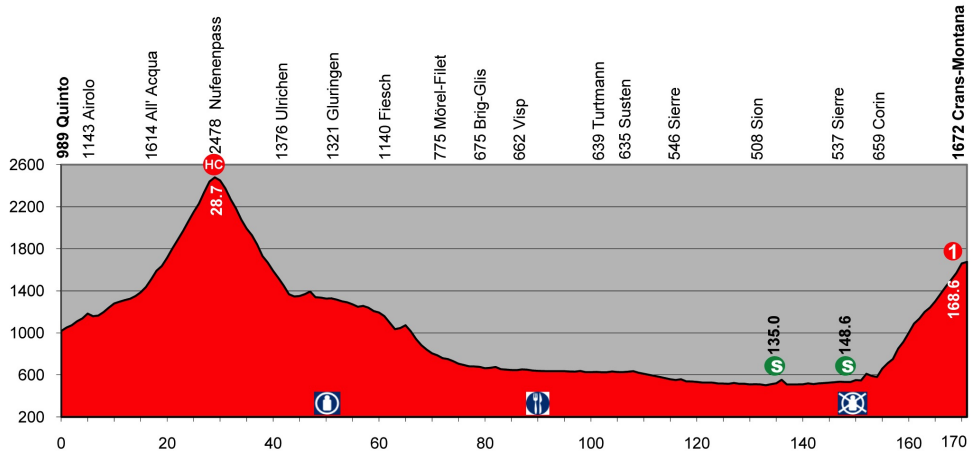


Abbildung 7.3: Streckenprofil aus der Tour de Suisse 2013[1]

7.4.3 Abstände

Jedes Aufnahmegerät zeichnet unter anderem die aktuelle Geschwindigkeit auf, diese wird wie in Abbildung 3 dargestellt. Der Server versucht zusätzlich die Distanz zwischen den Geräten zu berechnen und zeigt diese zusammen mit dem zeitlichen Rückstand an. Der TourSpeaker kann zusätzlich den Abstand in der Figur überschreiben, sofern diese Daten aktueller sind

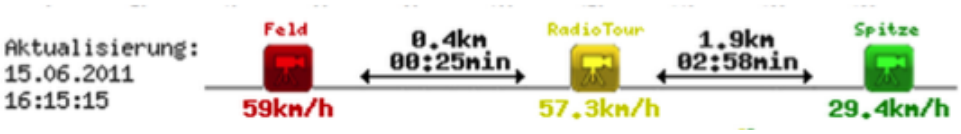


Abbildung 7.4: Abstände zwischen den Aufnahmesystemen

7.4.4 Rennsituation

Die gegenwärtige Rennsituation wird grafisch dargestellt, bei kleineren (Verfolger-) Gruppen werden die Fahrer namentlich aufgelistet, beim Feld wird nur die gesamte Anzahl Fahrer angezeigt. Der Vorsprung bzw. Rückstand ist bei allen Gruppen relativ zur Spitze angegeben. Zusätzlich wird der letzte Wert ebenfalls angegeben, um eine allfällige Tendenz feststellen zu können.

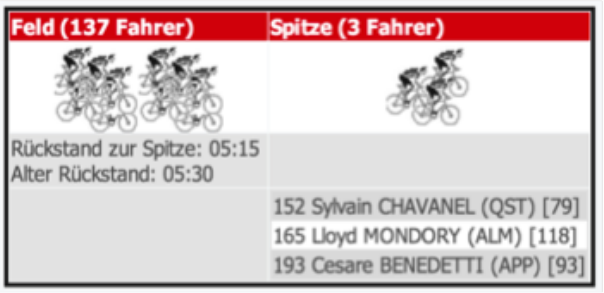


Abbildung 7.5: Die aktuelle Rennsituation dargestellt

7.4.5 Ranglisten

Aus der aktuellen Rennsituation lässt sich eine virtuelle (Live) Rangliste bestimmen. Diese Rangliste wiederspiegelt den aktuellen Stand im Rennen, kann beliebig sortiert werden und beinhaltet die folgenden Spalten:

- Rang
- Startnummer
- Fahrername
- Team
- Land
- Rückstand zur Spitze

Die offiziellen Ranglisten werden von DataSport / Festina / Matsport bezogen.

7.4.6 Kartenausschnitt

Auf einem Kartenausschnitt wird die zurückgelegte Strecke eingezeichnet, die Positionen der Aufnahmegeräte werden auf der Karte farblich verschieden dargestellt. Die Farben entsprechen den anderen Elementen (siehe Abstände und Stream). Neben Google Maps sollen auch andere Karten-APIs verwendet werden könne

Abbildung 7.6: Die Positionen der Aufnahmegeräte

7.4.7 Live Ticker

Eine externe Person kann über ein Webinterface einen Live Ticker führen und Ereignisse während dem Rennen erfassen, diese werden dann sofort auf der Webseite dargestellt.

7.4.8 Replay

Das gesamte Rennen wird aufgezeichnet und kann später in Echtzeit oder im Schnelldurchlauf mit verschiedenen Geschwindigkeiten wiedergegeben werden. Je nach Rennen existieren mehrere Etappen, diese können dann einzeln ausgewählt werden. Die vergangenen Rennen können nach Jahr und Event sortiert und ausgewählt werden.

7.4.9 Werbebanner

Auf der Webseite wird ein Bereich definiert, in welchen Werbebanner dargestellt werden. Diese können in den Einstellungen ein und ausgeschaltet werden.

7.4.10 Mobile Version

Die Webseite soll auch auf Tablets und Smartphones sämtliche Informationen darstellen können. Dabei wird der Ansatz des Responsive Web Design verfolgt wobei die Clients die Webseite entsprechend ihrer Bildschirmgrösse rendern.

Abbildung 7.7: Die Darstellung der Webseite auf die Anzeigegrösse angepasst[2]

7.4.11 Rennstandort (Marschtabelle)

Die Besucher der TourLive Webseite können aufgrund Ihres aktuellen Standortes bestimmen, wann die Spitze am jeweiligen Ort eintreffen wird. Der Standort wird durch die Browser Geolocation API ermittelt damit kann dann die Distanz zur Spitze berechnet werden und aufgrund der Durchschnittsgeschwindigkeit den ungefähren Ankunftszeitpunkt. Die ungefähren Ankunftszeiten gehen auch aus der Marschtabelle hervor. Diese lässt sich ebenfalls anzeigen.

7.4.12 Abstandsentwicklung

Die verschiedenen Aufnahmesysteme können sich unter Umständen weit voneinander entfernen. Die Entwicklung dieses Abstandes während des Rennen bietet für die Radsport-Kenner ein gutes Bild, wie sich das Feld verändert.

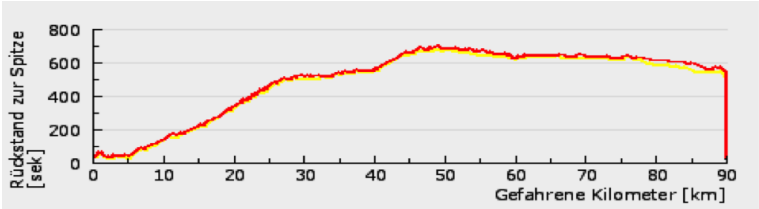


Abbildung 7.8: Abstandsentwicklung nach Rennkilometer in Sekunden

7.4.13 Public API

Sämtliche Grafiken und Daten, welche auf der Webseite angezeigt werden, können auch einzeln über eine festgelegte Schnittstelle als JSON empfangen werden. Dies ermöglicht die Entwicklung von beliebigen Anwendungen durch Dritte. Den Zugang zur öffentlichen Schnittstelle wird durch einen API Key ermöglicht, einen solchen Key erhält, wer sich als Entwickler bei TourLive registriert; dadurch kann die Nutzung überwacht und allenfalls eingeschränkt werden.

7.4.14 Input API

Die aktuelle Rennsituation sowie weitere Informationen zum Rennen werden durch mehrere Android Geräte erfasst. Die Aufnahmegeräte senden sämtliche Informationen via JSON an die Serverschnittstelle.
Weiter werden Daten direkt vom RadioTour Speaker empfangen, diese liefern vor allem Informationen zur Situation im Rennfeld, wie viele Fahrer in welcher Formation fahren.

7.4.15 Nutzungsstatisktik

Über die Nutzung von TourLive wird eine Statistik geführt. Dabei wird auf eine bestehendes Web Analytics System zurückgegriffen, dieses System ist nicht Bestandteil der Arbeit es wird dabei auf Piwik oder Google verwendet.

7.5 Anforderungen Android und Device Management Server

7.5.1 Funktionsumfang bestehendes Aufnahmesystem

Das bestehende TourLive System basiert auf einer Symbian App, die über Nokia Modelle (z.B. Nokia N82 und Nokia X6) betrieben werden. Die Daten (Positionsdaten, Bilder, Videostream) werden über das Mobilfunknetz via ein proprietäres Protokoll zum bestehenden TourLive-Server übertragen. Über eine Gerätestatusseite[3] können einfache Geräteverwaltungsaufgaben durchgeführt werden. Folgende Übersicht bietet einen groben Überblick über das aktuelle System: `http://www.cnlab.ch/tourlive/Radrennen.html#TourLive_Aufnahmesysteme_im_Tour_Tross`

Das Symbian System soll nun auf Android portiert werden, wobei der Funktionsumfang für den ausschliesslichen Einsatz bei Radrennen eingegrenzt wird.

Views

Die aktuelle Symbian App bietet folgende Views. Diese werden in 3 Spalten beschrieben, wobei die rechte Spalte einen Ausblick in die neue Android App und deren Funktionalität darstellt. Die Angaben unter „Neues System“ dienen jedoch nur der Übersicht und im Vergleich zum alten System. Eine exakte Spezifizierung inklusive neuer Funktionalitäten erfolgt im nächsten Kapitel.

Diese View wird beim Start der App angezeigt. Über die beiden Pfeile in der horizontalen Navigationsleiste wird über die verschiedenen Views navigiert.

Altes System - View GPS Info

- Geschwindigkeit [km/h]
- Höhe [m]
- Richtung (in Azimut) / Beschleunigung (Beschleunigung kann weggelassen werden)
- Steigung [%]
- Latitude (Koordinate)
- Longitude (Koordinate)

Neues System

- Geschwindigkeit [km/h]
- Höhe [m]
- Richtung (in Azimut)
- Steigung [%], über 100m gerechnet
- Latitude (Koordinate)
- Longitude (Koordinate)

3. bisherige Gerätestatusseite, `http://www.tourlive.ch/tds12/status.php`, aufgerufen am 27.02.2013

Altes System - View Trip Info

View erreichbar durch einfaches Betätigen der „nach links“-Navigation.

- Zeit Trip [hh:mm:ss]
- Höhe Trip [m]
- Distanz [km]
- Durchschnittliche Geschwindigkeit [km/h]
- UTC Zeit (Weltzeit)
- Datum (aktuelles Datum)

Neues System

- Zeit [hh:mm:ss]
- Höhe [m]
- Distanz [km]
- Durchschnittliche Geschwindigkeit [km/h]

Altes System - View Total Info

View erreichbar durch zweifaches Betätigen der „nach links“-Navigation.

- Zeit Total [hh:mm:ss]
- Zeit Tour [hh:mm:ss]
- Distanz Total [km]
- Distanz Tour [km]
- Höhe Total [m]
- Höhe Tour [m]

Neues System

Diese View fällt weg. Es werden nur noch Daten pro Etappe erfasst.

Altes System - View ODB Info

View erreichbar durch einfaches Betätigen der „nach rechts“-Navigation. On Board Diagnose Bus (war für Benzinverbrauch, Motordrehzahl) Bezieht sich auf ein externes Gerät. Ist im neuen System irrelevant und wird deshalb an dieser Stelle nicht dokumentiert.

Bezieht sich auf ein externes Gerät. Ist im neuen System irrelevant und wird deshalb an dieser Stelle nicht dokumentiert.

Neues System

Diese View fällt weg. Ein OBD wird nicht mehr benötigt.

Altes System - View ODB Verbrauch		**Neues System**
iew erreichbar durch zweifaches Betätigen der „nach links“-Navigation.	Bezieht sich auf ein externes Gerät. Ist im neuen System irrelevant und wird deshalb an dieser Stelle nicht dokumentiert.	Diese View fällt weg.

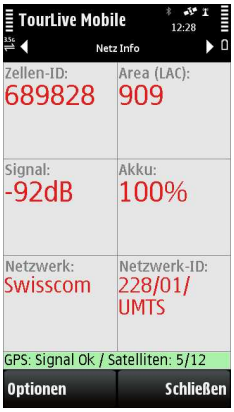
Altes System - View Puls Info		**Neues System**
View erreichbar durch dreifaches Betätigen der „nach links“-Navigation. On Board Diagnose Bus (war für Benzinverbrauch, Motordrehzahl) Bezieht sich auf ein externes Gerät. Ist im neuen System irrelevant und wird deshalb an dieser Stelle nicht dokumentiert.	Bezieht sich auf ein externes Gerät. Ist im neuen System irrelevant und wird deshalb an dieser Stelle nicht dokumentiert.	Diese View fällt weg. Ein OBD wird nicht mehr benötigt.

Altes System - View ODB Verbrauch		**Neues System**
iew erreichbar durch zweifaches Betätigen der „nach links“-Navigation.	Bezieht sich auf ein externes Gerät. Ist im neuen System irrelevant und wird deshalb an dieser Stelle nicht dokumentiert.	Diese View fällt weg. Ein externes Puls Messgerät wird nicht mehr benötigt.

Altes System - View Netz Info

View erreichbar durch vierfaches Betätigen der „nach links“-Navigation.

- Zellen ID (Funkzellen Nummer)
- Area (Location Area)
- Signal (Signalstärke in dB)
- Akku (Akku Ladestatus)
- Netzwerk (Provider)
- Netzwerk ID (Verbindungseigenschaften)
  228 = Mobile Network Code
  01 = Mobile Country Code
  UMTS = Technologie



Neues System

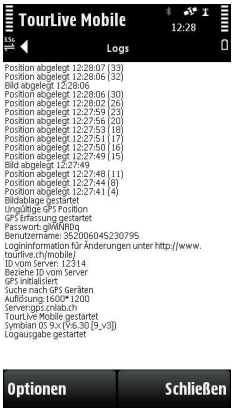
- Zellen ID
- Area
- Signal
- Netzwerk ID
- Technologie (UMTS)
- Datenrate

Zusätzlich

- RTT
- Packet-Loss

Altes System - View Logs

View erreichbar durch fünffaches Betätigen der „nach links“-Navigation.
Log um Fehlerquellen zu eruieren.



Neues System

Diese View wird in einem ähnlichen Stil beibehalten.

Altes System - View Hauptmenü

View wird sichtbar durch das Betätigen des „Optionen“-Buttons.

- Start der Aufzeichnung
- Konfiguration der App (siehe unten)
- Bild auslösen (manuelle Bildauslösung)
- Hintergrundbeleuchtung (Ein / Ausschalten > Stromsparfunktion)
- ODB > nicht mehr relevant
- Puls > nicht mehr Relevant
- GPS [Submenü]
- Reset GPS Trip Daten
- Reset GPS Tour Daten
- Reset alle GPS Daten

Neues System

- Einstellungen
- About

Altes System - View Konfiguration

View wird sichtbar durch das Betätigen des „Konfiguration“-Buttons im Optionen-Menü.

- Benutzername
- Grösse (irrelevant in der neuen Version)
- Gewicht (irrelevant in der neuen Version)
- Alter (irrelevant in der neuen Version)
- Geschlecht (irrelevant in der neuen Version)
- GPS Intervall (0 – 600 sec)
- GPS Speichern Intervall (0 – 600 sec)

Neues System

- Benutzername
- GPS Intervall (0 – 60 sec)
- GPS Speichern Intervall (0 – 600 sec) (Wird mit GPS Intervall zusammengefasst)
- Bild Intervall (0 – 60 sec)

- Bild Intervall (0 – 600 sec)
- Einheiten (km – Meilen - ..) (nur noch MKS Einheiten standardmässig)
- Zusatzgerät (irrelevant in der neuen Version)
- Video Streaming (Ein / Aus)
- Bilder drehen (Ein / Aus)
- Offline Mode (Ein / Aus) (irrelevant in der neuen Version, wird automatisch gemacht)
- GPS Dateiformat (TourLive + Google Earth, Google Earth) (irrelevant in der neuen Version, wird serverseitig gerechnet)

- Bild Intervall (0 – 60 sec)
- Video Streaming (Ein / Aus)
- Bilder drehen (Ein / Aus)
- Automatischer Start (Neu verschiedene Modi)
- Hintergrundbeleuchtung (Normal / Dauernd an)
- Bildauflösung
- Bildmodus

- Automatischer Start (Ein / Aus)
- Hintergrundbeleuchtung (Normal / Dauernd an)
- Einfaches Beenden (Ein / Aus) (irrelevant in der neuen Version)
- Schreibe Logfile (Ein / Aus) (irrelevant in der neuen Version, wird immer geschrieben)
- Heartbeat Ton (Lautstärke) (irrelevant in der neuen Version, keine Töne mehr)
- Fehlerton (Lautstärke) (irrelevant in der neuen Version, keine Töne mehr)
- Bildauflösung (1600 x 1200 , . . .)
- Bildmodus (Portrait / Landscape)

Zusätzlich

- Kritischer Akkustand
- Betriebsmodus
- TourLive Server Primär
- TourLive Server Sekundär
- Device Management Server
- Kamera automatisch ausrichten
- Videoauflösung
- Renndistanz Korrektur
- Gespeicherte Daten löschen

7.5.2 Anforderungen – Neues Aufnahmesystem

Das neu zu entwickelnde Aufnahmesystem basiert grundsätzlich auf der Funktionalität der bestehenden Symbian App. Wobei auf die Unterstützung externer Geräte (Puls-Messung, Onboard Diagnose Bus (OBD)), verzichtet werden kann. Einen Überblick über die übernommenen Funktionen bietet die dreispaltige Beschreibung im Kapitel „Funktionsumfang bestehendes Aufnahmesystem [Symbian App]“.

Hardware / Software - Aufnahmesystem

Bei den Aufnahmegeräten wird gemäss Vorgabe des Auftraggebers auf das Betriebssystem Android ab Version 4.x (Ice Cream Sandwich) gesetzt. Der Prototyp wird auf zwei Samsung Nexus Geräten getestet. Mittelfristig soll die zu entwickelnde App auf Geräten von verschiedenen Herstellern (Samsung, HTC, ...) betrieben werden können. Im Rahmen der Arbeit ist die App auf zwei bis drei weiteren Geräten zu testen.

Betriebsmodi / Device Management

Die Geräte sollen sowohl über ein Management-Portal, analog zum zur bisherigen Verwaltungsseite [4] , fernverwaltet als auch über ein „Einstellungen“-Menü direkt am Gerät konfiguriert werden können.

Daraus resultieren zwei Betriebsmodi: „managed“ und „unmanaged“. Beim ersten App-Start nach der Installation der App, wird der Benutzer gefragt, in welchem Modus das Aufnahmesystem betrieben werden soll. Die beiden Modi lassen sich am Gerät selber jederzeit ändern. Wählt der Benutzer beim ersten App-Start innerhalb von 10 Sekunden keinen Modus, wird das Gerät automatisch in den Betriebsmodus „managed“ versetzt und eine Standardkonfiguration vom Device Management Server bezogen.

Bei allen weiteren App-Starts werden die Einstellungen vom letzten Betrieb übernommen sofern der Betriebsmodus zuvor „unmanaged“ war. War der Betriebsmodus auf „managed“ eingestellt, so wird die Konfiguration vom Management Server bezogen. Ist dieser nicht verfügbar, wird der Modus auf „unmanaged“ gesetzt und die existierenden Einstellungen verwendet.

Device Management Portal

Über das Management-Portal können Geräteprofile beziehungsweise ein Standardprofil definiert werden, die bei Verwendung des „managed“ – Modus verwendet werden. Die Geräte werden über eine eindeutige Identifikationsnummer identifiziert. Ebenfalls Bestandteil des Management-Portals ist Einsicht bzw. Abrufen der Applikationslogs zur Eruierung allfälliger Fehlerquellen. Das Applikationslog soll jedoch auch direkt am Gerät eingesehen und mit den üblichen Android „Share“-Optionen (z.Bsp.: Versand per E-Mail,...) geteilt werden können.

Betriebsmodus „managed“
Der Betriebsmodus „managed“ erlaubt eine vollständige Fernverwaltung der Android App. Wird das Gerät eingeschaltet, bezieht es vom Device Management Server eine App-Konfiguration. Sämtliche Einstellungen werden über das Managementportal vorgenommen. Am Gerät selber können keine Einstellungen mehr vorgenommen werden. Die Einstellungen werden als read-only angezeigt. Es ist am Gerät jedoch möglich, den Betriebsmodus zu ändern.

Betriebsmodus „unmanaged“
Der Betriebsmodus „unmanaged“ funktioniert genau umgekehrt. Die Einstellungen des Gerätes werden zwar auf dem Device-Management-Server angezeigt, können jedoch nicht verändert werden. Die Einstellungen werden lokal am Gerät vorgenommen. Über das Device Management Portal ist es jedoch möglich, den Betriebsmodus zu ändern

Notfallwiederherstellung

Über einen weiteren Management-Kanal (z.Bsp.: SMS) soll eine Notfallwiederherstellung des Systems ermöglicht werden. Beispielsweise wenn das Gerät über das

4. Verwaltungsseite, `http://www.tourlive.ch/tds11/status.php`, aufgerufen am 27.02.2013

Mobilfunkdatennetz nicht mehr erreichbar ist. Dieser Notfallwiederherstellungsservice wird ein eine Zweit-App ausgegliedert, damit beim Crash der Haupt-App zumindest der Notfallwiederherstellungsservice noch funktioniert.

Betriebslog

Geloggt wird unter anderem folgendes:

- ganze Kommunikation mit dem Server
- Aufnahme Start/Stop
- Exceptions

Funktionalität TourLive App

Allgemein

Die nachfolgend beschriebenen Daten die gesammelt werden (Bilder, Positionen, etc.) werden einem Rennen bzw. einer Etappe zugeordnet. Eine Etappe (z.Bsp.: Etappe 1 des Rennens Tour de Suisse) ist einem Rennen (z.Bsp. Tour de Suisse) zugeordnet. Es gibt Rennen, welche nur eine Etappe haben. Dies als Einführung ins Gesamtsystem. Am Aufnahmesystem beziehungsweise in der Android-App ist es nicht möglich Rennen oder Etappen zu definieren. Die gesammelten Daten werden versehen mit einem Zeitstempel und einer eindeutigen Geräteidentifikationsnummer an den TourLiveServer gesendet. Das matching zum entsprechenden Rennen beziehungsweise der Etappe erfolgt erst auf dem Server.

Zur Aufnahme und Übertragung der Daten werden vier verschiedene Aufnahmestart-Modi unterschieden:

- Manuell (direkt über den Hauptscreen der App)
- Zeitbasiert
- Fernverwaltung (start über das Device Management Portal)
- Bei Aktivierung einer externen Stromquelle (Feuerzeuganzünder im Auto,..)

Positionsaufnahme

Die primäre Funktion des Aufnahmesystems besteht in der Aufnahme von Geopositionsdaten, deren Weiterverarbeitung und der Übermittlung an den TourLive-Server. Über das GPS-Modul des Mobilfunkgerätes werden folgende Daten ermittelt:

- Aktuelle Position [GPS Longitude / Latitude]
- Aktuelle Höhe (GPS Höhe) [m]
- Aktuelle Zeit (GPS Timestamp) [unix_time]
- Geschwindigkeit (wird gelesen) [km/h]
- Richtung (in Azimut, wird berechnet) [°]
- Steigung (über die letzten 100m, wird berechnet) [%]
- Anzahl Satelliten mit denen das Aufnahmegerät verbunden ist

Für die Aufzeichnung der Positionsdaten per GPS stehen folgende Einstellungen zur Wahl:

- Aufnahmeintervall der Positionsdaten (1, 2, 5, 10, 15, 30, 60 sec)

Bildaufnahme

Ein wesentlicher Bestandteil des Aufnahmesystems ist die Übertragung von Bildmaterial in Form von einzelnen Bildern oder einem Videostream. Folgende Funktionalität muss dabei beachtet werden:

- Anpassung der Bildauflösung adaptiv an die verfügbare Datenraten (optional)
- Bilder sollen automatisch, in der korrekten Ausrichtung an den Server geschickt werden (Stichwort Gerätesensoren)

Desweitern sollen bezüglich Bildaufnahme verschiedene Einstellungen zur Auswahl stehen:

- Einzelbilder mit konfigurierbaren Aufnahmeintervallen (1, 2, 5, 10, 15, 30, 60 sec)
- Videostream (echter Videostream oder schnelles Aneinanderreihen von Einzelbildern)
- Front-/Back-Kamera muss wählbar sein

Power Management

Während der Tour wird das Aufnahmegerät über den Zigarettenanzünder mit Strom versorgt. Fällt diese Stromversorgung aus beziehungsweise fällt der Akku-Stand unter einen kritischen Grenzwert, so soll ein Energiesparmodus aktiviert werden um das Aufnahmesystem möglichst lange Betriebsbereit zu halten. Dieser Stromsparmodus umfasst:

- Vergrösserung GPS-Intervall
- Verkleinerung Bild-Auflösung (weniger Daten müssen übertragen werden)
- Vergrösserung Daten-Übertragungs-Intervall an TourLive Server (Overhead wird gespart)
- Deaktivierung Video-Streaming
- Herunterfahren der Bildschirmhelligkeit

Alarming Funktionen

Treten Probleme auf, so soll auf dem Telefon sowie in der Management Konsole darüber informiert werden. Als zu meldende Probleme gelten folgende:

- GPS nicht verfügbar oder zu wenig Satelliten
- Keine Mobile-Verbindung 2 Minuten
- Smartphone wird nicht mehr geladen (Stromzufuhr unterbrochen)
- Smartphone Akkustand ist unter 50%, 25%, 10%

Kommunikation mit dem TourLive Server

Die gesammelten Textdaten werden über eine offene, freie Schnittstelle (z.Bsp. *HTTP/JavaScript Object Notation (JSON)*) an einen Webservice (TourLive Server) übertragen. Eine ähnliche Lösung soll für das Bildmaterial sowie für den Videostream gefunden werden.

Kommunikation mit dem Device-Management Server

Die Kommunikation mit dem Device-Management Server erfolgt ebenfalls über eine offene, freie Schnittstelle. Zusätzlich soll die Möglichkeit einer Notfallwiederherstellung über einen Drittkanal (z.Bsp.: SMS) realisiert werden.

Mobile Performance Informationen

Die Streckenführung der Tour de Suisse wird jedes Jahr neu festgelegt. Dabei wird selbstverständlich keine Rücksicht auf die Netzabdeckung der Mobilfunknetze genommen. Gerade bei Bergetappen abseits der Zivilisation muss mit einer mässigen Netzabdeckung gerechnet werden. Die Analyse der Verbindungseigenschaften (Packet Loss, RTT,..) soll Aufschluss darüber bringen, wann und wo keine Daten übertragen werden konnten und dient somit einer allfälligen Fehleranalyse. Konkret sollen folgende Daten erfasst werden:

- Zellennummer (Mobilfunkantenne)
- Location Area Code (LAC)
- Mobile Netzwerk Code (MNC) (Provider)
- Mobile Country Code (MCC)
- Signal (Signalstärke in dB) und/oder „Striche“
- Übertragungsstandard [GPRS, EDGE, UMTS, LTE,..])
- Download Datenrate
- Upload Datenrate
- RTT
- Packet Loss (Dup Acks)

7.5.3 Nichtfunktionale Anforderungen

Neben der effektiven App-Funktionalität müssen folgende nichtfunktionalen Anforderungen beachtet werden.

Sicherheit

In Bezug auf Vertraulichkeit und Integrität werden keine speziellen Anforderungen gestellt. Die Datenübertragung erfolgt unverschlüsselt. Das Aufnahmegerät muss sich am TourLive Server nicht explizit authentisieren. In Bezug auf Verfügbarkeit sind die Anforderungen hoch. Während dem Rennen soll es keine Lücken ohne Daten von mehr als 5 Minuten geben. Können während einer bestimmten Zeitspanne keine Daten übertragen werden, sollen diese gepuffert und zu einem späteren Zeitpunkt übertragen werden.

Ausfallsicherheit

Die gesammelten Daten (Positionen, Bilder,..) werden parallel auf dem lokalen Gerätespeicher abgelegt. Erreicht dieser 80% der verfügbaren Kapazität werden automatisch alte Eventdaten gelöscht. Über dieses lokale Caching wird sichergestellt, dass die gesammelten Daten aufgrund technischer Probleme bei der Datenübertragung nicht verloren gehen.

Anforderungen ans GUI

Das GUI soll intuitiv ohne Anleitung oder Einführung bedient werden können.

Mehrsprachigkeit

In der Grundversion werden die Sprachen Deutsch und Englisch unterstützt. In einem späteren Schritt soll man die Möglichkeit haben, die App um weitere Sprachen einfach zu ergänzen.

7.6 Evaluation Webframework

7.6.1 Kriterienkatalog

Die folgende Kriterien wurden zusammen mit dem Industriepartner definiert und festgelegt:

- Open Source Software, keine Lizenzgebühren
- Beispiel- und Referenzprojekte vorhanden
- Hohe Performance und stabile Verfügbarkeit auch bei hoher Auslastung
- Skalierbar für mehrere Rennen und Etappen
- Datenbankanbindung an *MariaDB*
- Weiterentwicklung durch cnlab Software AG muss möglich sein

Optionale, gewünschte Anforderungen:

- Vorkenntnisse in der betreffenden Programmiersprache
- Gleiche Technologie für *API*, Webseite und Datenverarbeitung
- Umfangreiche Dokumentation und Tutorials (Community[5])

7.6.2 Mögliche Lösungen

Django

Django ist ein auf Python basiertes Webframework. Es wird eingesetzt bei berühmten Webseiten wie z.B. Pinterest, Instagram und The Washington Post. Django beinhaltet einen OR Mapper, Templates zur Darstellung und einen URL Dispatcher als Controller.

Spring Framework

Spring ist ein Framework für die Erstellung von Webprojekten. Es basiert auf Java Technologien und fördert Dependency Injection und aspektorientierte Programmierung.

JavaServer Faces

JSF ist ein Framework-Standard für Webapplikationen in Java. Es verwendet die Java Servlet Technologie und benötigt einen Servlet Container für den Betrieb. Mit

5. Die Community ist die Verbreitung einer Technologie sowie die Hilfsbereitschaft in Foren und Portalen. Dies ist äusserts schwierig zu beurteilen und kann sich auch schnell ändern.

JSF können Komponenten für User Interfaces einfach in Webseiten eingebaut werden.

Symfony

Symfony ist ein Open Source PHP Webframework und verfolgt das MVC Pattern. Die Zuordnung der Models geschieht dabei über die Namensgleichheit in Singular und Plural und nicht über Konfigurationsdateien (convention over configuration). Weiter können durch Konsolenapplikationen Anwendungen generiert werden.

Ruby on Rails

Rails ist ein Webframework geschrieben in Ruby. Es ist geprägt von den Prinzipien „don't repeat yourself" und „convention over configuration". Ruby on Rails besteht aus fünf Modulen, jedes dieser Module übernimmt gewisse Funktionen, so z.B. der Action Mailer versendet und empfängt E-Mails.

7.6.3 Nutzwertanalyse

Kriterium	Gewicht	Django		Spring MVC		JSF		Symfony		Rails	
		Wert	Total	Wert	Total	Wert	Total	Wert	Total	Wert	Total
Keine Lizenzkosten	10	1	10	1	10	1	10	1	10	1	10
Referenzprojekte	8	1[6]	8	1[7]	8	1[8]	8	1[9]	8	1[10]	8
Performance	10	*									
Skalierbarkeit	8	*									
DB und ORM	9	1	9	1	9	1	9	1	9	1	9
Weiterentwicklung	7	1	7	1	7	1	7	1	7	1	7
Vorkenntnisse	5	1	5	5	25	3	15	2	10	0	0
Gleiche Technologie	3	1	3	1	3	1	3	1	3	1	3
Freie Dokumentation	3	1	3	1	3	1	3	1	3	1	3
Total			45		65		55		50		40

Abbildung 7.9: Nutzwertanalyse

Erläuterung zu Performance und Skalierbarkeit

Bezüglich der Performance gibt es verschiedene Ansichten zu den verschiedenen Frameworks. Da es aber zu allen obigen Frameworks grosse Projekte gibt, kann davon ausgegangen werden, dass die Performance und Skalierbarkeit für das TourLive Projekt ausreicht.

Erläuterung zur Gewichtung der Kriterien

Die verschiedenen Kriterien wurden in einer Skala von 1 – 10 gewichtet, wobei 10 das wichtigste Kriterium darstellt. Gemäss cnlab Software AG dürfen für das Framework keine Lizenzkosten anfallen und die Webseite muss auch unter Last stabil verfügbar sein. Daher werden diese beiden Kriterien mit dem höchsten Gewicht 10 versehen. Da das System einen stark wachsenden Datenbestand haben wird, ist die Implementation von Datenbanken ebenfalls von grosser Bedeutung. Referenzprojekte dienen als Beweis für die Skalierbarkeit und die Performance des jeweiligen Frameworks. Für die Studierenden ist der Zugang zu freier Dokumentation sowie Vorkenntnisse relevant, jedoch nicht zwingend erforderlich. Wünschenswert

ist ebenfalls, dass für das gesamte Webprojekt dasselbe Framework verwendet werden kann. Diese Punkte werden daher mit den Gewichten 3-5 bewertet.

DB und ORM

Ruby on Rails verfolgt das Ziel möglichst abstrakt mit Datenbanken zu arbeiten und kann deshalb problemlos mit verschiedenen Datenbanksystemen arbeiten. Gleiches gilt für den Ansatz bei Django, dort werden verschiedene Datenbankadapter zur Verfügung gestellt. Auch Spring arbeitet z.B. mit Hibernate als ORM problemlos mit verschiedenen Datenbanken.

Weiterentwicklung durch cnlab Software AG

Nach Abschluss der Arbeit wird das Projekt durch die cnlab Software AG weiterentwickelt. Daher muss bei der Auswahl des Frameworks darauf geachtet werden, dass eine weitere Entwicklung möglich ist.

Schlussfolgerungen

Da die Frameworks sehr ähnliche Ansätze verfolgen und aktuell eine grosse Entwicklergemeinschaft geniessen, sind die Unterschiede, abgesehen von der Programmiersprache welche als Grundlage dient, sehr klein. Entscheidend sind schliesslich die Vorkenntnisse: Java ist diejenige, für die bereits das grösste Vorwissen besteht. Daher sind für die engere Wahl die beiden Lösungen Spring Framework und JSF im Rennen. Da wir in einer kleinen Projektarbeit bereits mit JSF gearbeitet haben, möchten wir nach unseren eher negativen Erfahrungen damit von einer Entwicklung mit JSF absehen und auf das Spring MVC Framework setzen.

7.6.4 Anmerkung zum Framework

Das Java Spring Framework baut auf dem Prinzip *Dependency Injection* auf. Dependency Injection wird unter anderem mit dem Begriff Inversion of Control zusammengefasst und im folgenden Abschnitt erläutert. Weiter fördert Spring gute Programmierpraktiken wie z.B. die Trennung von Model, View und Controller oder die Verwendung von Objekt relationalen Mappern beim Einsatz von Datenbanken in der Persitenzschicht.

Dependency Injection

Die Grundlage für Dependency Injection basiert auf dem Prinzip, dass möglichst wenig Abhängigkeiten eines Objekts von einem anderen Objekt besitzt werden. Die Abhängigkeiten sollen zur Verfügung gestellt werden. So kann unabhängig von der Implementation auf die Schnittstellen zugegriffen werden. Dies erlaubt zum Beispiel einen späteren Austausch der Abhängigkeit. In Spring übernimmt der Dependency Injection Container diese Aufgabe. Die benötigten Abhängigkeiten können entweder mit einer Annotation im Code oder durch Konfiguration in einer xml Datei deklariert werden. Die nachfolgende Grafik stammt von Martin Fowlers Artiekel zum Thema Dependency Injection [Fow04].

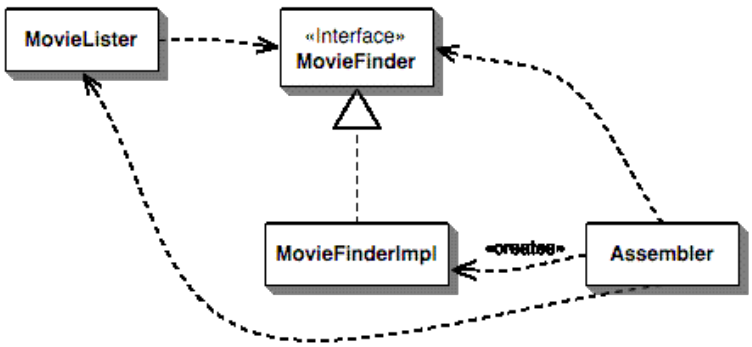


Abbildung 7.10: Dependency Injection nach Martin Fowler [Fow04]

Die Funktionsweise kann am einfachsten anhand des Beispiels von Martin Fowler in Abbildung 7.10 aufgezeigt werden. Die Klasse MovieLister zeigt und filtert Filme. Dabei spielt die Quelle woher die Filme kommen keine Rolle. MovieLister benötigt nur eine Liste von Filmen (Abhängigkeit). Die MovieFinderImpl Klasse bietet genau diese Möglichkeit und zeigt dies durch die Implementation des MovieFinder Interfaces an. Der Assembler fügt die Abhängigkeit in den MovieLister ein (Injection).

In dieser Weise können die Filme in einer Excel Datei, in einer Datenbank oder in einer Textdatei gespeichert werden, die Klasse MovieLister kann in jedem Fall verwendet werden.

7.6.5 Spring Framework

Für die Entwicklung von Webapplikationen bietet Spring das integrierte Modul *MVC* an. Damit werden die grundlegenden Funktionalitäten einer Webapplikationen bereits abgedeckt.

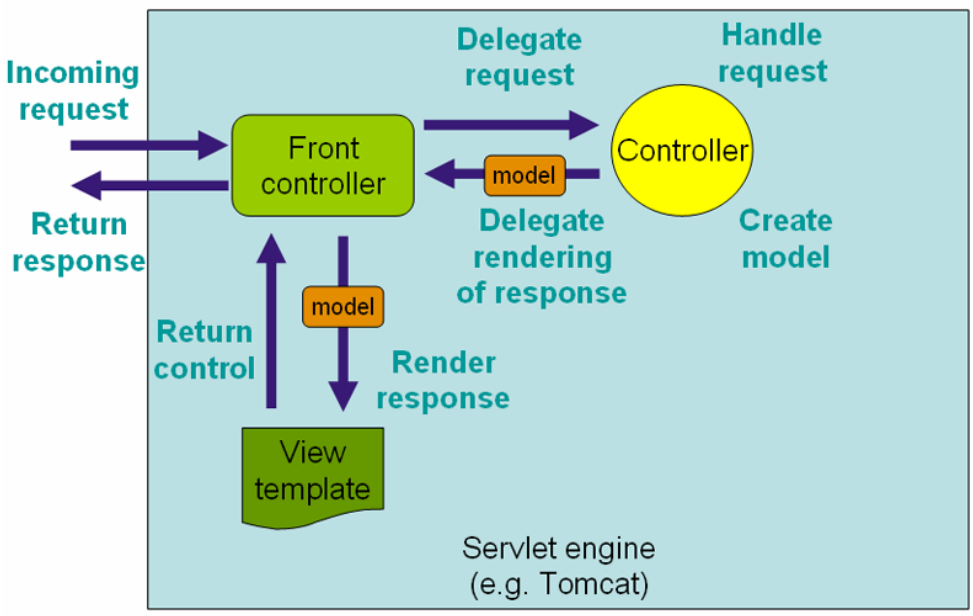


Abbildung 7.11: Aufbau von Spring [Sam11]

Die Anfragen werden durch den Front Controller empfangen und an den passen-

den Controller der Applikation weitergeleitet. Dieser fügt die angeforderten Daten (Model) hinzu. Das Model wird dann in die View Templates eingefügt und via den Front Controller an den Client zurückgeschickt.

Dieses abstrakte Vorgehen kann durch ein konkretes Beispiel sehr gut dargestellt werden. Eine Anfrage wird durch den Aufruf einer URL im Browser ausgelöst (z.B. http://www.tourlive.ch/rennen). Der Front Controller sucht den Applikations Controller für die Ressource */rennen*. Bereitet das Model, in diesem Fall alle sichtbaren Rennen, vor und sendet dieses zurück. Der Front Controller sucht den View Resolver und übergibt das Model (alle sichtbaren Rennen). Dort wird das Model zur Darstellung gebracht und schlussendlich via den Front Controller an den Browser zurückgesendet. Für jede Ressource gibt es also einen Controller, dabei können aber auch generische Elemente wie z.B. der Rennname in der Ressource verwendet werden (z.B. http://www.tourlive.ch/rennen/tourdesuisse).

Für die Einarbeitung in das Spring Framework wurden verschiedene Quellen verwendet. Alle Angaben dazu befinden sich im Literaturverzeichnis. Der obige Abschnitt ist eine Zusammenfassung von Inhalte aus den SpingSource Tutorials [Sam11] und Mkyong [Yon11] .

7.7 Externes Design - Android App

Folgendes Dokument beschreibt das externe Design der Android App. Als Designgrundlage der Mockups dienen primär die in Zusammenarbeit mit cnlab Software AG festgelegten Requirements. Als Referenz stand die bestehende Symbian-App zur Verfügung.

7.7.1 Einführung

In einem ersten Schritt wurden die Android GUI Principles[6] studiert und aufgrund dieser ein Design-Konzept für die neue App erstellt.

Folgend ein Beispiel einer Applikation, welche den GUI Guidelines entspricht und ähnliche Komponenten benutzt wie die zukünftige TourLiveApp.

6. Android GUI Principleshttp://developer.android.com/design/get-started/principles.html, zuletzt besuch 21.03.2013

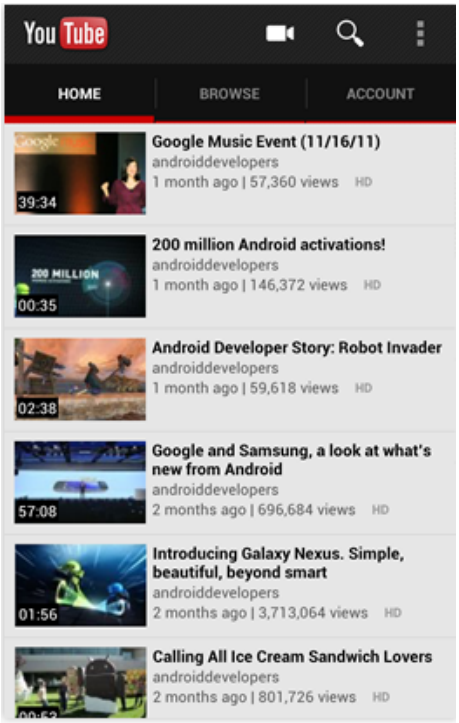


Abbildung 7.12: Standard Ansicht nach den Android GUI Guidelines

7.7.2 Grobübersicht TourLiveApp

Das Design der Android App gliedert sich grundsätzlich in ein Dashboard sowie 5 Informationsansichten die über Tabs erreichbar sind. Über den “Settings”-Button lassen sich Konfigurationen vornehmen sowie den About Screen anzeigen.

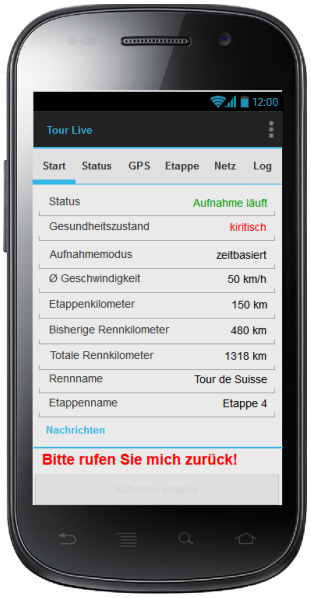
Abbildung 7.13: Übersicht über alle Views

Folgend werden alle Ansichten im Detail beschrieben. Dabei wurde auf mit Hilfe folgender Legende gearbeitet um die einzelnen Werte zu beschreiben: * berechneter

Wert der stets aktualisiert wird + Wert der aufgrund einer Messung eruiert wird (Android Service Provider) ° App-Einstellung die angezeigt wird

7.7.3 Startscreen - Ansicht

Wird die App gestartet öffnet sich die „Start“-View. Sie dient als „Dashboard“ und gibt Informationen zum allgemeinen Zustand der App.



Nutzen / Zielperson

Diese View dient dazu, dem Fahrer und Beifahrer, eine Übersicht zu geben um im Falle eines Problems zu intervenieren. Mit Hilfe der Nachrichten kann der Systemüberwacher mit den Autoinsassen kommunizieren kann.

Erläuterung der Ansicht

Status: Zeigt an, ob Daten an den TourLive Server gesendet werden oder nicht

Gesundheitszustand: Zusammenfassung der Ansicht „Status“

Aufnahmemodus (°): manuell, zeitbasiert, fernverwaltet, externe Stromquelle

Durchschnittsgeschwindigkeit (+): Die Durchschnittlichegeschwindigkeit wird ermittelt, über alle bereits gemessenen Locations, anhand deren Geschwindigkeit

Etappenkilometer (*): Die in der aktuellen Etappe zurückgelegten Kilometer

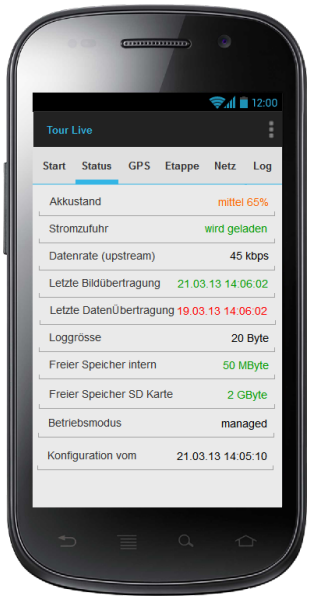
Bisherige Rennkilometer (*): Die ganze Strecke, über mehrere Etappen verteilt, die zurückgelegt wurde. Dieser Wert wird vom TourLiveServer übermittelt.

Totale Rennkilometer (*): Die totale Anzahl Kilometer, die in diesem Rennen zurückgelegt wird. Wird vom TourLiveServer an die App übermittelt.

Nachrichten (*): Das Management Device Portal hat die Möglichkeit, Nachrichten via das Mobile an die Fahrer zu übermitteln. Diese werden rot dargestellt.

7.7.4 Status - Ansicht

Allgemeiner Zustand des Systems. Orange eingefärbte Zustände können längerfristig zu Problemen führen, rot eingefärbte Zustände sind kritisch und verhindern das einwandfreie Funktionieren des Systems. Es folgt eine Beschreibung der einzelnen Zustände:



Nutzen / Zielperson

Zielperson ist auch hier der Laie (Fahrer / Beifahrer der auf einen Blick sehen möchte ob die App korrekt funktioniert. Aus diesem Grund werden problematische Werte orange und kritische Werte rot hinterlegt.

Erläuterung der Ansicht

Akkustand (*): gut (70% - 100%, grün), mittel (30% - 70%, orange), schwach (0% - 30%, rot)

Stromzufuhr (*): nicht vorhanden (rot) / wird geladen (grün)

Datenrate (upstream) (*): Datenrate mit der Daten an den Server übertragen werden.

Letzte Bildübertragung (*): Zeitpunkt der letzten Bildübertragung (< 2min grün, 2min< x <3min orange, >3min rot)

Letzte GPS Übertragung: Zeitpunkt, wann die letzten Positionsdaten übertragen wurden (< 2min grün, 2min< x <3min orange, >3min rot)

Loggrösse (*): Grösse der Logdatei

Freier Speicher intern (*): Nicht belegter Telefonspeicher (5030% - 50%, orange, 0% - 30%, rot)

Freier Speicher SD Karte (*): Freier Platz auf der SD Speicherkarte (50% - 100% grün, 30% - 50% orange, 0% - 30% rot)

Betriebsmodus (°): managed (Gerät wird vom Device Management Portal verwaltet) oder unmanaged (Gerät wird über die lokalen Geräteeinstellungen verwaltet)

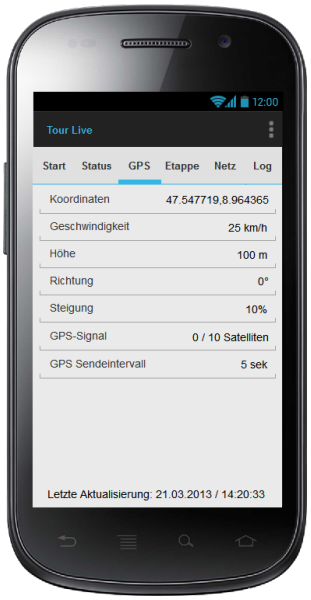
7.7.5 Daten - Ansichten

Die folgenden Ansichten zeigen alle Informationen, welche an den Server übermittelt werden. Am unteren Rand der Ansicht wird jeweils die „letzte Aktualisierung“ angezeigt. Dies entspricht dem Zeitpunkt wann die Daten zum letzten Mal von den verschiedenen Service Providern bezogen bzw. berechnet und an den TourLiveServer übertragen wurden. Dieser Übertragungsintervall kann in den Einstellungen definiert werden.

Nutzen / Zielperson

Diese Ansichten helfen vorrangig dem Techniker allfällige Probleme zu erkennen. Beispielsweise um zu überprüfen ob die korrekten Werte an den Server übertragen werden.

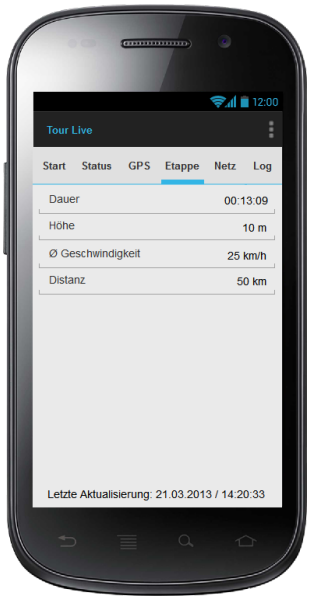
GPS – Übersicht



Erläuterung der Ansicht
Koordinaten (*): Empfangene GPS Koordinaten im Format WGS84
Geschwindigkeit (*): Empfangene Geschwindigkeit vom Satellit gemessen
Höhe (*): Höhe, vom Satellit ermittelt
Richtung (+): Richtung in Grad, Abweichung von Norden, wichtig um zu erkennen, ob gleich wie die Windrichtung
Steigung (+): Aktuelle Steigung über 100 Meter gemessen
GPS-Signal (*): Verfügbarkeit der GPS-Satelliten
GPS-Intervall (°): In welchem Intervall die GPS-Daten getracked werden.
Letzte Aktualisierung (*): Zeitpunkt, der letzten Aktualisierung der Daten

Etappen – Übersicht

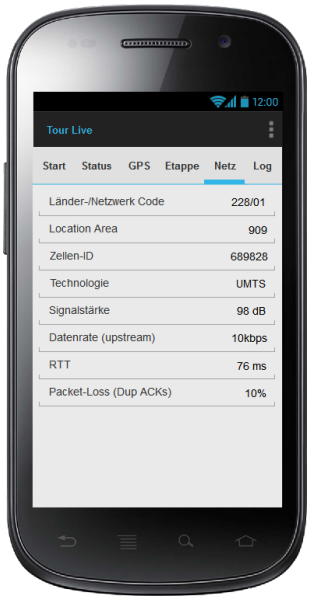
Alle Informationen rund um die gesammelten Etappendaten.



Erläuterung der Ansicht
Dauer (*): verstrichene Zeit während der aktuellen Etappe seitdem die Aufnahme gestartet wurde.
Höhe (*): zurückgelegte Höhe, berechnet über alle bisher gesammelten Positionsdaten
Durchschnittsgeschwindigkeit (*): Die Durchschnittliche Geschwindigkeit wird über alle der aktuellen Etappe gemessenen Positionsdaten ermittelt.
Distanz (*): Die in der aktuellen Etappe zurückgelegten Kilometer. Wird ebenfalls über sämtliche der aktuellen Etappe ermittelten Positionsdaten berechnet.
Letzte Aktualisierung (+): Zeitpunkt, der letzten Aktualisierung der Ansicht

Netz – Übersicht

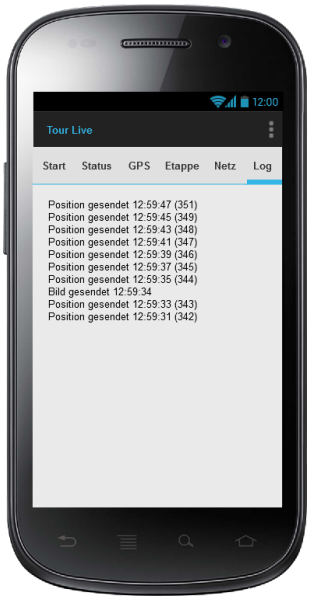
Folgende Ansicht dient zur Eruierung von Übertragungsproblemen. Zur anschliessenden Analyse der Qualität der Datenverbindung werden gleichzeitig zu den Positionsdaten auch Informationen zum vorhandenen Mobilfunknetzwerk gesammelt.



Erläuterung der Ansicht
Länder-/Netzwerk Code (+): Mobile Country[7] und Network Codes[8]
Location Area (+): Eine Location-Area wird vom Mobilefunkanbieter definiert und fast mehrere Zellen (Antennen) zusammen.
Zellen-ID (+): Zeigt die Zellennummer an mit der das Gerät aktuell verbunden ist.
Technologie (+): Mobiles Kommunikationsprotokoll
Signalstärke (+): Zeigt die aktuelle Mobilfunksignalstärke an
Datenrate (upstream) (+): Gemessene Datenrate bei der Bildübertragung
RTT (+): mit ICMP gemessene Round-Trip-Time
Packet-Loss (dup ACKs) (+): Packetverlust beim Upload der Daten

Log

Zur Fehlerbehebung allfälliger Fehler werden diverse Aktionen geloggt.



Folgende Aktionen werden geloggt:

- Bild versendet
- Daten versendet
- Einstellungen synchronisiert
- Aufgetrettene Exceptions

7.7.6 Einstellungen

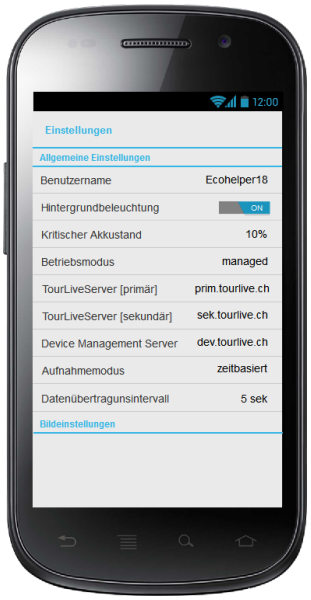
Über den „Settings“-Button in der rechten oberen Ecke werden die App-Einstellungen angepasst. Ist am Geräte ein physischer „Settings“-Knopf vorhanden, können die Einstellungen auch über diesen geöffnet werden.

7. Mobile Country Code, `http://en.wikipedia.org/wiki/Mobile_country_code`, aufgerufen am 22.03.2013
8. Mobile Network Code, `http://en.wikipedia.org/wiki/Mobile_Network_Code`, aufgerufen am 22.03.2013

Nutzen / Zielperson

Die Einstellungen werden entweder am Gerät (Betriebsmodus: unmanaged) oder über den DeviceManagementServer (Betriebsmodus: managed) vorgenommen. Die Einstellungen dienen vorgängig technisch versierten Personen um entsprechende Konfigurationen vorzunehmen.

Erläuterung der Ansicht
Einstellungen: Einstellungen der App
About: Zeigt den About Screen an



Erläuterung der Ansicht

Benutzername: Benutzername zur Identifizierung des Gerätes für den Benutzer. Innerhalb des TourLiveSystems wird mit einer eindeutigen Geräteidentifikationsnummer gearbeitet

Hintergrundbeleuchtung: Aktivierung / Deaktivierung der permanenten Hintergrundbeleuchtung

Kritischer Akkustand: Grenzwert „kritischer“ Akkuzustand für Power-Saving Funktionalität: nach Unterschreiten dieses Wertes werden Stromsparmassnahmen ergriffen

Betriebsmodus: Betriebsmodus „managed“ oder „unmanaged“

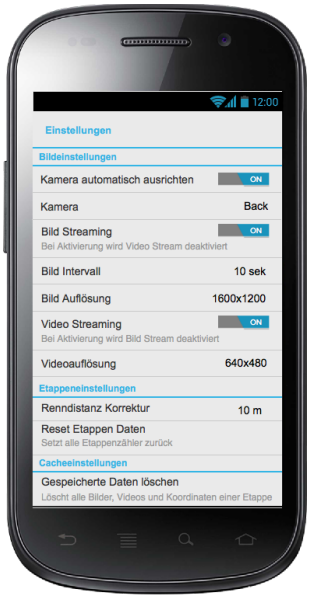
TourLiveServer [primär]: Primärer TourLiveServer an den die Daten gesendet werden

TourLiveServer [sekundär]: Sekundärer TourLiveServer der als Backup dient, falls der primäre TourLiveServer nicht mehr verfügbar ist

Device Management Server: Device Management Server mit dem die Gerätekonfiguration synchronisiert wird.

Aufnahmemodus: Aufnahmemodus (zeitbasiert, manuell, fernsteuerung, aufgrund aktiver Stromzufuhr)

Datenübertragungsintervall: Zeitintervall in dem die Positionsdaten/Netzwerkdaten aufgezeichnet, an den Server gesendet und die App-Views aktualisiert werden



Automatische Ausrichtung der Kamera: Ist der „Switch“ deaktiviert, so kann über ein Untermenü die Ausrichtung der Kamera explizit definiert werden.
Bild- / Video - Streaming: Aktivierung / Deaktivierung Übertragung von Bildern / Videos (es kann nur Video-Streaming oder Bilder-Streaming aktiv sein)
Kamera: Zeigt die aktuelle Ausrichtung der Kamera an.
Bild Intervall: Zeitintervall in dem die Bilder an den TourLive Server übertragen werden
Bild Auflösung: Auflösung der Bilder (Abhängig vom Gerätetyp)
Video Streaming: Aktivierung / Deaktivierung Übertragung vom Video-Stream (es kann nur Video-Streaming oder Bilder-Streaming aktiv sein)
Videoauflösung: Auflösung des Video-Streams (Abhängig vom Kamera-Modell)

7.8 Werkzeuge und Entwicklungsumgebung

Für die drei Teilprojekte TourLive Server, Device Management Server und Aufnahmesystem wurde jeweils die Java Programmiersprache verwendet. Dennoch gibt es Unterschiede bei den Entwicklungsumgebungen, diese sind im folgenden Abschnitt erläutert.

7.8.1 TourLive Server

- Entwicklungsumgebung: Spring Tool Suite (STS), basierend auf Eclipse, `http://www.springsource.org/sts`
- Framework: Spring, Java Webframework, `http://www.springsource.org/`
- Mockups: Balsamiq Mockup für erste Entwürfe, `http://www.balsamiq.com/`
- Versionierungssystem: git, `http://git-scm.com/`

Installation und Deployment

Um das Projekt zu builden und auf einem Webserver zu betreiben sind folgende Schritte notwendig.

- Projekt aus GitHub klonen oder komprimierte zip Datei von der CD entpacken
- In der pom.xml Datei ganz unten ein Profil für die Entwicklungsumgebung erstellen und vergewissern, dass der Datenbankserver läuft
- Projekt mit Maven builden:

```
# bash oder andere Shell starten und
# ins Projektverzeichnis wechseln
~ $> cd RadioTourWebsite

# Dependencies herunterladen und Tests starten
RadioTourWebsite $> mvn install -P meinprofil

# Projekt kompilieren und als deployable war exportieren
RadioTourWebsite $> mvn package -P meinprofil

# Das War File liegt dann im Verzeichnis
RadioTourWebsite $> ~/RadioTourWebsite/target/ba-1.0.0-BUILD-
    SNAPSHOT.war
```

Quellcode 7.1: Build und Test mit Maven

- Die *war* Datei kann nun auf verschiedene Arten auf dem Tomcat deployed werden, am einfachsten ist das Autodeployment von Tomcat

```
# war Datei aus dem target Ordner in den Tomcat webapps
# Ordner kopieren
RadioTourWebsite $> cp target/ba-1.0.0-BUILD-SNAPSHOT.war /var/lib
    /tomcat7/webapps/ROOT.war

# Bemerkung: Der Pfad zum webapps Ordner kann sich je
# nach Plattform unterscheiden. Wird die Datei in
# ROOT.war umbenannt so wird die Applikation auf dem
# Domainroot (http://tlng.cnlab.ch/) deployed.
# Danach Tomcat neu starten
RadioTourWebsite $> /etc/init.d/tomcat7 restart
```

Quellcode 7.2: Deployment auf Tomcat

Sämtliche Videos und Bilder werden nicht direkt über die Webapplikation ausgeliefert, sondern über einen konfigurierbaren Pfad (im Profil ganz unten im pom.xml) abgelegt und von dort aus zur Verfügung gestellt. Der empfohlene Webserver für den Betrieb ist ein Tomcat für die Auslieferung der Bilder und Videodateien kann ein anderer Server verwendet werden, um den Hauptwebserver zu entlasten. Dabei kann aber auch ein Tomcat verwendet werden, die Aussage, dass der Tomcat Server statische Inhalte langsamer ausliefert, wirkt sich gemäss Mark Thomas im Betrieb kaum auf die Performance aus [Tho10].

7.8.2 Device Management Server

- Entwicklungsumgebung: Spring Tool Suite (STS), basierend auf Eclipse, `http://www.springsource.org/sts`
- Framework: Spring, Java Webframework, `http://www.springsource.org/`
- Mockups: Evolus Pencil für erste Entwürfe, `http://pencil.evolus.vn/`
- Versionierungssystem: git, `http://git-scm.com/`

Installation und Deployment

Um den Device Management Server zu builden, können die gleichen Schritte ausgeführt werden wie beim TourLive Server. Was alles gemacht werden muss ist im Kapitel 7.8.1 beschrieben.

7.8.3 Aufnahmesystem

- Entwicklungsumgebung: Android Development Tools (ADT), basierend auf Eclipse, `https://dl-ssl.google.com/android/eclipse/`
- Mockups: Evolus Pencil für erste Entwürfe, `http://pencil.evolus.vn/`
- Versionierungssystem: git, `http://git-scm.com/`

Android Version

Eine Anwendung wird für eine spezifische Android Version entwickelt und getestet. Somit kann garantiert werden, dass das Verhalten der Anwendung immer gleich ist. In dieser Arbeit ist dies die Version 4.0 mit dem Versionsnamen Ice Cream Sandwich[9]. Die Entwicklung auf einer Version schliesst jedoch nicht aus, dass die Anwendung in neueren Versionen nicht mehr lauffähig ist.

7.9 Kontaktadressen

Die Studierenden

Simon Stäheli
Im Langacher 21
8805 Richterswil

Patrizia Heer
Friedbergstrasse 13b
8512 Thundorf

Florian Bentele
Teufener Strasse 113
9000 St. Gallen

Der betreuende Professor

Prof. Dr. Peter Heinzman
Obere Bahnhofstrasse 32b
8640 Rapperswil

Der Industriepartner

Lukas Frey, cnlab Software AG
Obere Bahnhofstrasse 32b
8640 Rapperswil

9. Android Ice Cream Sandwich, `http://www.android.com/about/ice-cream-sandwich/`, aufgerufen am 29.04.2013

7.10 Poster

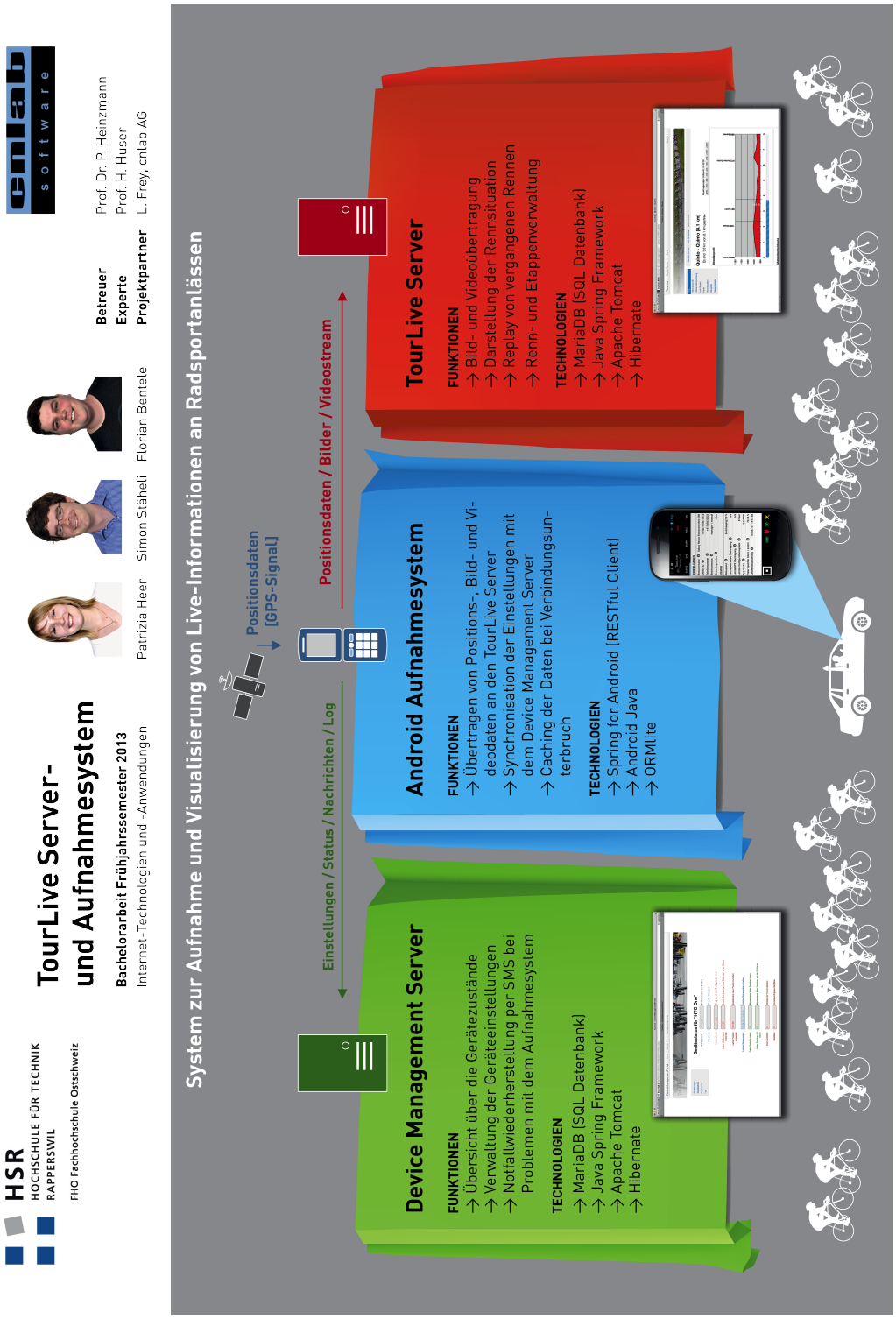


Abbildung 7.14: Poster zur Bachelorarbeit

7.11 Inhaltsverzeichnis der CD

Auf der beigelegten CD befinden sich folgende Ordner:

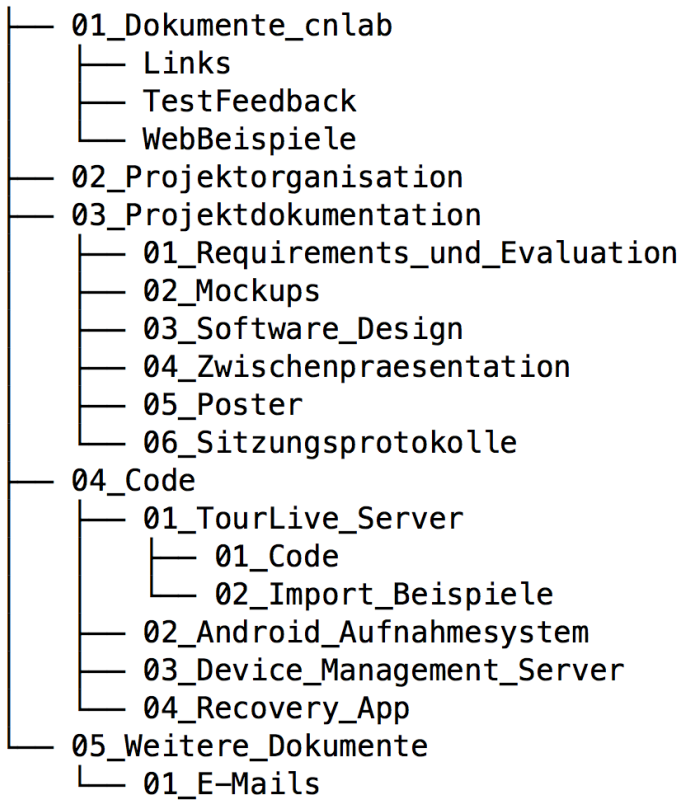


Abbildung 7.15: Inhaltsverzeichnis CD