

Bachelorarbeit, Abteilung Informatik

TourLive

Hochschule für Technik Rapperswil

Frühlingssemester 2018

Autoren	Dominik Good und Urs Forrer
Betreuer	Prof. Dr. Peter Heinzmann, Patrick Eichler
Experte	Dr. Thomas Siegenthaler
Gegenleser	Prof. Dr. Andreas Rinkel
Arbeitsperiode	19.02.2018 – 15.06.2018
Arbeitsumfang	360 Arbeitsstunden bzw. 12 ECTS pro Student
Link	github.com/tourlive

Abstract

Das TourLive System ermöglicht die Aufzeichnung und Darstellung des Renngeschehens bei Radrennen für Zuschauer, Rennorganisatoren, RadioTour Speaker und Mannschaftsleiter. Es besteht aus dem GPS Aufnahmesystem, der RadioTour Anwendung und einem Web-Anzeigesystem. Im Rahmen der Bachelorarbeit wurde das bereits bestehende TourLive System erweitert.

Die «RadioTour Android Anwendung» wurde in der Studienarbeit im Herbstsemester 2017 von Grund auf neu strukturiert und ein Prototyp implementiert. Im Rahmen der Bachelorarbeit wurden Systeme zur Real-Time Anzeige der RadioTour Daten implementiert. Ferner wurde der Prototyp perfektioniert.

Das in der Bachelorarbeit entwickelte System umfasst das TourLive-Frontend (Desktop- und Smartphone-Darstellung) und das Backend (TourLive-Server mit TourLive-API). Die TourLive-API dient zur Persistierung und zur Bereitstellung der Daten auf dem zentralen Server. Im Frontend werden die Daten für interessierte Zuschauer graphisch aufbereitet und übersichtlich dargestellt. Das Frontend ist auf Basis des JavaScripts-Frameworks React und dem State Management Redux implementiert.

Durch die Verwendung eines client-seitigen JavaScript-Frameworks, Caching und Thread-Pools ist es möglich auch eine grosse Last von gleichzeitigen Benutzer zu tragen. Bei einem Lasttest mit bis zu 1'000 gleichzeitigen Nutzern betrugen die Antwortzeiten stets weniger als eine Sekunde. An der diesjährigen Tour de Suisse wird sich zeigen, ob das gesamte System die effektive Nutzlast tragen kann.

Das System soll nach der Tour de Suisse durch weitere Funktionen ergänzt werden (z.B. Integration TourLive-Wo-Anwendung, TourLive-Abstandsanwendung mit Analyse)

Management Summary

Ausgangslage

Der Radsport ist in der Schweiz und auch in vielen anderen Teilen der Welt ein populärer Sport. So nehmen an der alljährlichen «Tour de Suisse» im Juni über 180 Rennfahrer aus aller Welt teil. Cnlab stellt für Radsport seit Jahren ein umfangreiches System zur Verfügung. Dieses besteht aus einer **Webseite** (tourlive.ch), einer **Wo-Anwendung** (tourlive.ch/wo, Anzeige von GPS-Trackern auf einer Karte) sowie einer API, welche als Kommunikationsinterface zwischen den Komponenten dient. Als Aufnahmekomponente dient eine Android-Anwendung (genannt RadioTour Anwendung). In dieser App bildet der RadioTour Speaker während dem Rennen die aktuelle Situation ab.

Die Teile wurden durch cnlab und in verschiedenen Studien- und Bachelorarbeiten entwickelt und aktualisiert. So wurde die RadioTour Anwendung im Herbstsemester 2017/2018 von Grund auf neugebaut.

Ziel

Das Ziel dieser Bachelorarbeit ist es, die im letzten Semester gebaute RadioTour Anwendung zu optimieren und die Daten der App auf einer Webanwendung zur Verfügung zu stellen. Jeder Zuschauer soll am Strassenrand die aktuelle Rennsituation verfolgen können.

Vorgehen und Technologie

Das Projekt wurde anhand von RUP (Rational Unified Process) mit den vier Phasen Inception, Elaboration, Construction und Transition geplant und zusammen mit Gedanken von Scrum umgesetzt.

In der Startphase des Projekts wurde die RadioTour Anwendung (Studienarbeit HS 2017/2018) im Bereich Performance stark optimiert. Nach eingehender Analyse des bestehenden Systems bzw. der Konkurrenten wurden in Zusammenarbeit mit den Betreuern die Anforderungen für die neue Webanwendung «TourLive Web» definiert. Mittels Prototypen wurde die einzelnen Funktionen aufgezeigt und diskutiert, bevor sie entsprechend implementiert wurden.

In regelmässigen Sitzungen mit dem Dozenten Prof. Dr. Peter Heinzmann und dem Assistenten Patrick Eichler wurden kontinuierlich Änderungswünsche aufgenommen, diskutiert und umgesetzt. Neue Versionen der «TourLive Web» Anwendung wurden in regelmässigen Abständen auf den Testserver deployed.

Bei den Technologien wurde auf eine Mehrsprachigkeit gesetzt. Im Backend wird Java zusammen mit dem Play Framework eingesetzt, während im Frontend mit React und Redux gearbeitet wird.

Ergebnisse

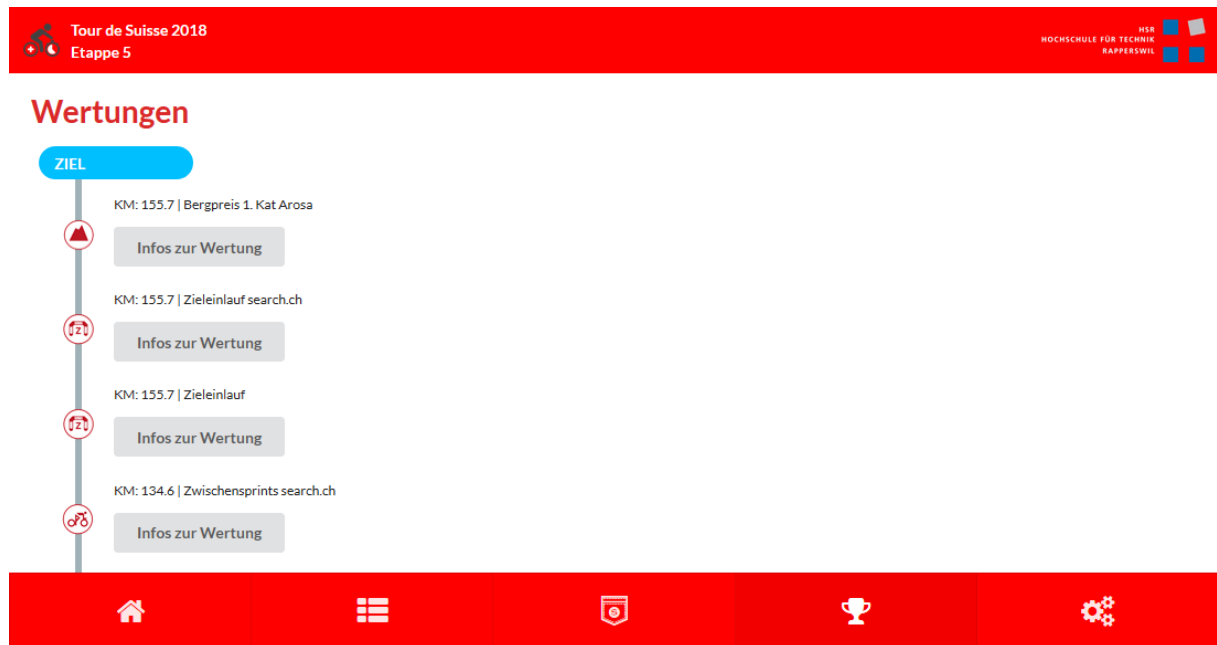


Abbildung 1: Zuschaueranwendung

Aus der Bachelorarbeit resultierten eine neue Webanwendung für den Zuschauer, eine API (Schnittstelle zwischen Webanwendung und RadioTour App) sowie eine Admin-Anwendung zur Verwaltung letzterer. Die beiden Frontendkomponenten (Zuschaueranwendung & Admin-Anwendung) wurden im clientseitigen JavaScript-Framework React umgesetzt. Sie verwenden für die Darstellung Semantic UI[1] als GUI-Framework und erfüllen damit die Anforderung einer dynamischen Webseite für Smartphones, Tablets sowie Desktops. Für die API wurde das Playframework eingesetzt.

Die Rennen, die Etappen, sowie GPX-Daten können über die gesicherte Admin-Anwendung verwaltet werden. Die aktuellen Updates der RadioTour-App (Android-Anwendung) werden ebenfalls gesichert mit einem Passwort an den Server übermittelt.

Ausblick

Die Android «RadioTour Anwendung» (aus der Studienarbeit) sowie die neu entwickelten Komponenten werden an einem Radrennen in Gippingen sowie an der Tour de Suisse eingesetzt, welches am 09. Juni 2018 startet.

Die Webanwendung «TourLive Web» befindet sich auf einem guten Stand. Nächste Schritte wären sicher eine Mehrsprachigkeit für die Zuschauer an den Rennen in den Französisch und Italienisch sprechenden Teilen der Schweiz. Mit einer umfangreicheren Datenanalyse könnte die Webanwendung «TourLive Web» auch für Journalisten interessant werden. Auch die Ablösung der bestehenden «TourLive Wo-Anwendung» wäre durchaus ein nächster möglicher Schritt. Nicht zu Letzt würde es sich sehr eignen die Anwendung durch einen interaktiven Teil der Zuschauer zu ergänzen (wie in der Aufgabenstellung angedacht aber aus Zeitgründen nicht umgesetzt).

Aufgabenstellung

Im Folgenden wird die Aufgabenstellung zur TourLive Bachelorarbeit im Detail erläutert.

TourLive

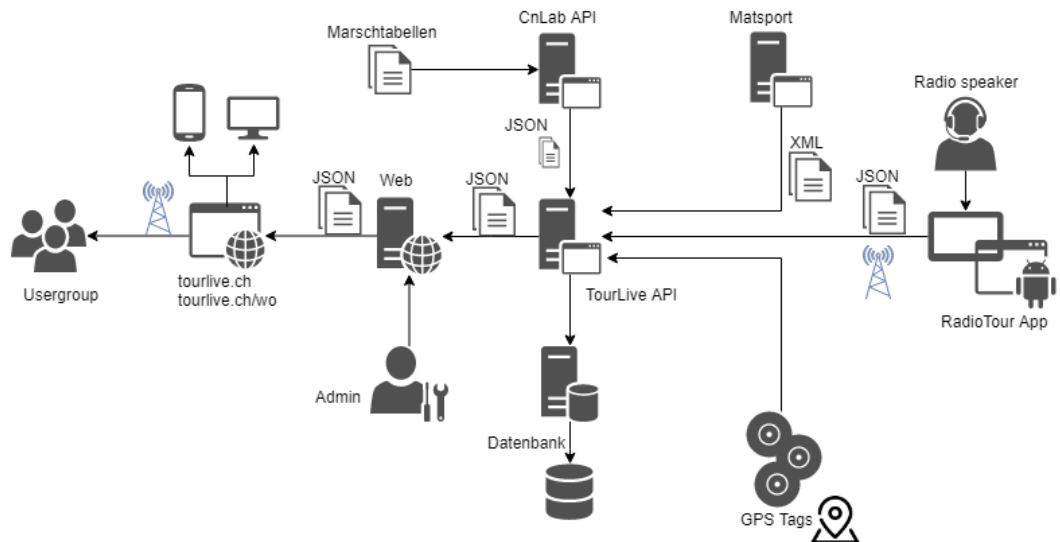
Studiengang:	Informatik (I)
Semester:	FS 2018
Durchführung:	Bachelorarbeit
<hr/>	
Fachrichtung:	Internet-Technologien und -Anwendungen
Institut:	INS
Studentengruppe:	Urs Forrer, Dominik Good
<hr/>	
Betreuer:	Prof. Dr. Peter Heinzmann
Assistent:	Patrick Eichler
Experte:	Dr. Thomas Siegenthaler
Korreferent:	Prof. Dr. Andreas Rinkel

Ausgangslage

Der RadioTour Speaker ist die zentrale Renninformationsstelle bei Profi Radrennen. Er informiert über Zeitabstände und Zusammensetzung von Fahrergruppen, Zwischenklassen und Wertungen (Sprint, Bergpreis), spezielle Ereignisse (z.B. Sturz, Defekt, Aufgaben) und allgemeine Angaben zum Rennen (z.B. Aufgebote zu Dopingkontrollen). Diese Daten sammelt der RadioTour Speaker via Funk von seinem Chrono Team, welches die Fahrergruppen auf dem Motorrad begleitet.

Im Rahmen der Studienarbeit haben Urs Forrer und Dominik Good eine neue Android-Anwendung für den RadioTour Speaker erstellt.

Im Rahmen der Bachelorarbeit sollen zuerst Korrekturen, Optimierungen und Erweiterungen in der RadioTour Speaker Anwendung erledigt werden. Im Hauptteil der Arbeit sollen die Daten aus der RadioTour Speaker Anwendung über eine responsive Web-Anwendung der Öffentlichkeit verfügbar gemacht werden.



TourLive API

- Aufnahme von Änderungen in der RadioTour App (JSON)
- Erstellen von JSON Dateien, die bei Gebrauch an den Webserver weitergegeben werden
- Erstmaliger Bezug bei Tour Start der Informationen von der CnLab API (JSON)
- Bezug der aktuellen Daten pro Etappe von Matsport
- Bezug der GPS Tags (JSON)

Benutzer Notizen

- Spielaspekt, der die Benutzer dazu animiert, auf die Webseite zu gehen
- Schlechte Datenraten berücksichtigen
- Wartezeiten kurz halten

Notizen zu Web-Server

- so statisch wie möglich, da viele Benutzer auf einmal darauf zugreifen werden
- Automatische Updates der Daten
- Responsive

Abbildung 2: BA «TourLive» Übersicht

Ziel

Nach dieser Bachelorarbeit soll die RadioTour Speaker Anwendung an der Tour de Suisse 2018 eingesetzt werden können. Interessierte Kreise – vor allem die Zuschauer am Strassenrand sollen über eine Webanwendung die Renninformationen abrufen können. Die Anwendung soll auf einer Plattform realisiert werden, welche auch die bei der Tour de Suisse typische Spitzenlast verkraftet. Das User Interface soll für Mobilegeräte (iOS, Android) und Desktop-Browser (Google Chrome, Firefox, Safari, Internet Explorer, Edge) ausgelegt sein.

Die aktuellen Tour de Suisse Wertungen sollen mit zwei nicht-offiziellen Wertungen ergänzt werden: Die Preisgeldwertung soll für jeden Fahrer anzeigen, welche Preisgelder er pro Etappe und insgesamt gewonnen hat. Die Spitzengruppenwertung soll für jeden Fahrer anzeigen, wie lange er pro Etappe und insgesamt in der Spitzengruppe präsent war.

Mit einer noch zu definierenden «Interaktionsanwendung» sollen die Zuschauer am Strassenrand animiert werden, die TourLive-Anwendung auf ihren Smartphones zu nutzen. Beispielsweise könnte mit einer «Max Applaus Funktion» über sein Smartphone virtuell Applaus gesendet werden, wenn die Fahrergruppen vorbeifahren.

Aufgaben

Einarbeitung

- Perfektionierung von Anwendung und Bericht der Studienarbeit
- Genaue Definition der Preisgeld- und Spitzengruppenwertungen
- Aufzeigen verschiedener Server Lösungen

Analyse

- Abschätzung der Leistungsanforderungen für die Web-Plattform
- Analyse / Vergleich von verschiedenen Web-Hosting Lösungen
- Erarbeitung eines Technologie-Stacks der Anwendung
- Studium verschiedener Lösungen zur Darstellung von Renninformationen

Design

- Entwurf einer Gesamtarchitektur
- Entwurf des Backend (API)
- Entwurf der Webanwendung für die Zuschauer am Strassenrand
 - Anzeige der Renninformationen (Gruppen und Zeitabstände)
 - Anzeige der Position der Rennkolonne auf Karte (vgl. www.tourlive.ch/wo/)
 - Anzeige der Position der Rennkolonne auf dem Streckenprofil (Höhenprofil)
 - Anzeige von allen Wertungen
- Optional: Entwurf einer Interaktionsanwendung für die Zuschauer (z.B. «virtuelle Applaus» Anwendung)

Realisierung

- Realisierung des Backends (API)
- Einführung der POST von der RadioTour Tablet Anwendung
- Realisierung der neuen Web-Anwendung
- Realisierung einer «virtueller Applaus» Anwendung

Testing

- Import von Radiotour- und Wertungsdaten aus alten Rennen
- Testing der nicht offiziellen Preisgeld- und Spitzengruppenwertungen
- Ausführliche User Tests durch Betreuer und Radio Tour Speaker Steve Bovay

Referenzen und Beispiele

1. Hinweise zur Durchführung von Studienarbeiten <https://drive.switch.ch/index.php/f/506363283>
2. Florian Bentele & Daniel Stucki, Studienarbeit, Abteilung Informatik Android Applikation RadioTour, Hochschule für Technik Rapperswil Frühjahrssemester 2012. <http://docplayer.org/2494735-Android-applikation-radiotour.html>
3. Steve Bovet, RadioTour Speaker Tour de Suisse, «The feeling of being in the middle of the race is indescribable» <http://www.tourdesuisse.ch/en/news/news-detail/news/the-feeling-of-being-in-the-middle-of-the-race-is-indescribable/>
4. TourLive Anwendung, <http://www.tourlive.ch>
5. TourLive Wo-Anwendung <http://www.tourlive.ch/wo>
6. TourLive Infrastruktur: <https://drive.google.com/open?id=1n0LIK2FBYhw1mkYXsPZ3DB-YiWzi6sh4>

Danksagung

Wir möchten an dieser Stelle uns bei allen denjenigen bedanken, die uns während der Bachelorarbeit unterstützt haben. Besonderen Dank gilt folgenden Personen:

- Prof. Dr. Peter Heinzmann, Dozent an der HSR Rapperswil und Mitarbeiter von cnlab, für die Betreuung und fachliche Unterstützung rund um Radrennen während der Bachelorarbeit
- Patrick Eichler, Assistent an der HSR und Mitarbeiter von cnlab, für die Betreuung und der technischen Hilfe rund um das bestehende TourLive System
- Lukas Frey, cnlab Mitarbeiter, für die Durchführung unserer beiden Lasttest
- Steve Bovay, RadioTour Speaker, für das Testing der Android Anwendung gemeinsamen mit dem Betreuer

Ein Dank geht ebenfalls an all diejenigen Personen welche im Vorfeld der Abgabe die Arbeit gegengelesen und entsprechend korrigiert haben.

Inhaltsverzeichnis

1	EINLEITUNG	11
1.1	AUSGANGSLAGE	11
1.2	VISION	14
1.3	AUFBAU DER ARBEIT	14
2	ANFORDERUNGSSPEZIFIKATION.....	15
2.1	FUNKTIONALE ANFORDERUNGEN (USE CASES).....	15
2.2	USE CASE DIAGRAMM	19
2.3	NICHT FUNKTIONALE ANFORDERUNGEN	22
2.4	STAKEHOLDERS	23
3	EVALUATIONEN	24
3.1	FRONTEND FRAMEWORK	24
3.2	DEPLOYMENTLÖSUNG (SERVERLÖSUNG)	28
3.3	MAP LIBRARY IM FRONTEND	36
3.4	SPEICHERUNG DER DATEN	39
4	USER INTERFACE DESIGN	41
4.1	PROTOTYP	41
4.2	UMSETZUNG	42
5	SOFTWARE ARCHITEKTUR.....	44
5.1	SYSTEMÜBERSICHT.....	44
5.2	ABLÄUFE IM SYSTEM	46
5.3	DOMAINMODELL	48
5.4	DATENMODELL	51
5.5	DEPLOYMENT	52
6	REALISIERUNG	54
6.1	FRONTEND SOWIE ADMIN-ANWENDUNG	55
6.2	BACKEND (TOURLIVE API).....	63
6.3	RADIO TOUR ANWENDUNG (ANDROID APP).....	69
7	TESTING	70
7.1	AUSGANGSLAGE FÜR DEN 1. LASTTEST	70
7.2	LASTTEST 1, 24.05.2018	70
7.3	AUSGANGSLAGE FÜR DEN 2. LASTTEST	71
7.4	LASTTEST 2, 04.06.2018	72
8	RESULTATE	73
8.1	RESULTAT	73
8.2	OFFENE PUNKTE	73
8.3	AUSBlick	74

I.	ANHANG.....	75
9	PROJEKTMANAGEMENT	76
9.1	PROJEKTMITGLIEDER	76
9.2	VORGEHEN.....	76
9.3	PHASEN	77
9.4	ZEITPLAN	80
9.5	RELEASES	81
9.6	MEILENSTEINE	81
9.7	RISIKEN.....	82
9.8	INFRASTRUKTUR / ENTWICKLUNGSUMGEBUNG.....	83
9.9	QUALITÄTSMASSNAHMEN.....	85
10	PERSÖNLICHE BERICHTE	87
10.1	URS FORRER.....	87
10.2	DOMINIK GOOD	88
11	ZEITAUSWERTUNGEN.....	89
11.1	ZEITAUFWÄNDE PRO KALENDERWOCHE	89
11.2	ZEITAUFWÄNDE PRO KOMPONENTE	90
11.3	ZEITAUFWÄNDE PRO PHASE.....	91
12	ARCHIVDATEI	92
13	INSTALLATIONSANLEITUNG.....	93
13.1	REQUIREMENTS	93
13.2	DOCKER-UMGEBUNG.....	93
13.3	INITIALE KONFIGURATION	95
14	BENUTZERANLEITUNGEN	97
14.1	LOGIN / LOGOUT	97
14.2	HOME.....	98
14.3	EINSTELLUNGEN	98
14.4	VORHANDENE DATEN	99
14.5	VERWALTEN DER STATISCHEN DATEN.....	100
14.6	VERWALTEN DER DYNAMISCHEN DATEN.....	101
15	LITERATURVERZEICHNIS	102
16	ABBILDUNGSVERZEICHNIS	104
17	GLOSSAR.....	106

1 Einleitung

Radrennen werden von vielen interessierten Sportsfans aktiv verfolgt. Um Ihnen Einblicke, Informationen und Ergebnisse zur Verfügung zu stellen wurden bereits einige Projektarbeiten gestartet, die genau das ermöglichen sollen.

1.1 Ausgangslage

Während einem Radrennen werden viele Informationen zwischen dem RadioTour Speaker und den anderen Beteiligten (Kommissäre, etc.) ausgetauscht. Die RadioTour Anwendung bietet die Möglichkeit diese Informationen (Fahrergruppen, erreichte Wertungen, Zeitabstände, usw.) festzuhalten. Die Abbildung 3 stellt eine typische Rennsituation dar und zeigt die Einbindung aller relevanten Akteure der RadioTour Anwendung.

Neben den Rennfahrern in der Abbildung 3, welche sich auf die Tour de Suisse bezieht, sind diverse Motorradfahrer und Autos in einer Kolonne unterwegs. Jede Gruppe, welche während eines Rennens vorhanden ist, besteht neben den Radfahrern zusätzlich aus einem Motorrad zusammen mit dessen Fahrer, welche im dauernden Kontakt mit dem RadioTour Speaker stehen. Der RadioTour Speaker (platziert in einem Auto am Ende der Kolonne) hält die über Funk erhaltenen Informationen in der Tablet Anwendung fest und besitzt somit immer einen aktuellen Überblick über die Rennsituation. Zusätzlich zu den Motorradfahrern und dem RadioTour Speaker sind noch Kommissäre, Werbekolonen, Mannschaftsfahrzeuge und Streckensicherungsfahrzeuge unterwegs. Zur Übersicht wurden diese in der Abbildung 3 nicht abgebildet.

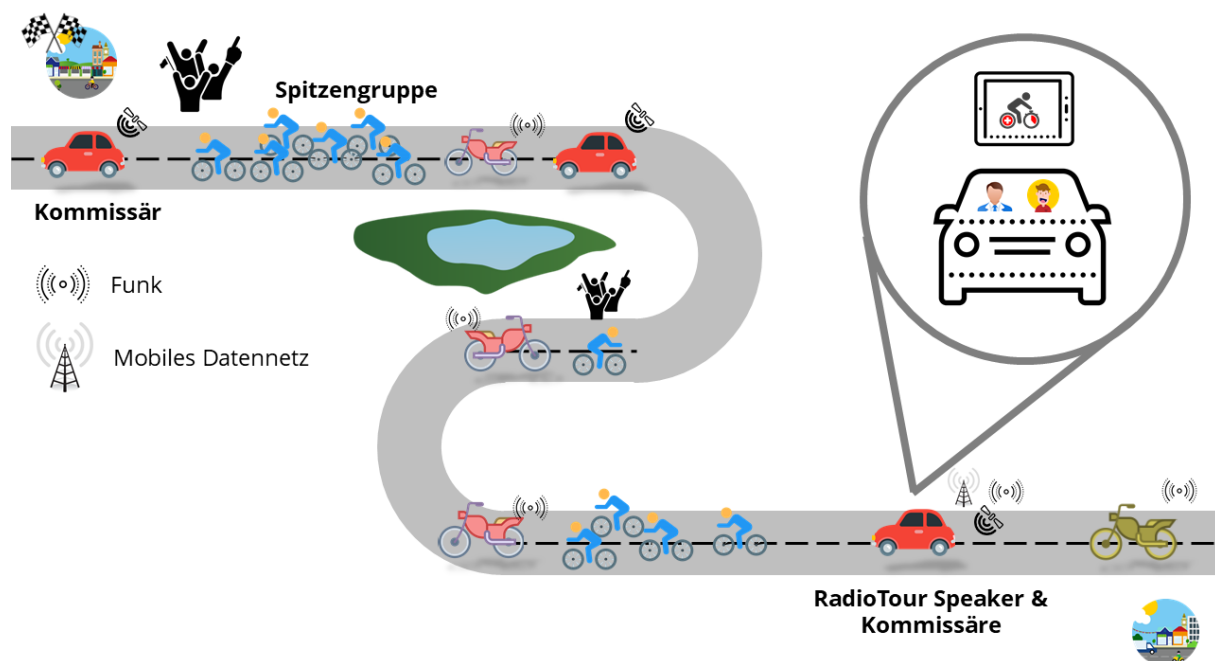


Abbildung 3: Ausgangslage

1.1.1 Stand der Technik

RadioTour Anwendung

In der Studienarbeit «RadioTour Anwendung», welche im Herbstsemester 2017 durchgeführt wurde, sind die Grundsteine in einem Prototyp der Applikation für den RadioTour Speaker gelegt worden. Mit diesem Prototyp ist es ihm möglich, die Rennsituation in digitalen Daten abzubilden.

The screenshot shows the 'Rennen' (Race) screen of the RadioTour application. The top status bar displays race details: 00:16 Spitzze - Feld, 0.0 m Höhe, 0 km/h Geschwindigkeit, 03:01:48 Rennzeit, and 00:04 Spitzze - Radio Tour, 3 Etappe, 0 km von 182 km Rennkilometer. The left sidebar shows a grid of rider numbers from 1 to 207. The main area displays rider information and groupings:

Rennnummer	Name	Team	Spitzenzeit	Gruppe	Fahrer
92	CIMOLAI Davide	GFC	64	SPITZENGRUPPE	2 Fahrer
102	GREIPEL André	LTS	71		
+					
2	GONÇALVES José	TKA	1	00:01	1 Fahrer
				00:00	
+					
12	GERRANS Simon	BMC	147	02:01	4 Fahrer
32	GESCHKE Simon	SUN	22	00:00	
42	VANDENBERGH Stijn	ALM	29		
53	GATTO Oscar	AST	37		
+					
				01:02	140 Fahrer
				02:00	
+					
				FELD	

Abbildung 4: RadioTour Anwendung, Rennen

The screenshot shows the 'Klassemente' (Classification) screen of the RadioTour application. The table displays rider rankings with the following columns: Nr., Name, Team, Land, Off. Zeit, Off. Rückstand, Virt. Rückstand, Punkte, P. Berg, DILG, Preisgeld, and TILG.

Nr.	Name	Team	Land	Off. Zeit	Off. Rückstand	Virt. Rückstand	Punkte	P. Berg	DILG	Preisgeld	TILG
4	HOLLENSTEIN Reto	TKA	SUI	00:00:00	00:00:00 (4)	00:00:59 (4)	0 (4)	3 (1)	0.0 km (4)	150 (1)	0 (6)
1	SPIIAK Simon	TKA	SLO	00:00:00	00:00:00 (1)	00:00:59 (2)	0 (1)	0 (2)	0.0 km (1)	0 (2)	0 (4)
2	GONÇALVES José	TKA	POR	00:00:00	00:00:00 (2)	00:00:00 (1)	0 (2)	0 (3)	0.0 km (2)	0 (3)	119 (1)
3	HAAS Nathan	TKA	AUS	00:00:00	00:00:00 (3)	00:00:59 (3)	0 (3)	0 (4)	0.0 km (3)	0 (4)	0 (5)
5	LAMMERTINK Maurits	TKA	NED	00:00:00	00:00:00 (5)	00:00:59 (5)	0 (5)	0 (5)	0.0 km (5)	0 (5)	0 (7)
6	SMIT Willem Jakobus	TKA	RSA	00:00:00	00:00:00 (6)	00:00:59 (6)	0 (6)	0 (6)	0.0 km (6)	0 (6)	0 (8)
7	CRAS Steff	TKA	BEL	00:00:00	00:00:00 (7)	00:00:59 (7)	0 (7)	0 (7)	0.0 km (7)	0 (7)	0 (9)
11	PORTE Richie	BMC	AUS	00:00:00	00:00:00 (8)	00:00:59 (8)	0 (8)	0 (8)	0.0 km (8)	0 (8)	0 (10)
12	GERRANS Simon	BMC	AUS	00:00:00	00:00:00 (9)	00:02:59 (147)	0 (9)	0 (9)	0.0 km (9)	0 (9)	0 (11)
13	KUNG Stefan	BMC	SUI	00:00:00	00:00:00 (10)	00:00:59 (9)	0 (10)	0 (10)	0.0 km (10)	0 (10)	0 (12)
14	DE MARCHI Alessandro	BMC	ITA	00:00:00	00:00:00 (11)	00:00:59 (10)	0 (11)	0 (11)	0.0 km (11)	0 (11)	0 (13)
15	SCHÄR Michael	BMC	SUI	00:00:00	00:00:00 (12)	00:00:59 (11)	0 (12)	0 (12)	0.0 km (12)	0 (12)	0 (14)
16	VAN AVERMAET Greg	BMC	BEL	00:00:00	00:00:00 (13)	00:00:59 (12)	0 (13)	0 (13)	0.0 km (13)	0 (13)	0 (15)
17	VAN GARDEREN Tejay	BMC	USA	00:00:00	00:00:00 (14)	00:00:59 (13)	0 (14)	0 (14)	0.0 km (14)	0 (14)	0 (16)
21	ALBASINI Michael	MTS	SUI	00:00:00	00:00:00 (15)	00:00:59 (14)	0 (15)	0 (15)	0.0 km (15)	0 (15)	0 (17)
22	MEYER Cameron	MTS	AUS	00:00:00	00:00:00 (16)	00:00:59 (15)	0 (16)	0 (16)	0.0 km (16)	0 (16)	0 (18)
23	JUUL-JENSEN Christopher	MTS	DEN	00:00:00	00:00:00 (17)	00:00:59 (16)	0 (17)	0 (17)	0.0 km (17)	0 (17)	0 (19)
24	BEWLEY Sam	MTS	NZL	00:00:00	00:00:00 (18)	00:00:59 (17)	0 (18)	0 (18)	0.0 km (18)	0 (18)	0 (20)
25	HAIG Jack	MTS	AUS	00:00:00	00:00:00 (19)	00:00:59 (18)	0 (19)	0 (19)	0.0 km (19)	0 (19)	0 (21)
26	HEPBURN Michael	MTS	AUS	00:00:00	00:00:00 (20)	00:00:59 (19)	0 (20)	0 (20)	0.0 km (20)	0 (20)	0 (22)
27	KREUZIGER Roman	MTS	CZE	00:00:00	00:00:00 (21)	00:00:59 (20)	0 (21)	0 (21)	0.0 km (21)	0 (21)	0 (23)
31	KELDERMAN Wilco	SUN	NED	00:00:00	00:00:00 (22)	00:00:59 (21)	0 (22)	0 (22)	0.0 km (22)	0 (22)	0 (24)
32	GESCHKE Simon	SUN	GER	00:00:00	00:00:00 (23)	00:00:59 (22)	0 (23)	0 (23)	0.0 km (23)	0 (23)	0 (25)
33	OOMEN Sam	SUN	NED	00:00:00	00:00:00 (24)	00:00:59 (23)	0 (24)	0 (24)	0.0 km (24)	0 (24)	0 (26)

Abbildung 5: RadioTour Anwendung, Klassemente

1.1.2 Umfeld

Die neuen Systeme stellen einen kleinen Teil des bestehenden TourLive-Systems rund um Radrennen dar, welches durch cnlab betreut und zur Verfügung gestellt wird. Im Folgenden werden die einzelnen Umsysteme detailliert erläutert.

cnlab API

Die cnlab API ist eine bereits bestehende Backend-Lösung, welche Daten für Radrennen liefert. Sie verwaltet zum einen statische Daten wie Wertungen, zum anderen dynamische Daten wie die aktuellen GPS Positionen der Renngruppen. Die Applikation ist auf Basis von Java implementiert und wird über einen Tomcat-Server betrieben. Die aktuelle Implementation der RadioTour-Anwendung erhält ihre Daten von der cnlab API.

TourLive Wo-Anwendung

Die TourLive Wo-Anwendung ist aus einer vorangegangenen Arbeit entstanden. Sie bildet die aktuellen Positionen aller Motorräder auf einer Kartenansicht dar. Zudem werden alle GPS Detailinformationen wahlweise eingeblendet. Auch die aktuelle Position auf dem Höhenprofil ist auf dieser Anwendung ersichtlich. Diverse Einstellungen ermöglichen verschiedene Aktualisierungsintervalle und es ist auch möglich einem spezifischen Fahrzeug auf der Karte zu folgen. Es werden aber keine Informationen zu Renngruppe oder dergleichen dargestellt.

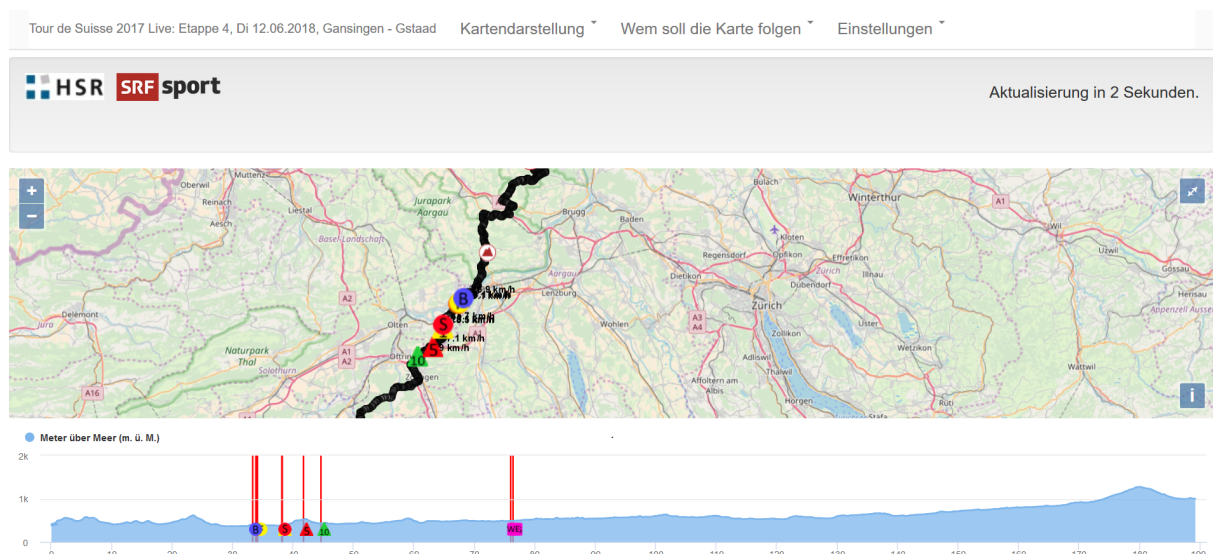


Abbildung 6: TourLive Wo-Anwendung

1.2 Vision

Die Zuschauer sollen sich an den Radrennen erfreuen können und mit unserer Dienstleistung weitere Detailinformationen zu diesen in Erfahrung bringen können. Dabei soll es keine Rolle spielen ob dies am Strassenrand oder zu Hause in der Wohnung passiert. Um das Erlebnis mit unserer Webseite so angenehm wie möglich zu machen soll das Design ansprechend und benutzerfreundlich sein. Des Weiteren sind Ladeverzögerungen so gering wie möglich zu halten.

1.3 Aufbau der Arbeit

Die Dokumentation zur Arbeit ist in verschiedenen Teile gegliedert. In den Kapiteln 2 und 3 werden Anforderungen definiert und dazu passende Lösungsvorschläge erarbeitet. Ebenfalls werden die eingesetzten Technologien definiert und festgelegt. Um dem Benutzer der Applikation ein bestmögliches Nutzererlebnis zu schaffen wird im Kapitel 4 spezifisch auf das Design des User Interfaces eingegangen.

Damit das Konzept der Arbeit richtig umgesetzt wird ist die ganze Software Architektur im Kapitel 5 beschrieben. Im Folgekapitel 6 wird auf die Realisierung dieser Architektur eingegangen und spezifische Code Auszüge genauer analysiert und beschrieben.

In den Kapiteln 7 und 8 wird die Software auf Herz und Nieren geprüft um ein anschließendes Fazit der Ergebnisse zu ziehen und ein Endresultat zu bestimmen. Im Anhang befinden sich weitere Kapitel zum Thema Projektmanagement, also wie das Projekt geplant und durchgeführt wurde.

2 Anforderungsspezifikation

In diesem Kapitel «Anforderungsspezifikation» werden die Rahmenbedingungen, Ansprüche und Anforderungen festgehalten. Sie stecken den Rahmen der Bachelorarbeit weitestgehend ab. Zur Anforderungsspezifikation gehören sowohl die funktionalen als auch die nicht funktionalen Anforderungen.

2.1 Funktionale Anforderungen (Use Cases)

Die funktionalen Anforderungen bilden das Fundament jeder Anwendung. Anhand deren entstehen UI-Entwürfe, Architekturen, Prototypen und schlussendlich die Anwendungen selbst. Die funktionalen Anforderungen ergaben sich aus einer Analyse der bestehenden Anwendungen sowie der Konkurrenz und wurden gemeinsam mit dem Betreuer abgesprochen.

Die Use Cases sind im Casual-Format beschreiben. Auf eine detailliertere Beschreibung wurde aufgrund der nicht allzu komplexen Use Cases verzichtet.

Die Aufstellung umfasst nur die im Rahmen der Bachelorarbeit realisierten Use Cases. Die gesamten Use Cases der RadioTour-Anwendung (Android-App) können der Studienarbeit «RadioTour Anwendung» vom Herbstsemester 2017 entnommen werden. Eine detailliertere Beschreibung ist dem abgegebenen USB-Stick zu entnehmen.

2.1.1 Erweiterung an der RadioTour Anwendung (Android App)

Legende Priorität: *** hoch, ** mittel, * niedrig

UC1 | Import der aktuellen Daten (*)**

Die aktuellen Daten zur Etappe werden nun von der TourLive API importiert. Dadurch wird der UC6 der TourLive API angestossen

UC2 | Renngruppe ändert sich (*)**

Eine Renngruppe ändert sich und ihre Inhalte (Fahrer mit Zeiten) werden zur TourLive API in Form einer JSON übermittelt. Dadurch wird der UC7 angestossen.

UC3 | Fahrerstatus ändert sich (*)**

Bei einem betroffenen Fahrer wird der Status gesetzt und der aktuelle Zustand wird dann per JSON übermittelt. Zustandswechsel sind bei Arzt, Sturz, Defekt, DNC oder Aufgabe möglich. Dadurch wird der UC8 angestossen.

UC4 | Eine Wertung wird einem Fahrer zugewiesen (*)**

Bei der Zuweisung einer Wertung werden alle Rangierungen und Punkte der Fahrer an die TourLive API mit JSON übermittelt. Dadurch wird der UC9 angestossen.

UC5 | Ein virtuelles Maillot wird neu vergeben (*)**

Die aktuellen Träger des virtuellen Maillots werden an die TourLive API übermittelt. Dadurch wird der UC10 angestossen.

2.1.2 TourLive API (Backend)

UC6 | GET – Bereitstellung der Renninformationen für RadioTour Anwendung (*)**

Die Daten (Etappen, Fahrer, Renngruppen, Wertungen, Maillots) der aktuellen Etappe werden der RadioTour Anwendung zur Verfügung gestellt und können importiert werden.

UC7 | UDPATE – Renngruppe ändert sich (*)**

Die betroffene Renngruppe und ihre Inhalte (Fahrer mit Zeiten) werden empfangen und in der Datenbank der API persistiert.

UC8 | UPDATE – Fahrerstatus ändert sich (*)**

Der aktualisierte Fahrer sowie dessen Status werden empfangen und in der Datenbank der API persistiert.

UC9 | UPDATE – Eine Wertung wird einem Fahrer zugewiesen (*)**

Alle Rangierungen und Punkte der Fahrer werden empfangen und in der Datenbank der API persistiert.

UC10 | UPDATE – Ein virtuelles Maillot wird neu vergeben (*)**

Die aktuellen Träger des virtuellen Maillots werden empfangen und in der Datenbank der API persistiert.

UC11 | Importieren der statischen Daten (*)**

Zum Start jeder Tour (jedes Rennens) werden die aktuellen Informationen zu Fahrern, Zeiten und der Marschtabelle von der cnlab API importiert und in der Datenbank persistiert. Der Import wird vom Admin Interface des Webservers gestartet.

UC12 | Importieren der dynamischen Daten (*)**

Zum Start jeder Etappe werden die aktuellen Informationen zu Fahrern, Zeiten und Wertungen von Matsport importiert und in der Datenbank persistiert. Der Import wird vom Admin Interface des Webservers gestartet.

UC13 | Aktuelle Daten zur Verfügung stellen (*)**

Während des Radrennens werden von den Webservern Aktualisierungsabfragen gestartet. Dabei werden alle Informationen Daten (Etappen, Fahrer, Renngruppen, Wertungen, Maillots, Karten, Höhenprofil, aktuelle Positionen) als JSON Datei an die Webserver übermittelt, um den Rennzustand auf der Webanwendung zu visualisieren. Die Routen werden in einem laufenden Verfahren für die jeweiligen Use Cases definiert.

UC14 | Input aus der Interaktionsanwendung (*, optional)

Während eines Rennens können die Zuschauer Interaktionen über die Webanwendung auslösen. Diese Inputs sollten in der API persistiert werden.

2.1.3 Webanwendungen

UC15 | Abfrage der aktuellen Daten (*)**

Es wird periodisch (alle 10 Sekunden) eine Abfrage an die TourLive API gestartet, um die aktuellen JSON Daten der Webserver mit den Renninformationen zu aktualisieren.

UC16 | Admin – Import der statischen Daten (*)**

Zum Start jeder Tour (Rennen) werden die Daten der Tour importiert. Dieser Use Case startet den Use Case 11 der TourLive API.

UC17 | Admin – Import der dynamischen Daten (*)**

Zum Start jeder Etappe werden die aktuellen Daten (vom offiziellen Zeitgeber) importiert. Mit diesem Use Case springt der Use Case UC12 an.

UC18 | Darstellung der Etappen (*, optional)**

Im Home-Screen werden alle verfügbaren Etappen angezeigt. Durch ein «Swipe» ist es möglich zwischen den Etappen zu wählen. In der ganzen Anwendung werden immer Informationen zur aktuelle gewählten Etappe dargestellt.

UC19 | Darstellung der aktuellen Rennsituation (Gruppe, Zeitabstände) (*)**

Im Home-Screen werden die aktuellen Gruppen auf einem Zeitstrahl dargestellt. Zusätzlich werden der aktuelle Kilometer und die Abstandszeit angezeigt. Durch einen Klick auf eine Gruppe werden alle darin enthaltenen Fahrer (Start Nr., Trikot, Flagge, Team, Name, virtueller Rang) aufgelistet. Zudem werden die globalen Informationen (Gruppenname, Zeitabstand, Anzahl Fahrer in Gruppe) dargestellt.

UC20 | Darstellung der Klassemente (*)**

Im Screen der Klassemente ist es möglich aus den verschiedenen Kategorien (Berg, Sprint, Swiss, ...) auszuwählen. Standardmässig ist die Kategorie virtuelle Zeit ausgewählt. Im Screen befindet sich eine Übersicht als Liste mit den Informationen zum jeweiligen Klassement, das heisst alle Fahrer mit Rang, Start Nr., Trikot, Flagge, Team und Name.

UC21 | Anzeige der Position der Rennkolone auf Karte (*)**

Es wird im Screen eine Karte angezeigt bei der die Gruppenposition auf der Strecke ersichtlich sind. Zudem wird auf der Karte der Standort des Nutzers angezeigt. Es gibt ein Button mit welchem zwischen der Kartenansicht und dem Höhenprofil gewechselt werden kann.

UC22 | Anzeige der Position der Rennkolone auf dem Höhenprofil (*)**

Es wird im Screen ein Höhenprofil angezeigt bei der die Gruppenposition auf der Strecke ersichtlich sind. Es werden Informationen zu ausgewählten Streckenpunkten detailliert (Ortsname, Rennkilometer, Höhe, geschätzte Ankunftszeit) dargestellt. Es gibt ein Button mit welchem zwischen der Kartenansicht und dem Streckenprofil gewechselt werden kann.

UC23 | Anzeige aller Wertungen (*)**

Es werden alle Wertungen der ausgewählten Etappe aufgelistet mit Informationen zu Sponsor, Punkten und bereits zugewiesenen Fahrern.

UC24 | Suche von Informationen zu einem bestimmten Fahrer (*)**

Im Suchfeld wird der Name des gewünschten Fahrers eingegeben. Dabei soll eine Auto-Vervollständigung-Auswahlmöglichkeit angezeigt werden. Wird der Fahrer gefunden und sein Name angeklickt wird ein Screen mit Informationen zum Fahrer aufbereitet. Dabei werden Angaben zum Fahrer (Start Nr., Trikot, Flagge, Team, Name, virtueller Rang) gemacht und dargestellt in welcher Gruppe er sich befindet. Zusätzlich sind seine bereits erreichten Maillots und Wertungen ersichtlich.

UC25 | Notifications darstellen (*)**

Im Falle eines Statuswechsels bei einem Fahrer (z.B. Sturz) wird eine Notification auf der Webseite eingeblendet.

UC26 | Interaktionsmöglichkeit für den Zuschauer (*, optional)

Während des Rennens können Interaktion in der Webanwendung von Zuschauer ausgelöst werden. Es gilt dem Zuschauer diese Möglichkeit zu bieten.

UC27 | Aktuelle Interaktionen anzeigen (*, optional)

Während des Rennens können Interaktion in der Webanwendung von Zuschauer ausgelöst werden. Um dem Zuschauer einen Überblick zu bieten werden die Interaktionsresultate in der Interaktionsanwendung dargestellt.

2.2 Use Case Diagramm

Die in diesem Kapitel aufgeführten Diagramme ergeben einen Überblick über die gesamten Funktionen des Systems sowie das Zusammenspiel deren.

Farbe	Bedeutung
	Optionale Features
	EBP nahe Use Cases
	CRUD nahe Use Cases
	Aktoren

Tabelle 1: Legende zu den Use Case Diagrammen

2.2.1 Aktoren

Nebst den zu realisierenden Systemen dienen folgende Systeme ebenfalls als Aktoren und stellen Daten oder Funktionen zur Verfügung.

Farbe	Bedeutung
cnlab API (Webservice)	Das cnlab API liefert die gesamten statischen Daten zur aktuellen Tour wie Fahrer, uvm., welche im Vorfeld in die TourLive API importiert werden.
Matsport	Matsport stellt nach jeder Etappe die aktuell gültigen Klasselemente und Zeiten zur Verfügung, welche im Anschluss in die API importiert werden.

Tabelle 2: Weitere Aktoren in den Use Case Diagrammen

2.2.2 Use Case Diagramm zu den Erweiterungen der RadioTour Anwendung (App)

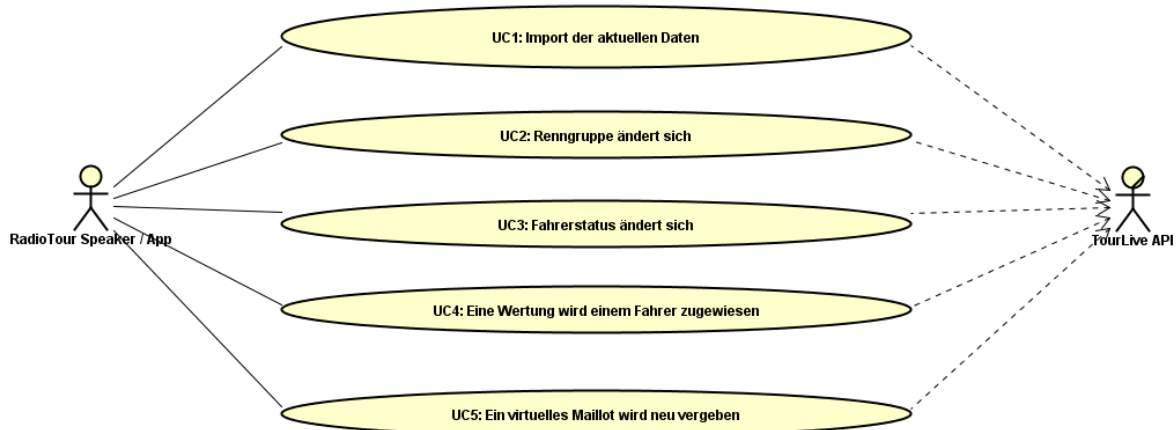


Abbildung 7: Use Case Diagramm zu den Erweiterungen der RadioTour Anwendung (App)

2.2.3 Use Case Diagramm der Webanwendungen

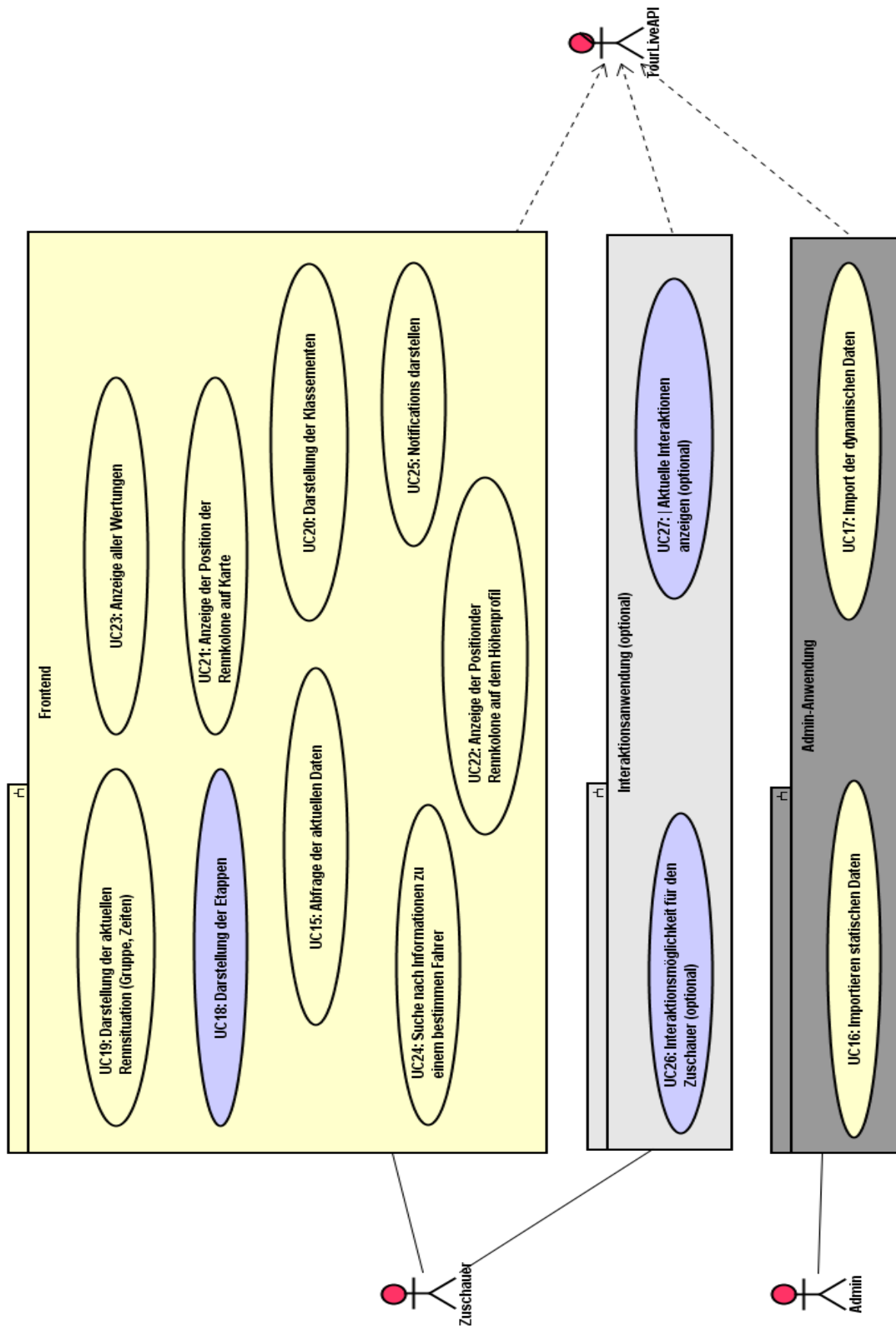


Abbildung 8: Use Case Diagramm der Webanwendungen

2.2.4 Use Case Diagramm «TourLive API» (Backend)

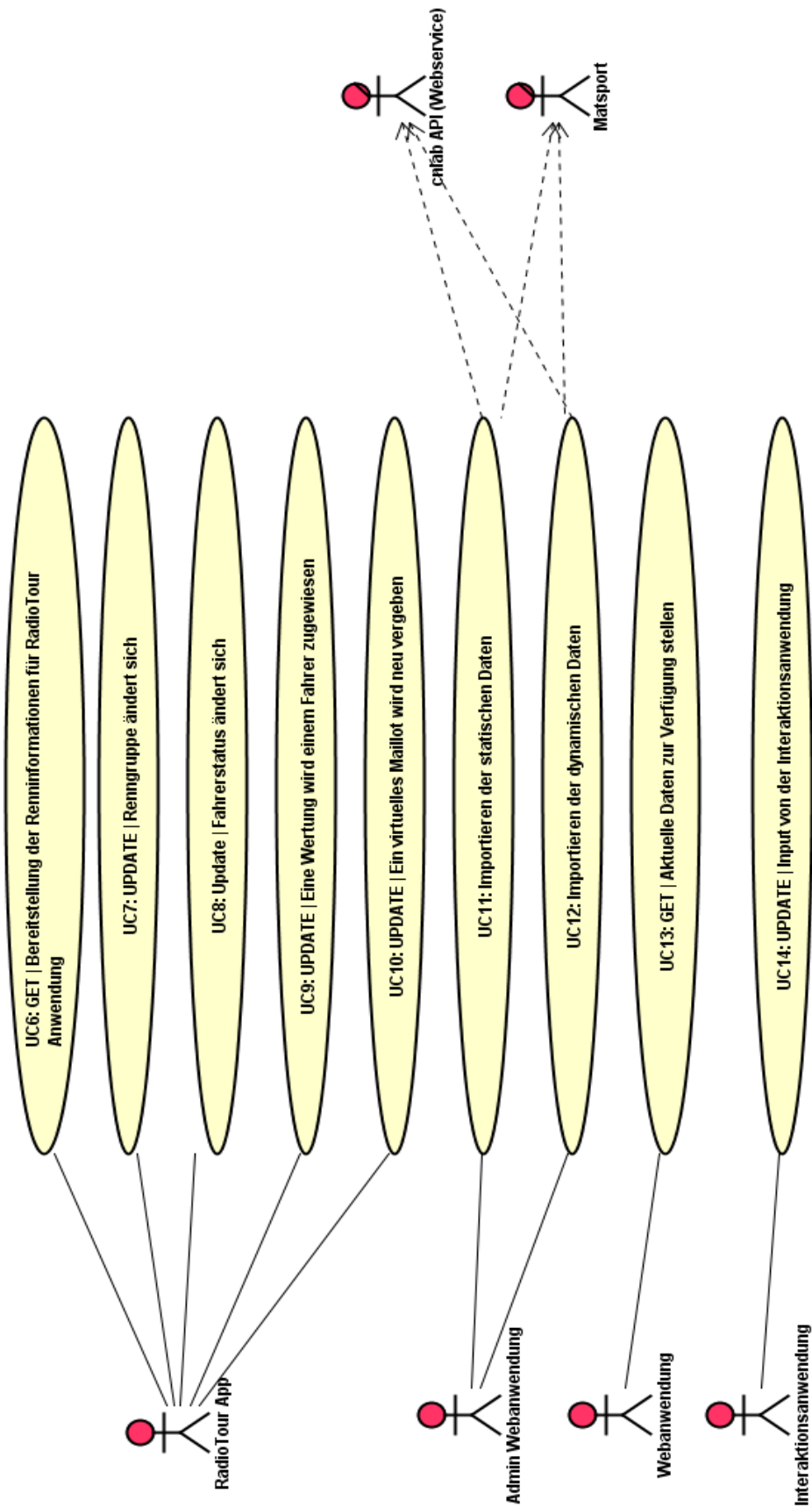


Abbildung 9: Use Case Diagramm «TourLive API» (Backend)

2.3 Nicht funktionale Anforderungen

Mit diesem Kapitel wird der Teil der nicht funktionalen Anforderungen (NFA) innerhalb der Anforderungsspezifikation abgedeckt. NFA's sind besonders wichtig, da sie einen erheblichen Einfluss auf die Architektur sowie auf die Art und Weise der Programmierung haben. Die nachstehend erwähnten NFA's decken nur das Frontend (Webanwendung) sowie das Backend (TourLive API) ab. Die nichtfunktionalen Anforderungen der RadioTour Anwendung (Android APP) können der Studienarbeit entnommen werden.

2.3.1 Mengengerüst

Das System TourLive soll in der Lage sein mehrere hundert Benutzer (mind. 400) gleichzeitig zu bedienen zu können. Die Auslastung der Ressourcen soll zu keiner Zeit über 100 Prozent betragen.

2.3.2 Zeitverhalten

Bei entsprechenden vorhandener Internetverbindung (3G oder WLAN) soll die Zeit für das Aufrufen der Anwendung weniger als 4 Sekunden dauern. Jeder weitere Task innerhalb der Webanwendung sollte innerhalb von Sub-Second Antwortzeit realisiert werden.

2.3.3 Sicherheit

Das System tauscht die Daten über eine gesicherte Verbindung zwischen den verschiedenen Anwendungsteilen aus. Dazu soll HTTPS eingesetzt werden. Dies schenkt dem Benutzer Vertrauen bei der Verwendung der Anwendung und unterbindet eine Übertragung der Daten im Klartext.

2.3.4 Ordnungsmässigkeit

TourLive unterliegt dem aktuellen geltenden Schweizer Recht. Die Einhaltung dieser Gesetze ist Voraussetzung um die Anwendung einer breiten Masse zur Verfügung stellen zu können.

2.3.5 Richtigkeit

Alle Angaben (Zeiten oder Punkte) sind in der genauesten Schreibweise (also keinen Rundungen) darzustellen. Jegliche Eingaben in das System (API oder Web) werden vor der Weitergabe geprüft und unterbinden eine Eingabe von falschen Daten.

2.3.6 Fehlertoleranz

Sämtliche Eingaben in das System (über API oder Web) werden vorgängig validiert und auf Richtigkeit geprüft. Bei Fehleingaben werden verständliche Fehler produziert, so dass der Benutzer entsprechend darauf reagieren kann. Die Validierung reduziert die Menge der fehlerhaften Daten im System. Bei Fehlern wird die gesamte aktuelle Aktion rückgängig gemacht (Rollback)

2.4 Stakeholders

Die Interessen der Stakeholder prägen die Ausarbeitung einer Anwendung im Allgemeinen sehr. Es gilt besonders die Stakeholder mit grossem Interesse und grossem Einfluss zu bedienen.



RadioTour Speaker, der Hauptnutzer der Android Anwendung



Zuschauer, die Hauptnutzer der Webanwendung



Administratoren, die Verwalter der Webanwendung



TV (SRF), Ausstrahler des Radrennens



Staat (Datenschutz), Sicherstellung des Datenschutzes

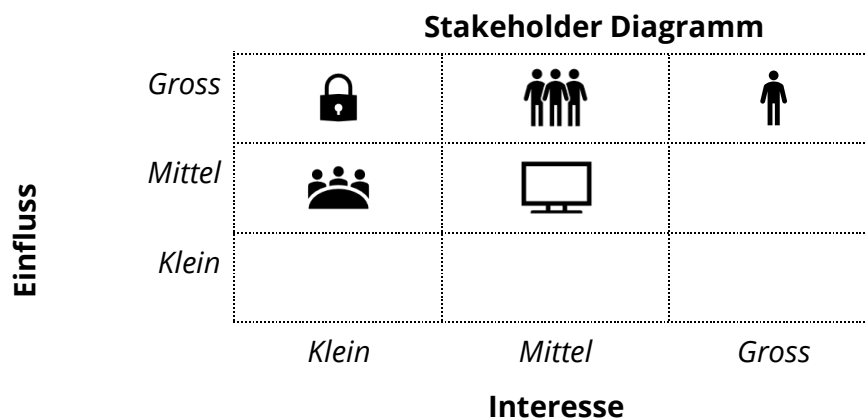


Abbildung 10: Stakeholder Diagramm

In unserem Fall muss daher der Fokus auf dem RadioTour Speaker, welcher die Android Anwendung nutzt, sowie auf den Zuschauern, welche die Webanwendung nutzen, liegen.

3 Evaluationen

In diesem Kapitel werden die relevanten Technologien und Frameworks, welche das System der Bachelorarbeit betreffen, evaluiert. Für jede einzelne Evaluation wurden entsprechende Kriterien erarbeitet. Die Auswahl der Komponente geschieht jeweils anhand der im Vorfeld definierten Kriterien.

3.1 Frontend Framework

Die Entscheidung für ein Framework ist von grosser Bedeutung. In der heutigen Zeit existieren mehrere Frameworks um Web-Applikationen zu schreiben. Bei der zu verwendenden Programmiersprache haben wir uns auf JavaScript eingeschränkt um unsere Möglichkeiten einzugrenzen. Zudem verfügen wir in diesem Bereich über fundierte Kenntnisse. Dies ist bei Java Web Frameworks nicht der Fall.

3.1.1 Kriterien

- Projekt nur so gross wie nötig halten
- Dokumentation und Ressourcen
- Bekanntheitsgrad (→ Support)
- Einfacher Einstieg
- Vorhandene Kenntnisse
- Erfahrungen von Kommilitonen

3.1.2 React [2]



React ist eine JavaScript Bibliothek für das Erstellen von Benutzeroberfläche und damit auch Web-Anwendungen. React entstand im Hause Facebook und wird für die Facebook-Anwendung auch aktiv gebraucht. Auch an-

dere namhafte Firmen wie Netflix, Airbnb oder Imgur setzen React ein. React selbst versteht sich nicht als komplettes Framework, sondern ist durch Module erweiterbar. Dazu zählt beispielsweise der React Router (die Routing Komponente von React). Durch diesen Umstand wird eine Anwendung so schlank wie möglich gehalten und nur auch das wirklich Nötige wird importiert.

Eine weitere Besonderheit von React liegt darin, dass es mit sogenannten Komponenten aufgebaut ist. In einer solchen Komponente wird die Logik abgekapselt und macht es einfacher komplexe UI bzw. Anwendungen zu erstellen. Diese Komponenten enthalten neben JavaScript-Code auch JSX HTML Code. Also alles an einem Ort. Der Code ist dadurch etwas übersichtlicher und von Hand durchgeführte DOM-Manipulationen werden verhindert.

Informationen von Variablen werden in einem State (zu Deutsch Zustand) gespeichert. Jede Komponente weist einen solchen State auf. Der State ist pro Komponente gültig. Sobald sich dieser State ändert, wird die dazugehörige Methode **render()** aufgerufen und das HTML und damit das UI aktualisiert.

Des Weiteren unterstützt React das sogenannte «Virtual Rendering». Das heisst konkret, dass nur wirklich die Elemente im HTML manipuliert werden, welche auch aktualisiert worden sind. Mittels Differenz zwischen dem echten (aktuell geladenen DOM) und dem virtuellen DOM wird eruiert, welche Elemente neu gerendert werden müssen.

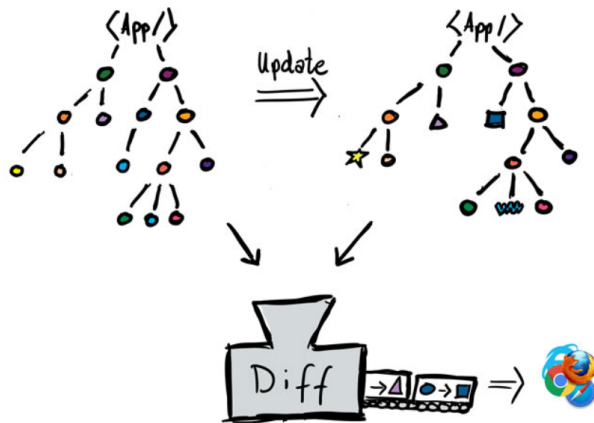


Abbildung 11: Virtuelles Rendering des DOMs [3]

React bietet einen grossen Umfang an kompatiblen Modulen an, welche über npm bzw. yarn nachgeladen werden können. Einer dieser Module ist Redux. Redux stellt einen globalen State zur Verfügung (über die Komponenten hinweg) und eignet sich daher ideal für das Zwischenspeichern von Daten. Jede Komponente bezieht die Daten anschliessend aus dem Redux Store.

Neben React hat Facebook zusätzlich das Projekt «React Native» gestartet, welches die Entwicklung von plattformunabhängigen Handy-Applikationen erlaubt.

Vorteile	Nachteile
<ul style="list-style-type: none"> - Facebook als Owner - Hohe Flexibilität - Manuelle Zusatzmodulverwaltung (es wird nur das importiert was wirklich gebraucht wird) - Grosses Ökosystem (viele Informationen im Netz vorhanden) - Geeignet für kleine Teams - Basis des Programmierkonzeptes im Web 	<ul style="list-style-type: none"> - Kein komplettes Framework, sondern nur Baukasten - Clean Code nicht unbedingt gegeben (JSX)

Tabelle 3: Vor- und Nachteile von React

3.1.3 Vue.JS [4]



Vue.js ist ein clientseitiges JavaScript-Framework für das Web. Es eignet sich zum Erstellen von Single-Page-Applikationen. Multi-Page-Anwendungen sind ebenfalls möglich. Vue.js basiert grund-

sätzlich auf denselben Prinzipien wie React. Es verwendet ebenso einen virtuellen DOM um nur die notwendigen Elemente im Frontend neu zu laden. Die Rechenzeit für die Darstellung ist beinahe identisch. Im Gegensatz zu React steht hinter Vue.js kein namhafter Entwickler. Es wird alleine durch mehrere Sponsoren getragen. Grössenmässig ist Vue.js ein Mittelweg zwischen React und Angular.

Ein wichtiger Hauptunterschied besteht darin, dass es in Vue.js Templates gibt, welche für den Aufbau der Webseite zuständig sind. Hingegen wird bei React nur JSX (deklarative XML Syntax die mit JavaScript arbeitet) verwendet.

Ebenso wie React bietet Vue.js eine dynamische Library und ist somit eigentlich kein Framework.

Vorteile	Nachteile
<ul style="list-style-type: none">- Clean Code- Einfachste Lernkurve- Lightweight Framework- Separation of Concerns- Geeignet für kleine Teams	<ul style="list-style-type: none">- Kein grosser Name der dahinter steht- Limitiertes Ökosystem

Tabelle 4: Vor- und Nachteile von Vue.js

3.1.4 Angular [5]



Angular ist ein komplettes clientseitiges JavaScript Framework für dynamische Webapplikationen. Grundsätzlich ist es für die Entwicklung von Single-Page-Apps gedacht. Entwickelt wurde es von Google und wird dort auch aktiv

für ihre Anwendungen eingesetzt.

Es unterstützt die bidirektionale Datenbindung und ist auf gute Testbarkeit ausgelegt. Die Architektur ist dabei in verschiedene Komponenten aufgeteilt. Es trennt so die Zuständigkeit zwischen View und Logik.

Angular ist vor allem für die Entwicklung von grossen Projekten gedacht. Durch das vielfältige Tooling wird der Entwickler dabei unterstützt einen sauberen und wartbaren Code zu programmieren. Es wird unter anderem bei Tesla und Microsoft eingesetzt.

Vorteile	Nachteile
<ul style="list-style-type: none">- Google als Owner- Objekt orientierte Programmierung möglich- Strukturiert- Für grosse Projekte geeignet	<ul style="list-style-type: none">- Schwierige Einarbeitung- Grosser Overhead für kleine Projekte

Tabelle 5: Vor- und Nachteile von Angular

3.1.5 Entscheidung

Unsere Entscheidung fiel auf React. Eine Implementation mit Vue.js oder Angular wäre ebenfalls denkbar gewesen, da alle unsere Anforderungen erfüllt haben. Auch andere Programmiersprachen wie Java oder C# wären möglich gewesen, wurden aber nicht berücksichtigt.

Ausschlaggebend für die Auswahl von React waren mehrere Gründe. Mit React ist es möglich unser Projekt nur so gross wie nötig zu machen und entsprechende Module nach und nach zu importieren. Ein Punkt der gegen Angular spricht. Zudem steckt hinter React ein grosser und bekannter Name und eine Unterstützung ist über mehrere Jahre eher sicher. Etwas das gegen Vue.js spricht. Nicht zuletzt hat React eine breite Community und ist mit vielfältigen Hilfestellungen im Netz ausgestattet.

React ist der Grundbaustein für die anderen beiden Sprachen. Unsere Kenntnisse sind bei allen Frameworks noch nicht vorhanden. Aus diesem Grund beginnen wir mit einer Library, welche einfach zu erlernen ist. Zudem ist es nie schlecht den Grundbaustein zuerst zu erlernen.

3.2 Deploymentlösung (Serverlösung)

Um eine optimale Lösung für die Infrastruktur der RadioTour/TourLive Anwendung zu finden, wurden verschiedene Deployment- bzw. Serverlösungsmodelle genauer untersucht. Zu den untersuchten Modellen zählen IaaS, Container und PaaS.

3.2.1 IaaS (Virtueller Server)

Ein reiner virtueller Server gehört unter die Kategorie «Infrastructure as a Service» der drei grundlegenden Service-Modelle für das Cloud Computing (nach der Definition von cloudcomputingpatterns.org). Im Falle von IaaS bekommt der Kunde Zugriff auf den virtuellen Server, also die Computing Ressource. Das primäre Ziel ist hier physische Ressourcen zu teilen. Alles, bis und mit dem Virtualisierungs-Layer, wird vom entsprechenden Provider zur Verfügung gestellt. Als Kunde bzw. Betreiber eines virtuellen Servers ist man demnach für folgende Layer selbst zuständig:

- **Betriebssystem** (Installation, Updates, Patches, Konfiguration)
- Allfällige **Middleware**
- **Runtime** (z.B. Java)
- **Daten** (Datenbank)
- **Applikation** (z.B. TourLive Zuschauer-Webanwendung)

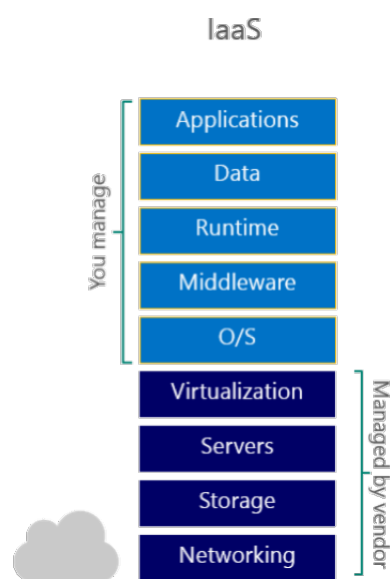


Abbildung 12: IaaS

Durch die grossen Zuständigkeiten muss ich als Betreiber auch eine entsprechend grosse Verantwortung tragen.

Als Nutzer bietet mir eine Lösung mit virtuellen Servern (IaaS) aber durchaus Vorteile. Mit IaaS kann das Betriebssystem sowie der gesamte Softwareteil selbst gewählt werden. Ein Vendor-Locking ist somit nicht vorhanden. Als weiteres Merkmal für diese Lösung stehen die geringen Kosten (Kosten für die Konfiguration werden ausser Acht gelassen). So kann ab wenigen Franken pro Monat eine virtuelle Maschine bei einem Schweizer IaaS-Provider gemietet werden.

Vor- und Nachteile

Die Tabelle 1Tabelle 6 fasst die Vor- und Nachteile einer IaaS-Lösung zusammen.

Vorteile	Nachteile
<ul style="list-style-type: none">- Geringe Kosten (ohne den Konfigurationsaufwand)- Kein Vendor-Locking bei Betriebssystem sowie der Software auf dem Server	<ul style="list-style-type: none">- Skalierbarkeit nur Maschinenweise möglich (kompletter Server muss konfiguriert werden)- Verantwortlich für einen grossen Teil des Stacks (der Layer)- Maintaining und Deployment der einzelnen Applikationen von Hand und ggf. sehr kompliziert<ul style="list-style-type: none">o Scripts können dafür geschrieben werden, sind allerdings statisch.

Tabelle 6: Vor- und Nachteile von IaaS

Provider/Produkte von virtuellen Maschinen (IaaS)

- Cloud Server¹ (Scaleway)
- Cloud Server oder Root Server² (Metanet)
- VPS³ (OVH)
- EC2⁴ (Amazon)
- Virtual Server⁵ (Microsoft Azure)
- Compute Engine⁶ (Google Cloud)

3.2.2 Container am Beispiel von Docker

Container, oder um ein Produkt zu nennen, Docker Container sind heutzutage State of the Art. Ein Container ist ein einfaches und selbstständiges Paket einer Software welches alles beinhaltet um eine Software auszuführen, d.h. den Code, die Runtime und allfällige Libraries und Einstellungen. Durch die Abkapselung läuft der Container überall gleich, egal auf welcher Umgebung (Windows, Linux, macOS).

Aufgrund der aktuellen Marktherrschaft von Docker beziehen wir uns bei dieser Lösung immer auf Docker.

Container sind eine Abstraktion auf dem App Layer und packen die Abhängigkeiten zusammen. Mehrere Container laufen auf derselben Maschine und teilen einen gemeinsamen Kernel. Dadurch sind die Container selbst nur wenige MB gross und starten dementsprechend auch schnell. Im Vergleich zu virtuellen Servern (IaaS) werden mehrere VMs

¹ <https://www.scaleway.com/>

² <https://www.metanet.ch/server>

³ https://www.ovh.de/virtual_server/

⁴ https://aws.amazon.com/de/ec2/?nc2=h_l3_c

⁵ <https://azure.microsoft.com/de-de/services/virtual-machines/>

⁶ <https://cloud.google.com/compute/?hl=de>

(Virtuelle Maschinerien) auf einem physischen Gerät laufen gelassen. Jede VM hat eine volle Kopie des OS und braucht je nach Betriebssystem mehrere zehn GB an Platz.

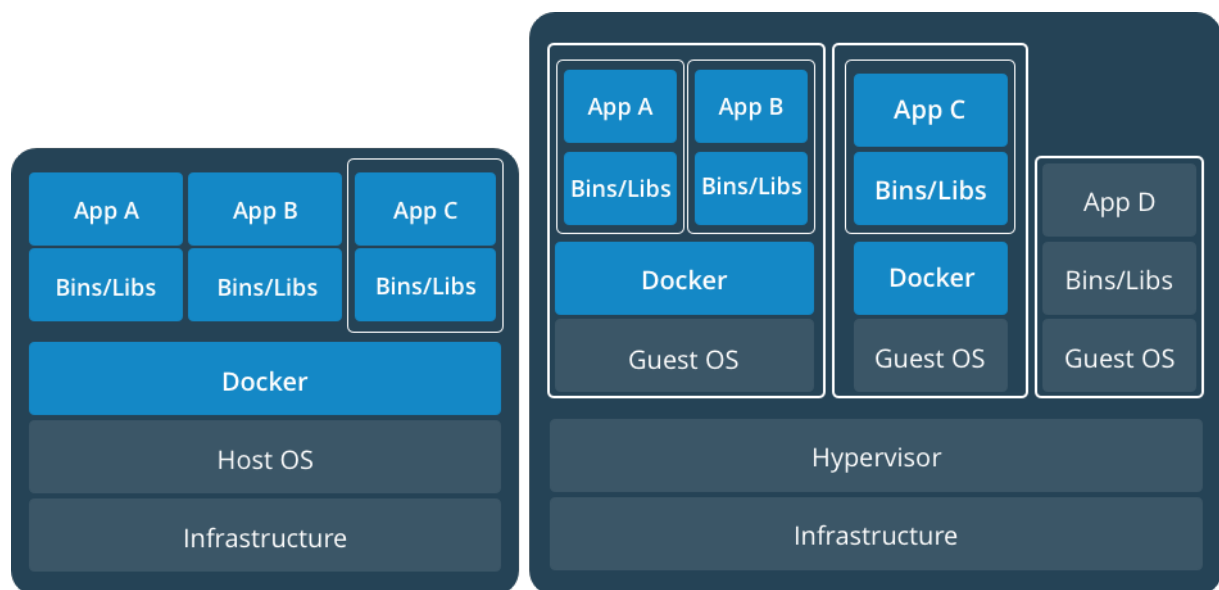
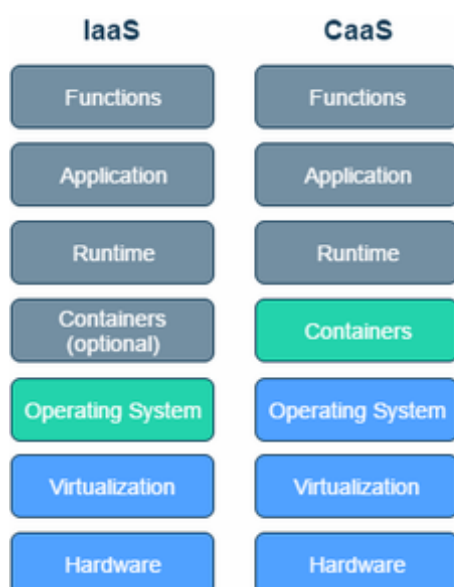


Abbildung 13: Unterschied Container und virtuelle Maschine [6]

Auf den ersten Blick sind die Vorteile nicht ganz klar. Man braucht ja trotzdem eine virtuelle Maschine (IaaS) um die Docker Engine zu installieren. Die Verantwortlichkeit für einen grossen Teil des Stacks bleibt also. Auf einer VM kann ich aber mehrere Container laufen lassen und brauche somit weniger, dafür grössere Instanzen. Durch Dockerfiles entfällt eine komplizierte Installation und Konfiguration durch Scripts. Zudem kann ich die Applikation einfach auf einer anderen virtuellen Maschine laufen lassen (Dockerfile beschreibt ja alles). Durch Docker Swarm hat man zudem die Möglichkeit ein Cluster über mehrere Maschinen hinweg zu bilden.



Es gibt auch Anbieter wie Microsoft Azure, welche uns die virtuellen Maschinen abnehmen und uns dafür lediglich Container zur Verfügung stellen. Diese Konstellation nennt man dann Container as a Service. Die Abbildung 14 zeigt uns diesen Umstand. Für unseren Vergleich lassen wir CaaS erstmal aussen vor und betrachten die Lösung 2 als reine Container auf Basis von Docker.

Abbildung 14: IaaS vs. CaaS [7]

Vor- und Nachteile

Die Tabelle 7 fasst die Vor- und Nachteile einer Container Lösung zusammen.

Vorteile	Nachteile
<ul style="list-style-type: none">- Open Source Lösung (Docker Engine gratis)- Software innerhalb des Containers kann ganz nach den eigenen Bedürfnissen ausgewählt werden- Automatisches Load Balancing über mehrere Instanzen mit Docker Swarm- Applikation läuft überall gleich- Container beliebig konfigurierbar- Rechte innerhalb Container können selbst bestimmt werden.- Produkte wie DB auch auf Docker möglich- Skalierung über Docker Command einfach möglich.	<ul style="list-style-type: none">- Maintaining von Docker nötig- Infrastruktur auf der Docker läuft ist nötig (ohne CaaS)- Verantwortlich für einen grossen Teil des Stacks.

Tabelle 7: Vor- und Nachteile von Container

3.2.3 Platform as a Service (z.B. Google App Engine)

Genau wie die Infrastructure as a Service umfasst PaaS die verschiedenen Infrastrukturkomponenten wie Server, Speicher, Virtualisierung und Netzwerk. Zusätzlich bietet Platform as a Service weitere Layers, dazu gehören OS, Middleware und Runtime, an. Der Benutzer, in diesem Fall der Entwickler, braucht sich daher nur noch um die Applikation sowie die Daten zu kümmern.

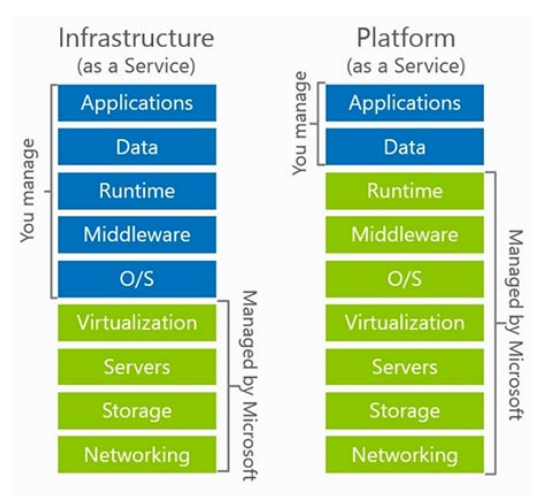


Abbildung 15: Vergleich IaaS und PaaS anhand von Microsoft [8]

Programmiermodelle

Programmierung für PaaS Anbieter entspricht etwa der Programmierung für Enterprise Anwendungen. So sind Asynchronität und Statelessness (Zustandlosigkeit) wichtige Eigenschaften. Die Anwendung darf beispielweise keinen Zustand speichern, also keine Sticky Sessions. Die Verwaltung einer PaaS Anwendung kann bei den meisten Cloud Anbietern entweder über eine API oder über ein entsprechendes Web-Interface getätigt werden. Ein Deployment kann direkt aus der Entwicklungsumgebung heraus geschehen. So zum Beispiel beim Google App Engine aus Eclipse heraus.

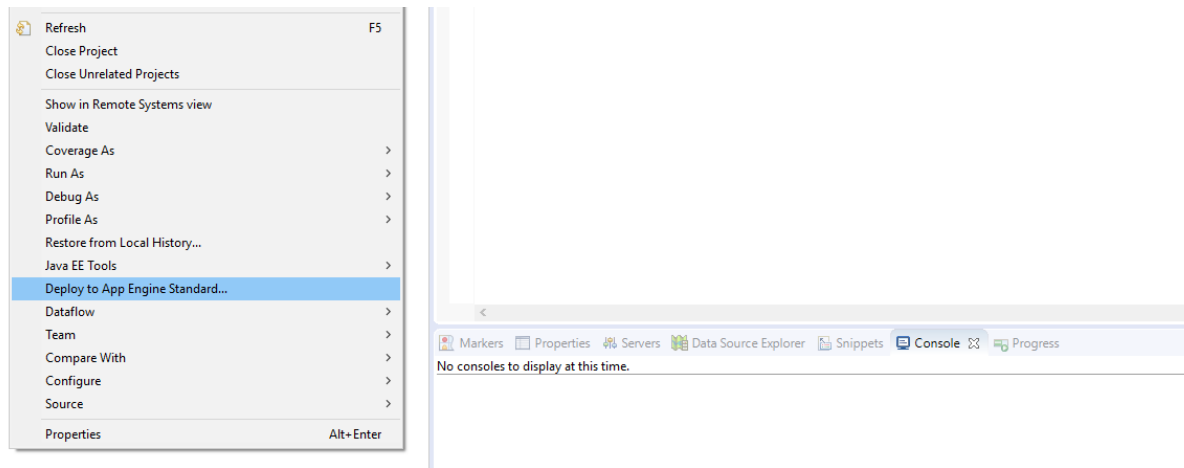


Abbildung 16: Google App Engine in Eclipse

Laufzeitumgebung

Die Laufzeitumgebungen der Cloud Anbieter bieten nur einen Teil der eigentlichen Laufzeitanwendung an. Diese Beschränkung wurde gemacht um entsprechende Eigenschaften wie Skalierbarkeit optimal anzubieten. In der Google App Engine ist es beispielsweise nicht erlaubt einen Thread zu starten. Um dennoch diese oder ähnliche Features zu nutzen, bieten die Anbieter weitere APIs zur Nutzung an.

Persistenz

Fast jede Anwendung muss Daten speichern und braucht daher Persistenz. In PaaS können die Daten aber nicht einfach auf die Festplatte gespeichert werden. Der Provider kann die Laufzeitumgebungen ausschalten und die Anwendung auf einer anderen Laufzeitumgebung wieder starten. Aus diesem Grund bieten die meisten PaaS-Anbieter verschiedene Lösungen zur Persistierung an. Meist nicht auf Basis von relationalen Datenbanken.

Vor- und Nachteile

Die Tabelle 8 fasst die Vor- und Nachteile einer PaaS Lösung zusammen.

Vorteile	Nachteile
<ul style="list-style-type: none">- Skalierung der Ressourcen wird automatisch durch den Anbieter erledigt- Nur Verantwortung für Daten und Code/Programm Schicht- Professioneller Support für die darunterliegenden Layers	<ul style="list-style-type: none">- Vendor bzw. Anbieter Locking- Always on (immer an)- Keine Rechte auf OS oder dergleichen- Für Datenbanken ungeeignet (Produkte vom jeweiligen Anbieter werden bevorzugt)- Spezielle Programmierweise nötig- Preise für Dienst eher hoch (Automatisch Skalierung zahlt man mit)

Tabelle 8: Vor- und Nachteile von PaaS

Provider von Platform as a Service

- Google Cloud⁷ (Google App Engine)
- App Service⁸ (Microsoft Azure)
- AppFog⁹ (Century Link)
- Platform¹⁰ (Heroku)
- AWS Elastic Beanstalk¹¹ (Amazon)
- Public Platform¹² (APPUiO)

3.2.4 Gegenüberstellung

In diesem Abschnitt wurde eine Gegenüberstellung aller Lösungen erstellt. Um den Vergleich möglichst aussagekräftig zu gestalten wurde drei konkrete Produkte aus den jeweiligen Kategorien ausgewählt. Für den Vergleich wurden weiter folgende Annahmen getroffen.

- Programmiersprache ist Java und Javascript
- Grösse und Anzahl der Instanzen
 - 4 Instanzen (1 API, 3 Webserver)
 - 2 GB pro Instanz
 - 1 GB Storage pro Instanz

Diese Annahmen sind exemplarisch und entsprechen nicht den schlussendlich benötigten Ressourcen für die TourLive Anwendung. Die Preise haben den Stand von Mitte März.

⁷ <https://cloud.google.com/appengine/?hl=de>

⁸ <https://azure.microsoft.com/de-de/services/app-service/>

⁹ <https://www.ctl.io/appfog/>

¹⁰ <https://www.heroku.com/platform>

¹¹ <https://aws.amazon.com/de/elasticbeanstalk/>

¹² <https://www.appuiio.ch/>

	IaaS am Beispiel von Cloud Server von meta-net.ch	Docker (Cloud Server metanet.ch mit Docker Engine)	PaaS am Beispiel von Google App Engine
Kosten *	85 CHF / Monat	69 CHF / Monat	150 Dollar / Monat
Vendor-Locking	nein	nein	ja
Maintaining ausgelagert	nein	nein	ja
Deployment	statische Skripts	Docker Files	via Entwicklungsumgebung
Verantwortlich für einen grossen Teil des Stacks	ja	ja	nein
Skalierbarkeit	Ja, aber mühsam	Ja (Docker Swarm)	ja
Konfigurierbarkeit	ja	ja	teilweise
Infrastruktur extern	Ja (intern auch möglich)	ja	ja
Programmiersprachen	Java, C#, etc.	Jave, C#, etc.	Java, jedoch Abhängigkeiten zu Provider bezüglich Versionen und Erweiterungen
Technologiefreiheit	ja	ja	teilweise, Abhängig von Provider
Datenbankfreiheit	ja	ja	teilweise, Abhängig von Provider
Verwaltungsrechte	ja	ja	Nein, Provider verwaltet Infrastruktur

Tabelle 9: Gegenüberstellung Deployment-Modelle

3.2.5 Entscheidung

Im Rahmen der Evaluation haben wir uns für eine Lösung mit Container und damit für Docker entschieden. Unter diversen Gesichtspunkten sehen wir diese Lösung als die am geeignetsten. Im folgenden Abschnitt werden die Gründe dafür kurz erläutert.

Für den Entscheid waren uns folgende Kriterien wichtig:

- Kosten (nicht zu hoch)
- Wartbarkeit (möglichst einfach, so dass die Anwendung nach unserer BA auch noch gut gewartet werden kann)
- Möglichkeiten zur Skalierung (innert wenigen Minuten)
- Freiheit bei den Programmiersprachen (da Technologien noch nicht feststehen)

In Anbetracht dieser Kriterien und den Informationen aus der Evaluation (Tabelle 9) fiel der Entscheid auf Container und damit auf Docker. Diese Lösung weist das beste Kosten / Nutzen Verhältnis auf. Im Vergleich zu IaaS sind wir zwar weiter für einen grossen Teil des Stacks verantwortlich, haben aber Vorteile beim Management. Mittels den Dockerfiles können wir unsere Software sozusagen paketieren und auf jedem beliebigen Server laufen lassen (ohne Änderung). Über das Docker Command können wir die Instanzen entsprechend schnell skalieren und sind innert Minuten besser aufgestellt. Eine Skalierung mit virtuellen Maschinen bei IaaS wäre zwar auch gegeben, es muss allerdings eine zusätzliche Maschine konfiguriert werden.

Eine PaaS Lösung erachten wir aufgrund der hohen Kosten und den vielen Einschränkungen für dieses Projekt nicht als sinnvoll. Diese Punkte überwiegen das Hauptargument der schnellen Skalierung bei einem PaaS-Provider.

3.3 Map Library im Frontend

Zur Darstellung der Daten in der Zuschaueranwendung werden unter anderem Karten eingesetzt. Für diese Karten möchten wir auf bereits vorhandenen Bibliotheken (Libraries) aufbauen um nicht komplett von 0 beginnen zu müssen. Analysiert wurden sowohl Bibliotheken mit Google Maps Daten also auch Bibliotheken mit OpenStreetMap-Daten (freizugängliches und veränderbares Kartenmaterial).

3.3.1 Anforderungen

An die Map Library wurden folgende Anforderungen gestellt:

- Einsatz möglichst einfach
- Kein Einsatz von zusätzlichen Tools
- Schnelle Ladezeiten
- Keine Kosten
- Keine API Key für die Verwendung (problematisch da Client-seitig)

3.3.2 React Leaflet

Mit der Erweiterung «React Leaflet» wird ein Wrapper zur der allgemein verfügbaren Leaflet-Karten-Bibliothek zur Verfügung gestellt. Leaflet basiert auf den OpenStreetMap-Daten, welche für jeden frei zugänglich sind. Sie arbeitet mit Tiles in Form von Bildern. Pro Zoomlevel und Standort werden eine gewisse Anzahl Bilder heruntergeladen und als Karten zusammengesetzt.

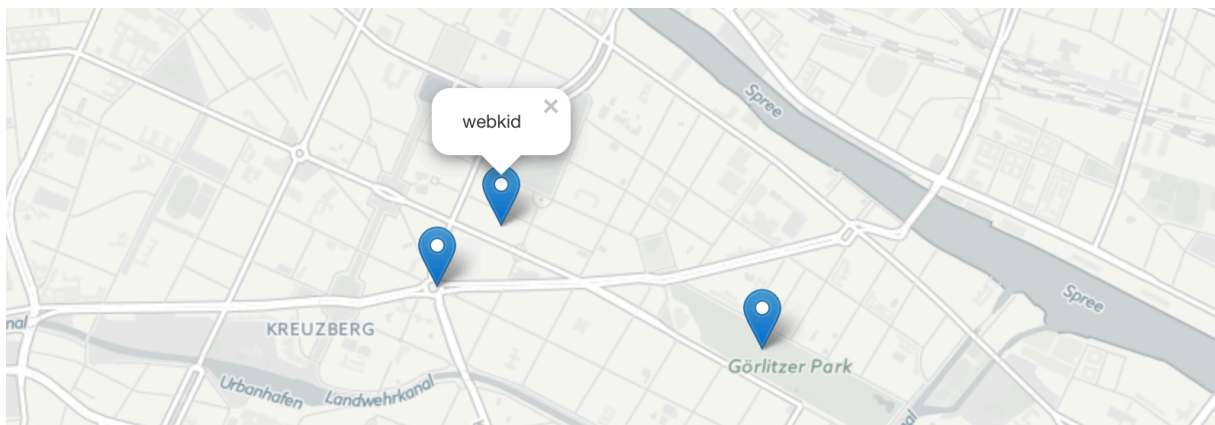


Abbildung 17: React Leaflet Beispiel [9]

Die Nutzung ist sehr gut dokumentiert und lässt viele eigene Erweiterungen zu. Dazu gehören eigene Icons oder Linien, welche auf der Karte gezeichnet werden können.

3.3.3 Google Map React

Diese Bibliothek basiert auf der API sowie den Kartendaten von Google. Die Nutzung dieser Daten setzt einen API-Key voraus. Ohne API Key können keine Google Dienste genutzt werden. Mit dieser Komponente wird die Nutzung stark vereinfacht. Die Dokumentation dazu ist mehr als ausreichend vorhanden und sollte für unseren Einsatz ausreichen. Auch offline können die Daten teilweise angezeigt werden. Die Platzierung von Punkte (sogenannten Markern) auf der Karte ist ebenfalls möglich. Durch die Anbindung von Google ist es ebenfalls möglich Informationen zum Verkehr sowie die Routenberechnung zu verwenden.



Abbildung 18: Google Map React Beispiel [10]

3.3.4 React MapGL

Bei dieser Bibliothek handelt es sich ebenfalls um eine, welche die OpenStreetMap Daten nutzt. Hier wird nicht mit Tiles als Bilder gearbeitet, sondern mit Tiles als Vektoren. Das Arbeiten mit Vektoren ermöglicht schärfere Bilder. Im Hintergrund wird auf WebGL gesetzt. Der Einsatz von WebGL ermöglicht, dass auf der Karte auch mehrere tausend Punkte dargestellt werden können (wie auf dem untenstehenden Bild angedeutet).

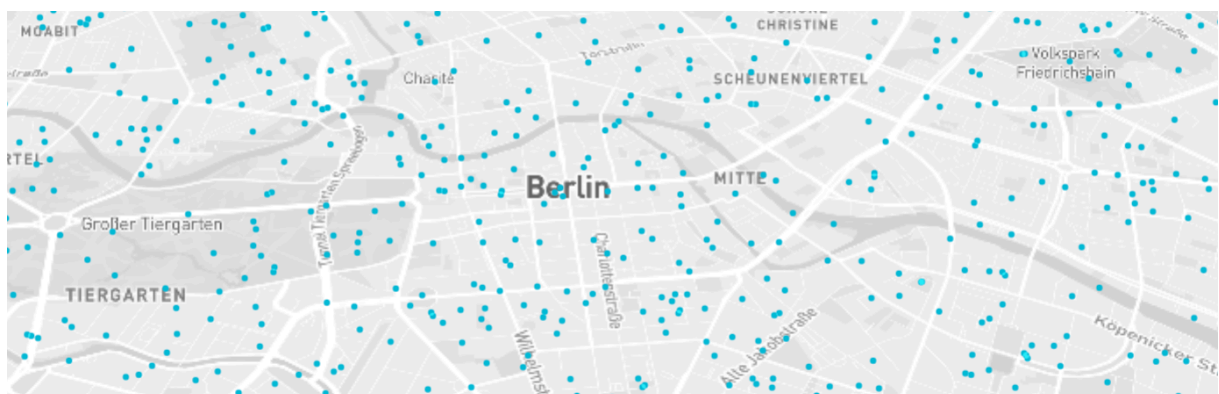


Abbildung 19: React MapGL Beispiel [11]

3.3.5 Entscheidung

Unter den im Vorfeld gestellten Anforderungen sowie zusätzlichen Erfahrungen von Mitstudenten wird für die Bachelorarbeit die React Leaflet Bibliothek eingesetzt. Der Einsatz von React MapGL wäre grundsätzlich auch machbar gewesen. Für den uns vorliegenden Use Case ist es nicht unbedingt nötig, dass wir sogenannte «Vektor Tiles» einsetzen müssen. Zudem hält sich die Anzahl Marker in Grenzen weswegen WebGL als Technologie ebenfalls nicht notwendig ist (ab 10'000 Marker). Bei einem Marker handelt es sich um ein Icon bzw. um ein eingezeichneter Punkt auf der Karte.

Auf die Nutzung von Google Map React wurde verzichtet, da Google für die Verwendung ihrer API einen API Key voraussetzt. Dieser API Key wäre bei allen Benutzern hinterlegt und sichtbar und somit nicht mehr sicher.

3.4 Speicherung der Daten

In diesem Abschnitt wird auf grundlegende Entscheide eingegangen, welche Art von Persistierung aus welchem Grund genutzt wird. Zudem wird festgelegt, welche Datenspeicherungslösung für das Projekt am sinnvollsten ist.

Eine Grundlegende Frage zur Datenspeicherung ist, ob die Daten redundant oder nicht redundant zur Verfügung stehen sollen. Ein gängiges Beispiel für Datenredundanz findet sich in der Computerwelt wieder. Ist ein Computersystem mit mehreren Harddisks vorhanden werden diese vielfach gespiegelt, um bei einem Verlust einer Harddisk auf eine andere ausweichen zu können. Dabei werden sämtliche Daten von der einen Harddisk zur Anderen laufend gespiegelt. Würde diese Redundanz nicht eingesetzt werden wären bei einem Harddisk-Defekt sämtliche Daten verloren.

Datenredundanz im Allgemeinen bedeutet also kurz gesagt das mehrfache Ablegen der gleichen Informationen. Dabei muss die Synchronität der Daten zwischen den verschiedenen Ablageorten stets gewährleistet werden oder entsprechende Algorithmen zur Wiederherstellung implementiert werden.

3.4.1 Keine redundante Persistierung

Der Vorteil dieser Lösung liegt darin, dass Speicherplatz in der Datenbank gespart werden kann. Zudem ist die Kopplung der Daten geringer, was ein flexibleres Bearbeiten der Daten ermöglicht. Aus dem Gesichtspunkt der Performance wirkt sich eine solche Persistierung allerdings negativ auf die Geschwindigkeit aus. Dies auf Grund dessen, da die miteinander verbundenen Daten bei jeder Abfrage neu berechnet werden müssen.

3.4.2 Redundante Persistierung

Durch eine redundante Persistierung der Daten wird in der Datenbank unnötig Speicherplatz verschwendet. Es erhöht zudem die Kopplung der Daten und das Bearbeiten der Daten ist durch diese Kopplung erschwert. Ein grosser Vorteil dieser Lösung ist jedoch die Performance, da Daten, die in Abhängigkeiten zueinanderstehen, an mehreren Orten verfügbar sind.

3.4.3 Fazit Persistierung

Auf Grund der Anforderung, dass die Web-Anwendung viele Nutzer (400 und mehr) gleichzeitig tragen muss ist es für die Umsetzung unabdingbar redundante Persistierung einzusetzen. Dies erhöht zwar den Speicherbedarf und die Kopplung der Daten, bietet jedoch einen Performancegewinn bei der Antwortzeit von Abfragen auf die Datenbank. Dies auf Grund dessen, da bei einem Request ca. zwei interne Abfragen gespart werden können.

3.4.4 Technologien [12]

Für die Persistierung der Daten gibt es verschiedene Möglichkeiten. Für das Projekt haben wir zwei Arten näher in Betracht gezogen. Einerseits das Prinzip des Key-Value Storage andererseits das Prinzip der Relationalen-Datenbank.

Folgendes Beispiel dient zur Veranschaulichung:

Ente
<ul style="list-style-type: none">- Key : String- Beine : int- Flügel : int- Farbe : Enum- Fliegend : boolean

Abbildung 20: Datenmodell Ente

Key-Value Storage (NoSQL)

Der Begriff NoSQL wird für den Ansatz der nicht relationalen Datenbankspeicherung genutzt. Das bedeutet, die Daten werden nicht in Tabellen gespeichert und die Datenbanksprache ist nicht SQL. Hier wird die sogenannte Key-Value Storage eingesetzt. Das bedeutet für einen spezifischen Schlüssel liegen verschiedenste Daten einer bestimmten Entität bereit. Ein Vorteil dieser Variante bildet sich dadurch, dass Entitäten vom gleichen Typ nicht alle definierten Merkmale dieser Entität im Speicher abbilden muss. Auf unser Beispieldatenmodell bezogen bedeutet das, wenn eine Entität vom Typ Ente gespeichert wird müssen nicht alle Merkmale definiert werden. Konkret ist es also möglich eine Entität Ente mit dem Key = „Kragenente“, Beine = „4“ und Farbe = „grau“ abzuspeichern. Eine weitere Entität kann unabhängig davon mit anderen und / oder denselben Attributen abgelegt werden. Zum Beispiel Key = „Löffelente“, Beine = „4“ und Fliegend = „wahr“.

Relationale Datenbank (SQL)

In der relationalen Datenbank-Welt werden die Entitäten in Tabellen abgespeichert. Hier gibt es für die jeweilige Entität einen spezifischen Schlüssel mit dem die Inhalte abgerufen werden können. Allerdings ist es bei diesem Modell möglich, Verknüpfungen zwischen Entitäten abzubilden und diese in Abhängigkeit zu stellen. Eine Einschränkung stellen hingegen die Einträge in den Tabellen dar, jede Spalte in der Tabelle muss einen definierten Wert enthalten um die Eindeutigkeit zu gewährleisten. Auf unser Beispieldatenmodell bezogen bedeutet das, wenn eine Entität vom Typ Ente gespeichert wird müssen alle Merkmale definiert werden oder standardmässig auf einen Initialwert gesetzt werden. Im Falle das eine Entität Ente mit dem Key = „Kragenente“, Beine = „4“ und Farbe = „grau“ gespeichert wird, werden automatisch die restlichen Attribute wie folgt ergänzt: Flügel = „0“, Fliegend = „falsch“.

3.4.5 Technologieentscheid Datenbank

Im Rahmen der TourLive Umgebung haben wir uns dazu entschieden, eine relationale Datenbank einzusetzen. Dies auf Grund der Gegebenheit, dass in unserem Datenmodell zwingend immer alle Attribute definiert sein müssen. Zusätzlich benötigen wir die Möglichkeit Verbindungen zwischen den Entitäten abzubilden, was im Key-Value Storage Bereich nicht so einfach möglich ist.

4 User Interface Design

Die Grundüberlegungen zum User Interface (Graphische Oberfläche) sind ein wichtiger Bestandteil dieser Bachelorarbeit. Denn nur ein sauber durchdachtes, intuitives Bedienpanel ermöglicht Erfolg im Einsatz in der Realität. Dabei gibt es viele Punkte zu beachten, welche später noch im Detail erläutert werden. Für das Design des Prototyps haben wir das Tool «JustInMind» [13] verwendet. Dieses erlaubt das Erstellen von klickbaren Prototypen und die Verwendung von Templates um die Elemente, die in jedem Screen vorhanden sind, nur einmalig zu designen.

4.1 Prototyp

Für den Prototyp haben wir jeden Screen graphisch abgebildet. Im Folgenden werden nur einzelne Screens ausgewählt und präsentiert. Der komplette Prototyp ist als Ganzes Projekt auf dem zusätzlich abgegebenen USB-Stick zu finden.

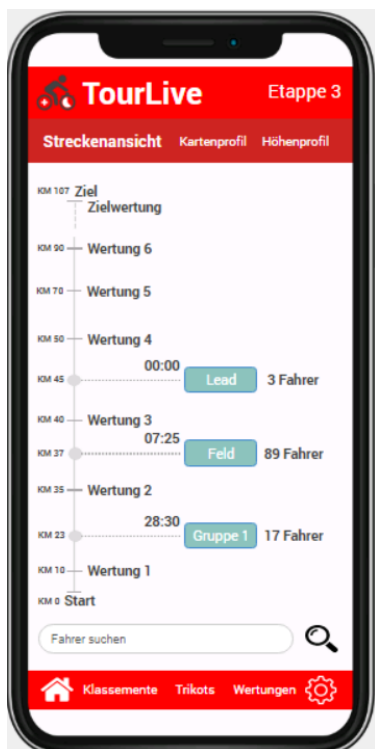


Abbildung 21: Prototyp Streckenansicht

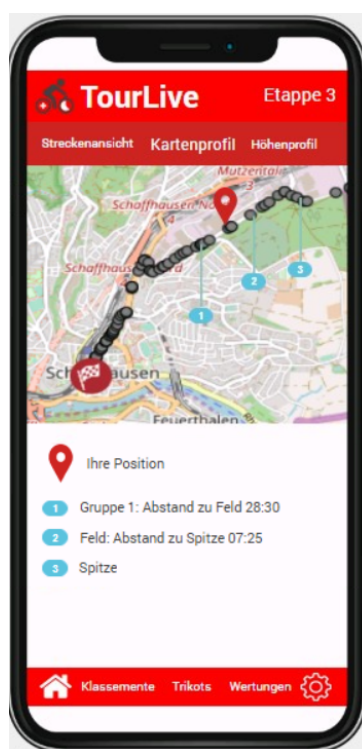


Abbildung 22: Prototyp Kartenprofil



Abbildung 23: Prototyp Wertungen

Für die Entwicklung des Prototyps wurden einige wichtige Punkte berücksichtigt. Die Hauptnavigationsleiste ist bewusst am unteren Bildschirmrand platziert worden. Die Platzierung wurde so gewählt, da die Navigationsleiste so in der Reichweite des Daumens des Nutzers liegt. Die am besten erreichbare Stelle für den Nutzer ist bekanntermaßen im unteren Drittel des Bildschirms. Der Einsatz eines Hamburger-Menüs haben wir bewusst abgelehnt, da uns dies im Modul HCID an der Hochschule HSR als nicht empfehlenswert vermittelt wurde.

Bei der Streckenansicht (Abbildung 21) ist die Leserichtung bewusst von unten nach oben gewählt worden. Der Grund liegt darin, dass das Rennen zum Schluss hin immer interessanter wird und der Zuschauer diese Information sofort im oberen Bereich des Bildschirms sehen soll, wenn er sich auf die Webseite einwählt. Durch automatisches Scrolling zum Rennkilometer der Feldgruppe hat der Zuschauer zudem immer die aktuelle Situation im Blick.

Im Kartenprofil (Abbildung 22) werden alle relevanten Streckenpunkte mit den entsprechenden Symbolen eingeblendet, um dem Zuschauer eine Übersicht zu verschaffen, wann welche Wertung vergeben wird. Weiter werden die Positionen der aktuell vorhandenen Renngruppen auf der Karte im Streckenverlauf angezeigt.

Um mehr Detailinformationen zu den Wertungen zu erhalten ist im Screen Wertungen (Abbildung 23) dieselbe Übersicht wie in der Streckenansicht dargestellt. Bei dieser ist es möglich zu den einzelnen Wertungen Informationen einzusehen. Das heisst, es wird ersichtlich, an welchem Kilometer diese Wertung vergeben wird und wer sie erhalten hat

4.2 Umsetzung

Als Grundlage für die Umsetzung der Zuschauer Webseite haben uns der im vorangegangenen Kapitel erarbeitete Prototyp gedient. Dabei sind während der Entwicklung einige Änderungen aufgetreten, die wir anhand der oben vorgestellten Ausschnitte des Prototyps genauer erläutern werden. Grundsätzlich sind die Texte aus der Navigationsleiste entfernt und mit Icons ersetzt worden. Aufgrund der Platzverhältnisse auf dem Smartphone wäre der Text etwas klein gewesen und daher kaum lesbar.



Abbildung 24: Streckenansicht umgesetzt



Abbildung 25: Kartenansicht umgesetzt



Abbildung 26: Wertungen umgesetzt

Bei der Streckenansicht (Abbildung 24: Streckenansicht umgesetzt) sind vermehrt Farben zum Einsatz gekommen. Die Farben dienen der Auflockerung des Screens für den Zuschauer. Es ist angenehmer ein Bild mit mehreren Farben zu betrachten, als ein Grautonbild. Für die Suche der Fahrer sind detaillierte Beschreibungen hinzugekommen, damit der Nutzer auch weiss was er mit diesem Feld machen kann, bzw. eingeben soll. Bei den Renngruppen werden weitere Detailinformationen angezeigt. Dazu zählen Informationen wie die Anzahl Fahrer sowie der Abstand zur Spitzengruppen. Die Renngruppen werden bewusst nicht im gleichen Stil wie die Wertungen dargestellt, so dass der Zuschauer jene unterscheiden kann.

In der Kartenansicht (Abbildung 25) werden die Wertungen sowie die aktuellen Renngruppen auf einer Karte dargestellt. Dadurch kann der Zuschauer jederzeit einordnen, wo auf der Strecke sich was befindet. Die Berechnung der Positionen passiert auf Basis von Abstände zwischen den Renngruppen und der Durchschnittsgeschwindigkeit des vorausfahrenden Autos.

Die Details der Wertungen (Abbildung 26) sind nicht mehr wie im Prototyp im gleichen Screen ersichtlich, sondern werden in einem separaten Screen eingeblendet. Während dem Entwickeln ist uns aufgefallen, dass die eingeblendete Informationen am unteren Rand nicht intuitiv ersichtlich sind. Deshalb haben wir uns dazu entschieden die Information aktiv in einem neuen Screen darzustellen, damit der Nutzer darauf aufmerksam wird.

5 Software Architektur

Im folgenden Kapitel werden die Systeme der TourLive Umgebung in ihrer Architektur detaillierter erläutert, um einen besseren Überblick über die Arbeit zu schaffen.

5.1 Systemübersicht

Das TourLive System besteht aus vier eigenen Systemen und insgesamt drei externen Abhängigkeiten. Namentlich sind folgende eigenen Systeme neu- bzw. weiter-entwickelt worden: Admin-Webanwendung, Zuschauer-Webanwendung (Frontend), TourLive API sowie die RadioTour Anwendung

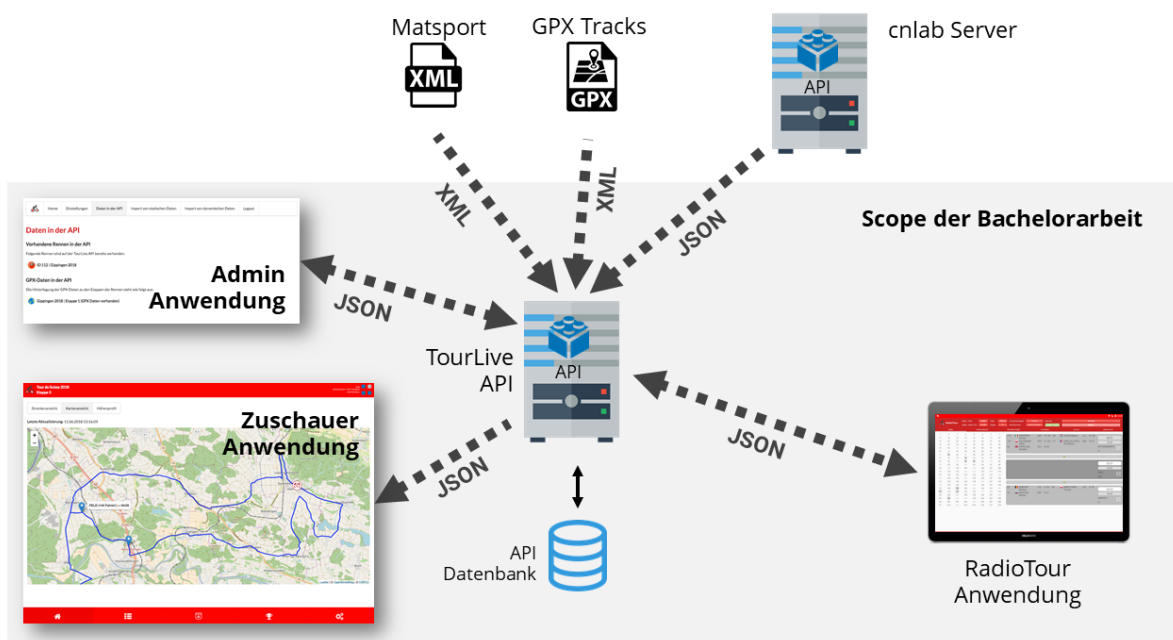


Abbildung 27: Systemübersicht

5.1.1 Eigene Systeme

TourLive API (Backend)

Die TourLive API dient als Schnittstelle für den Datenaustausch zwischen den verschiedenen Systemen. Dazu zählen sowohl die dynamischen als auch die statischen Daten. Die Daten des Radrennens werden in einer Datenbank persistiert und verwaltet. Die API versorgt sich von verschiedenen Quellen in verschiedenen Formaten mit Daten. Um Sie so effizient wie möglich zu machen wurden in der API aktiv Caching und Asynchronität eingesetzt.

Admin-Webanwendung

Die Admin Web-Applikation dient dazu die TourLive API zu verwalten. Mit dieser ist es möglich Einstellungen auf der API vorzunehmen, neue Daten zu importieren, Daten zu löschen und den aktuellen Status der Datenbank hinter der API abzufragen.

Über die Admin-Anwendung ist es möglich die externen Umsysteme in unser internes System zu integrieren. Die Anwendung tauscht Daten mit der API über das JSON Format aus und ist mit React & Redux implementiert.

Zuschauer-Webanwendung (Frontend)

Die Zuschauer-Webanwendung ist der eigentliche Hauptbestandteil der «TourLive»-Bachelorarbeit. Die auf React & Redux basierende Applikation stellt Informationen der Radrennen für den Zuschauer dar. Dabei werden die Daten für den Zuschauer in definierten Zeitabständen aktualisiert. Diese Zeitabstände können vom Zuschauer selber eingestellt werden, standardmässig betragen sie 10 Sekunden. Die abgefragten Daten werden von der TourLive API im JSON Format geliefert und in der Applikation interpretiert.

RadioTour Anwendung (Android App)

Die bereits in der Studienarbeit «RadioTour Anwendung» entwickelte RadioTour Anwendung ist in unser neues TourLive System integriert worden. Die Haupteiterweiterung der Anwendung ist die Übertragung von Änderungen während einem Radrennen. Das bedeutet, wenn die Applikation aktiv vom RadioTour Speaker verwendet wird und dieser z.B. neue Gruppen erstellt oder Wertungen vergibt werden diese Daten an den TourLive API im JSON Format übertragen und stehen dann für die Zuschaueranwendung zur Verfügung.

5.1.2 Externe Systeme

Matsport

Matsport ist aktuell der offizielle Zeitmesser der Tour de Suisse 2018. Nach jedem Rennen stellt Matsport die offiziellen Zeiten und Wertungspunkte der Radfahrer als XML-Dateien zur Verfügung. Um unser internes System nach jedem Rennen abzugleichen ist es möglich diese offiziellen XML-Dateien über die Admin Web-Applikation in unsere TourLive API und die dahinterliegende Datenbank einzulesen.

GPX Tracks

Für jedes Radrennen werden als Vorbereitung die Strecken der Etappen, welche für das Radrennen ausgewählt wurden, mit einem GPS Messgerät abgefahren. Aus den resultierenden Ergebnissen bilden sich die sogenannten GPX-Dateien. In diesen Dateien sind die Höhen- und Breitenangaben der Streckenpunkte abgelegt. Für eine Strecke von rund 180km ergeben sich somit rund 5'000 Geolokationen. Diese sind direkt über die Admin-Anwendung in unser System integrierbar und werden über diese in die Datenbank der TourLive API persistiert. Das Übertragungsformat in diesem Vorgang ist XML.

cnlab Server

Der cnlab Server dient als statische Datenquelle für das Radrennen. In ihm liegen Informationen zum Rennen, den Etappen, den Fahrern und den Wertungen. Zu Beginn eines Radrennens werden diese statischen Daten über die Admin-Anwendung in unser System importiert und persistiert. Während des Rennens ist unser System unabhängig von diesem externen System bis auf eine Ausnahme. Die GPS Daten der Begleitfahrzeuge eines Radrennens werden während einem Rennen vom cnlab Server bezogen. Die Übertragung der Daten passiert wiederum im JSON-Format.

5.2 Abläufe im System

Das entworfene System besteht aus mehreren Komponenten. All diese Komponenten tauschen Daten untereinander aus und arbeiten somit zusammen. Dieses Kapitel zeigt anhand von Beispielen, wie genau diese Abläufe von statten gehen.

Die Abbildung 28 zeigt den Benutzer als Einstiegspunkt. Je nach seinem Bedürfnis öffnet er in seinem Browser den Link zur Admin-Webanwendung (z.B. admin.tourlive.ch) oder zur Zuschauer-Webanwendung. Der Reverse Proxy regelt entsprechend zu welcher Komponente man weitergeleitet wird.

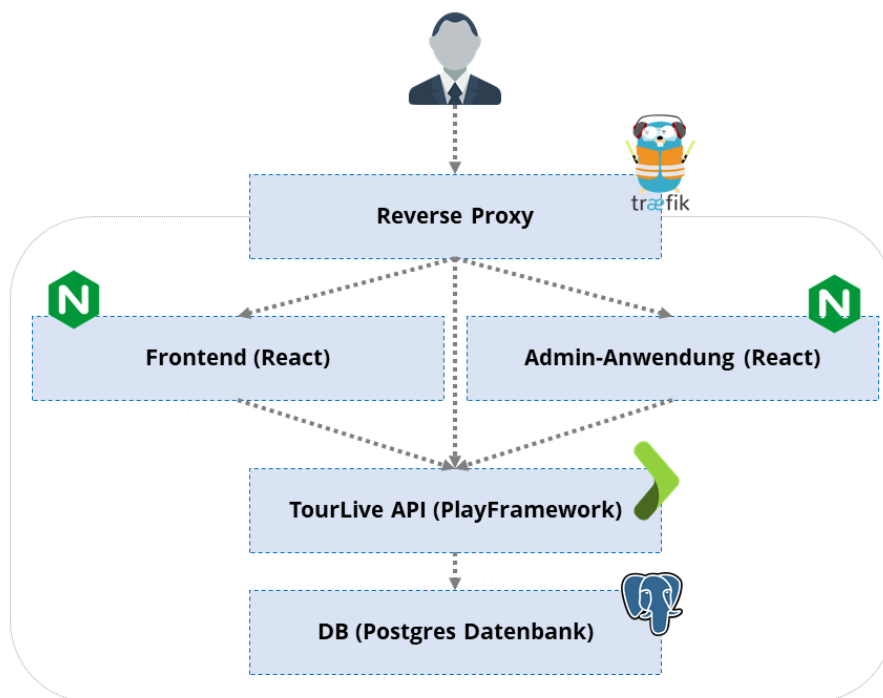


Abbildung 28: Verbindungsfluss vom Benutzer aus

Je nach Link oder Button, welcher der Benutzer im Frontend anklickt, werden unterschiedliche Aktionen durchgeführt. Dabei handelt es sich um ein Standardprozedere wie es von Webapplikationen üblich ist. Viel interessanter ist das Zusammenspiel der Systeme bzw. der Einfluss einer Aktion auf die anderen Systeme.

Das Sequenzdiagramm in Abbildung 29 zeigt genau eine Aktion durch alle Architekturschichten hindurch. Der Benutzer öffnet zum Beispiel die Ansicht mit den Klasselementen. Damit werden Routinen in der React-Anwendung durchlaufen und schlussendlich ein API Request für die aktuellen Klasselemente ausgelöst. Die TourLive API auf Basis des Play Frameworks nimmt diese Anfrage entgegen und ruft die Daten aus seiner Datenbank ab. Schlussendlich gibt die API die gefundenen Daten asynchron zurück. Die React-Anwendung speichert die Daten der API zwischen und bereitet diese für Frontend entsprechend auf. Die Daten werden dem Benutzer zum Ende in graphischer Form dargestellt.

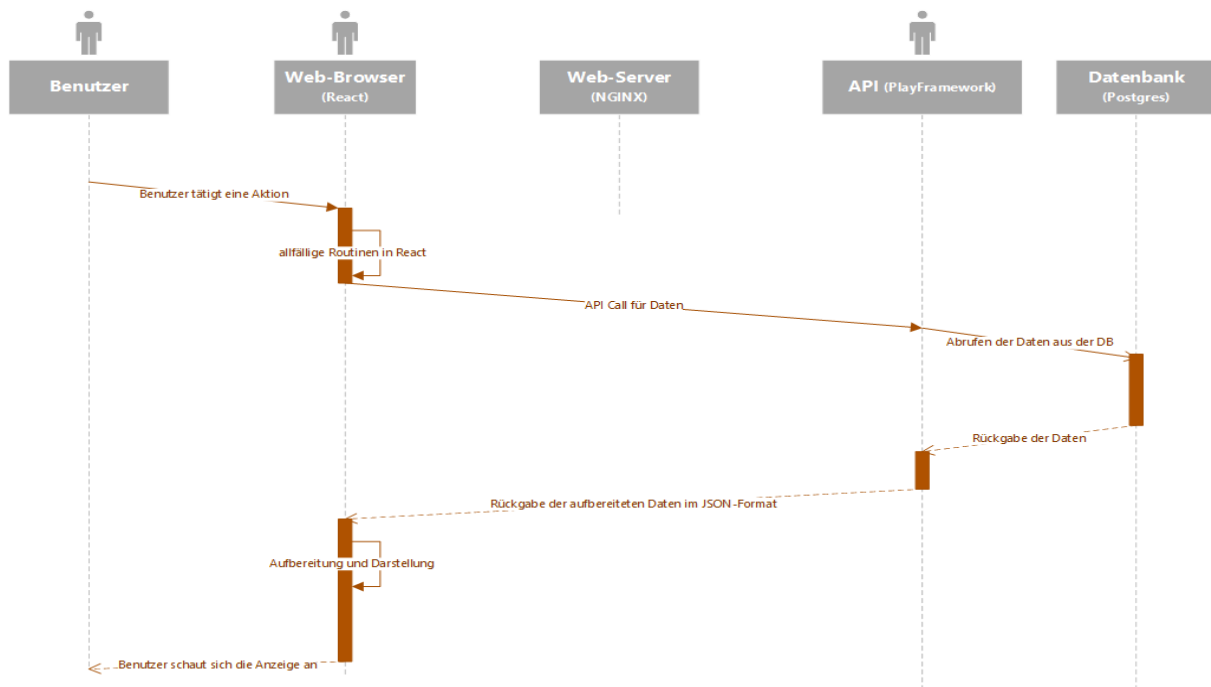


Abbildung 29: Abläufe und Auswirkungen einer Aktion im Frontend

Der oben erwähnte Ablauf ist für jegliche Aktionen aus dem Frontend gültig, welche einen Request nach sich ziehen. Die Frontend löst selbst im Hintergrund alle 10 Sekunden Abfragen auf die API aus und aktualisiert die Daten für das Frontend.

5.3 Domainmodell

Die Domäne rund um die «TourLive»-Umgebung besteht aus den folgenden Entitäten.

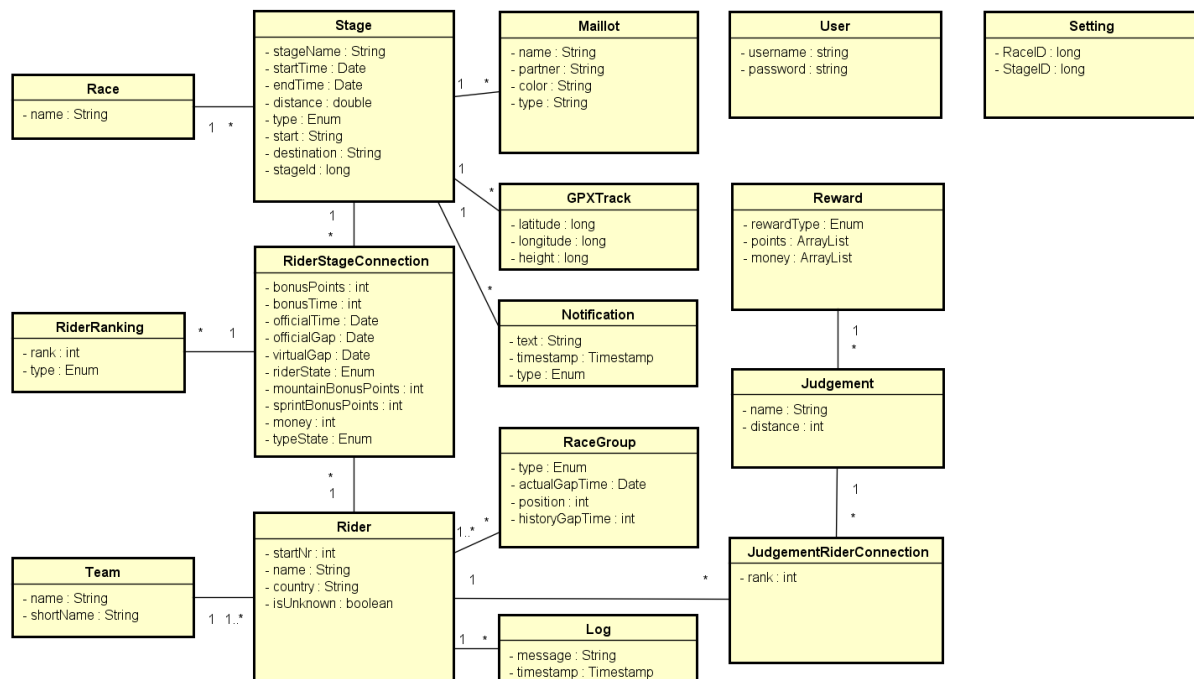


Abbildung 30: Domainmodell Backend

Die Entitäten der «TourLive» Umgebung im Detail.

Entität	Beschreibung
Race (Rennen / Tour)	Ein Race entspricht einem Rennevent (z.B. Tour de Suisse). Es weist nur einen Namen auf und dient zur Gruppierung der Etappen (Stages).
Stage (Etappe)	Stage (Etappe) Eine Stage repräsentiert eine Etappe und ist an ein Rennen (Race) gebunden.
Maillot (Trikot)	Das Maillot bedeutet ein spezielles Trikot, welches die Fahrer vor und während dem Rennen erreichen können. Ein Maillot kann in jeder Etappe einem anderen Fahrer zugewiesen sein und ist damit an die Stage (Etappe) gebunden.

Notification (Benachrichtigung)	Während einem Rennen ist es üblich, dass sich Änderungen in den Konstellationen von Gruppen ergeben oder Wertungen neu vergeben werden. Durch das Abspeichern dieser Änderungen als Benachrichtigungen ist es möglich den Benutzer zu informieren, wenn eine solche Änderung eintritt.
GPXTrack (GPX Lokationen)	Für die verschiedenen Rennen werden im Vorfeld die Streckenpunkte als GPXTracks erfasst. Diese Punkte werden in der API abgespeichert, um den Streckenverlauf z.B. auf einer Karte für die Nutzer darzustellen.
RiderStageConnection	Die RiderStageConnection bildet das Bindeglied zwischen einem Fahrer und einer Etappe. Sie repräsentiert den Stand (Zeiten, Punkte, ...) eines Fahrers während einer Etappe.
Rider (Fahrer)	Ein Rider repräsentiert einen Fahrer während eines Radrennens.
Team (Mannschaft)	Das Team (bestehend aus einem kurzen und langen Namen) repräsentiert alle Fahrer, welche für die gleiche Organisation unterwegs sind.
RaceGroup (Renngruppe)	Die RaceGroup bildet die verschiedenen Fahrergruppen ab, welche während eines Radrennens entstehen. Unterschieden wird zwischen Feld, Spitzengruppe und normalen Gruppen.
Log	Ein Log ist ein Eintrag in der Datenbank, wenn sich eine Änderung ergeben hat, die einen spezifischen Fahrer betrifft. Dabei gibt es verschiedene Typen von Logs die zum einen seinen Status betreffen (Aktiv, Arzt, ...) oder zum anderen eine Änderung seiner Position im Rennen widerspiegeln (z.B. Renngruppenwechsel). Diese Informationen sind nutzbar um eine Historie des Fahrers für die Zuschauer darzustellen.
Judgment (Wertung)	Ein Judgement ist eine Wertung, welche an bestimmten Positionen während eines Rennens aufgenommen werden.

Reward	Der Reward umschreibt die Punkte und Zeitgutschriften, welche bei der jeweiligen Wertung gutgeschrieben werden
JudgmentRiderConnection	JudgementRiderConnection ist eine Entität, welche die Beziehung eines Rider und eines Judgments (Wertung) genauer spezifiziert.
RiderRanking (Rangierung der Fahrer)	Ein RiderRanking ist eine Rangierung innerhalb einer bestimmten Gesamtwertung (z.B. Sprint, Berg, ...)
User (Benutzer)	Ein User in der Datenbank wird benötigt, um administrative Aufgaben zu verwalten und diese auch nur für bestimmte User nutzbar zu machen. Zuschauer benötigen keine User Accounts. Dieser Benutzer zur Sicherung von Manipulationen.
Setting (Einstellungen)	In den Settings wird das aktuelle Rennen und die aktuelle Etappe gespeichert um dem Nutzer der Daten zu signalisieren welche Routen in der API anzusprechen sind.

Tabelle 10: Elemente der Domäne

5.4 Datenmodell

Das Datenmodell zeigt die Struktur, wie die Daten in der lokalen Datenbank (PostgreSQL) der API gespeichert werden. Das gesamte Datenmodell wurde so ausgelegt, dass in der Datenbank gleichzeitig mehrere Etappen und Rennen gespeichert werden können. Durch die Einbindung von diversen Enum Datentypen ist die Anwendung offen für Modifikationen falls sich zukünftig Änderungen oder Erweiterungen in den jeweiligen Typen ergeben.

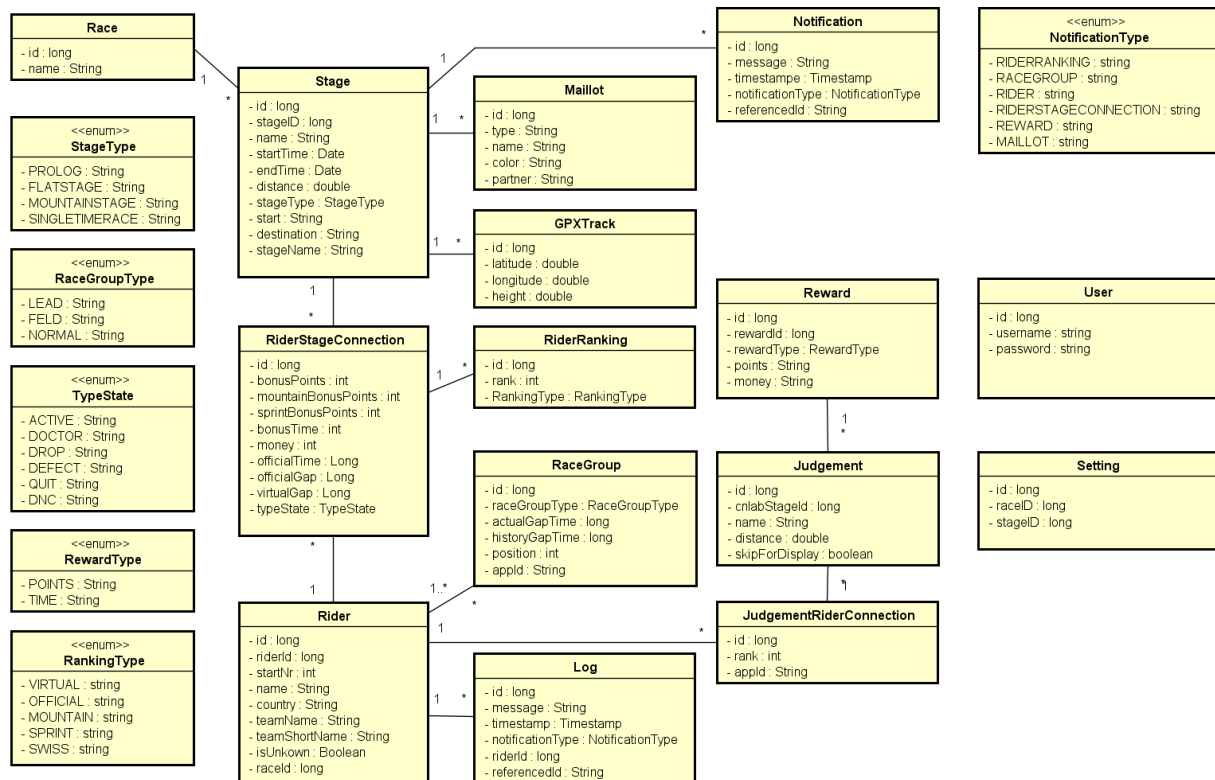


Abbildung 31: Datenmodell Backend

5.5 Deployment

Im Kapitel 3.2 wurde bereits, ohne die genaue Architektur zu kennen, eine Deployment-Lösung evaluiert. In dieser Evaluation fiel der Entscheid auf die Container basierte Lösung «Docker». Der Einsatz von Docker ermöglicht das System mit wenigen Handgriffen auf einem neuen Server zu deployen.

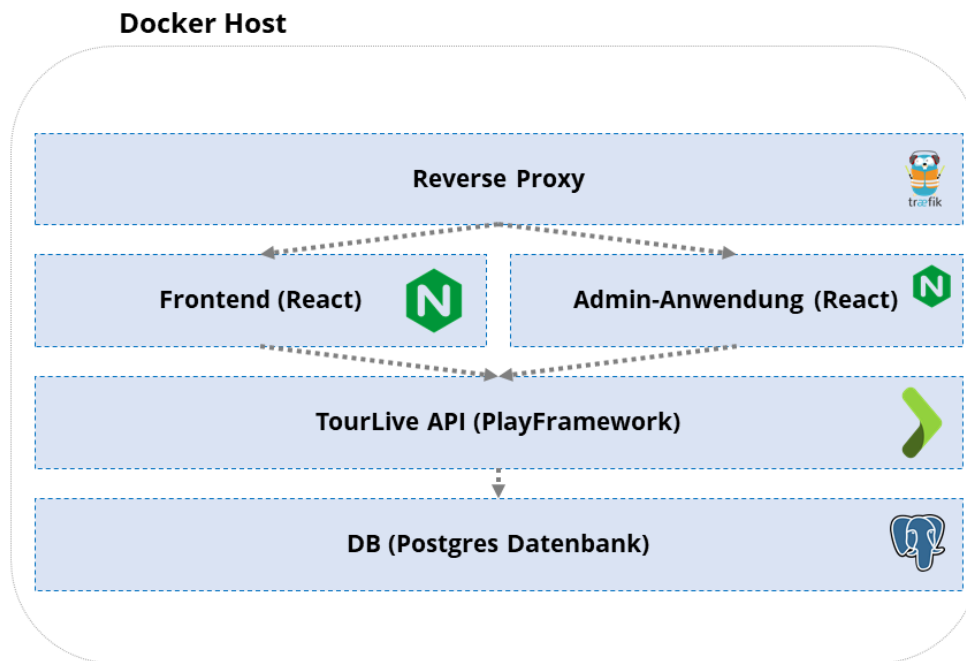


Abbildung 32: Docker-Container

Das TourLive System besteht aus mehreren Docker-Container und wird daher als Docker Container Gruppe veröffentlicht und betrieben. Die einzelnen Teile des Systems sind in jeweils eigenständigen Docker-Container zu finden und sind über ein internes Netzwerk miteinander verbunden. Alle Container mit einem Port, welcher auch gegen aussen erreicht ist, verfügen zudem über ein zusätzliches Proxy-Netzwerk. Über das Proxy-Netzwerk werden nur öffentliche Daten übermittelt. Die gesamte interne Kommunikation fließt über das bereits erwähnte «interne Netzwerk».

Reverse Proxy

Die Aufgabe des Reverse Proxy besteht darin alle öffentlichen Teilbereiche zusammenzufügen, sodass Sie unter derselben IP-Adresse verfügbar sind. Die Entscheidung zum welchen Container die Anfrage geleitet werden soll passiert auf Basis des Domainnamens. Die Konfiguration findet über Dateien statt und ist somit dynamisch. Weiter stellt der Reverse Proxy für alle HTTP-Endpunkte HTTPS zur Verfügung.

Zuschauer-Webanwendung (Frontend)

Der Frontend-Container beinhaltet die React-Applikation für die Zuschauer. Zudem beinhaltet es die Bilder der Fahrer, welche bei der Suche verwendet werden. Die Auslieferung der React-Applikation findet mit einem NGINX-Webserver statt. Es teilt sich keine Daten mit anderen Systemen.

Admin-Webanwendung

Der Admin-Container beinhaltet die React-Applikation für den Administrator. Die Auslieferung findet auch hier mit einem NGINX-Webserver statt. Es teilt sich ebenfalls keine Daten mit anderen Systemen.

TourLive API (PlayFramework)

Innerhalb des TourLive API – Containers (Backend) wird die mit dem PlayFramework programmierte API zur Verfügung gestellt. Die TourLive API dient als Schnittstelle zwischen dem Frontend und der RadioTour Anwendung (App). Dafür speichert es die Daten im Datenbank-Container ab.

DB (Postgres Datenbank)

Der Datenbank-Container dient zur Persistierung der Daten aus der TourLive API heraus.

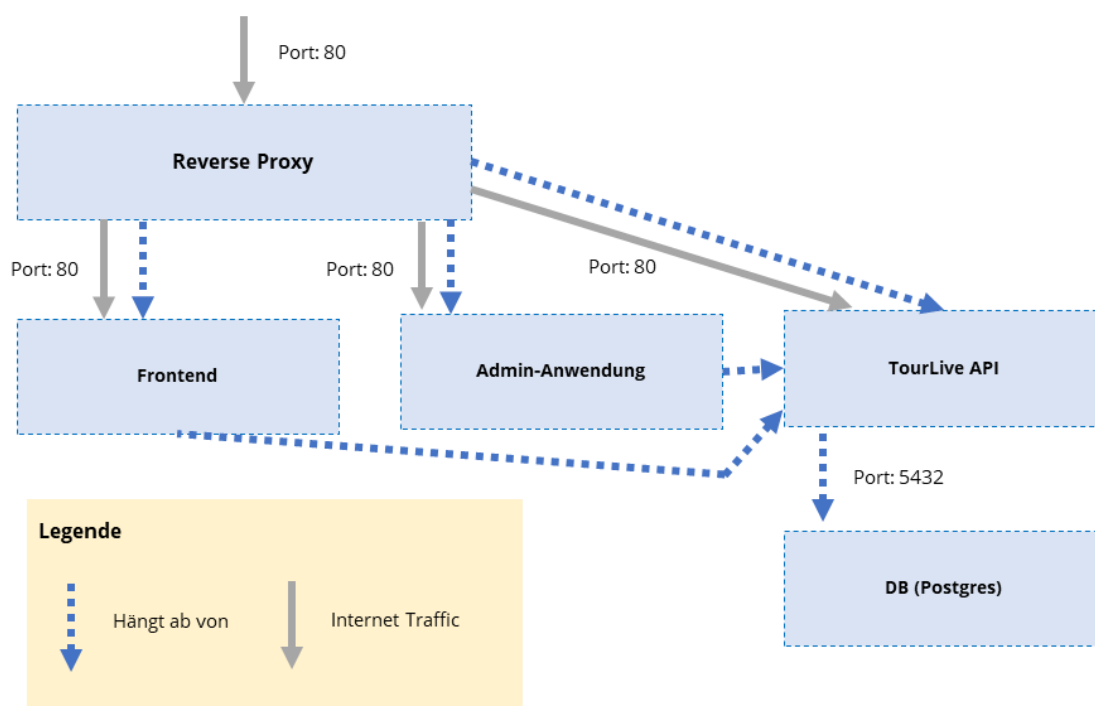


Abbildung 33: Abhängigkeiten der Docker Container

Der Reverse Proxy hängt von den beiden NGINX-Container sowie der TourLive API ab. Die beiden NGINX-Container wiederum hängen von der TourLive API ab. Zuletzt hängt die TourLive API von der Datenbank ab. Entsprechend wurde auch die Startreihenfolge so umgesetzt. Zuerst startete die Datenbank, dann die TourLive API, dann die NGINX-Webserver mit den Webanwendungen und zuletzt der Reverse Proxy.

5.5.1 Build

Das Erstellen (zu Englisch „Builden“) der Anwendung findet automatisiert statt. Dazu wird das Build-System Travis verwendet. Travis ist an die GitHub Repos der TourLive Umgebung gebunden. Nach ein Pull Request in den Master-Branch gemerget worden ist, startet Travis automatisch mit dem Erstellen des Docker Images (Abbild), lädt dieses hoch und startet die Docker-Container mit dem neuen Images erneut. Der detaillierte Ablauf des ganzen Continuous Integrations Workflow ist im Kapitel 9.9.3 beschrieben.

6 Realisierung

Die TourLive Arbeit wurde in drei Komponenten aufgeteilt (wie man den Anforderungsspezifikationen entnehmen kann). Die React-Anwendung «Frontend» bildet die Ansicht für den Zuschauer, die React-Anwendung «Admin» die Verwaltung des Systems und das Playframework im Sinne der API das Backend. Dazu kommt die bereits bestehende Komponente «RadioTour Anwendung» als Eingabequelle der Daten, welche vom RadioTour Speaker bedient wird.

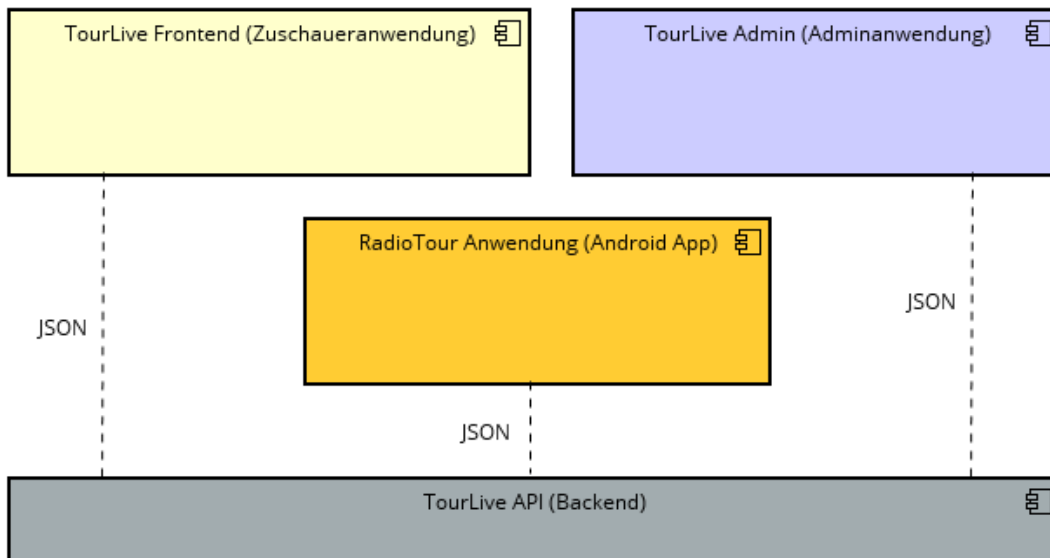


Abbildung 34: Kommunikation zwischen Komponenten

Die Kommunikation zwischen den verschiedenen Komponenten wird durch die TourLive API bewerkstelligt. Diese Schnittstelle basiert auf dem REST-Paradigma, wurde aber nur in einer RESTful Version umgesetzt. Als Behälter zur Übertragung der Daten dient das gängige und kompakte Datenformat JSON (JavaScript Object Nation). Die Form des JSONs ist so gestaltet und strukturiert, dass auch der Mensch den Inhalt einfach lesen und verstehen kann.

```
1 {  
2   "ID": 1209,  
3   "Name": "Bachelorarbeit",  
4   "ECTS": 12,  
5   "Voraussetzungen":  
6   {  
7     "ID": 1208,  
8     "Name": "Studienarbeit",  
9     "ECTS": 8  
10  }  
11 }
```

Abbildung 35: Beispiel von JSON

Das obige Beispiel von JSON zeigt diese Umstände sehr gut. Relationen und Verschachtelung von Objekten sind mit JSON ebenfalls realisierbar.

6.1 Frontend sowie Admin-Anwendung

Das Frontend (Webanwendung für die Zuschauer) sowohl als auch die Admin-Anwendung wurden mittels dem Framework React realisiert. Bei der darunterliegenden Programmiersprache handelt es sich um JavaScript. JavaScript selbst ist eine «weakly typed» Programmiersprache. Die Sprache unterstützt Objektorientierung, besitzt aber keine Klassen. Bei der aktuell verfügbaren Version handelt es sich um ECMAScript 2017.

6.1.1 React Komponenten

Das Hauptkonzept von React sind die sogenannten React Komponenten. Jeder dieser React Komponenten implementiert eine **render()**-Methode, welche Daten als Eingabe (als sogenannte Props) nimmt und diese dann entsprechend zurückgibt. Im untenstehenden Beispiel (Abbildung 36) wird der Rückgabewert je nach Index anders gewählt. Die Definition des Rückgabewertes findet im sogenannten JSX statt. JSX bringt JavaScript und HTML zusammen.

```
1 import React, {Component} from "react";
2
3 class RiderRaceGroupElement extends Component {
4   render() {
5     const rG = this.props.data;
6     const index = this.props.index;
7
8     const dt = new Date(rG.timestamp);
9
10    return(
11      <div className="history-element">
12        {index === 0 ? (
13          <span>{rG.message} | Aktuelle Renngruppe ({dt.timeNow()})</span>
14        ): <span>{rG.message} ({dt.timeNow()})</span> }
15      </div>
16    );
17  }
18 }
19
20 export default RiderRaceGroupElement;
```

Abbildung 36: React Komponente am Beispiel vom RiderRaceGroupElement

Der Aufruf dieser Komponente erfolgt über einen entsprechenden Tag.

```
1 <RiderRaceGroupElement key={i} data={rG} index={i}/>
```

Abbildung 37: Verwendung einer React Komponente

Aufgrund von diesem Umstand dürfen die Komponenten nicht gleich wie bestehenden HTML-Elemente benannt werden, da ansonsten die Referenzierung nicht mehr eindeutig ist. Die Data- sowie die Index-Eigenschaft sind in diesem Fall die definierten Props.

Zusätzlich zu den «Props» kann jede Komponente einen eigenen State (zu Deutsch «Zustand») haben. Somit lässt sich relativ leicht ein periodisches Update der Daten von der API ausführen, wie es in der Abbildung 38 gezeigt wird.

```
1 class Judgments extends Component {
2   constructor(props){
3     super(props);
4
5     this.state = {
6       updated: false,
7       selected : false,
8       judgmentSelected: false,
9       judgment : {},
10      timer : null
11    };
12    ...
13  }
14  componentDidMount() {
15    let timer = setInterval(this.tick, store.getState().settings.refreshPeriod * 1000);
16    this.setState({timer});
17  }
18  componentWillUnmount() {
19    clearInterval(this.state.timer);
20  }
21  tick() {
22    let stageID = store.getState().actualStage.data.id;
23    if (stageID !== undefined) {
24      store.dispatch(judgmentRiderConnectionActions.getJudgmentRiderConnections(stageID));
25    }
26  }
27  ...
28 }
29 export default Judgments;
```

Abbildung 38: React Komponente "Judgments" mit State

Wird die Komponente, in diesem Fall die Komponente «Judgments», gemountet, so wird in der Methode **componentDidMount()** ein Timer im Zustand der Komponente gesetzt. Dieser Timer aktualisiert über die Methode **tick()** alle x-Sekunden die lokal vorhandenen Daten. Nachdem die Komponente unmounted (nicht mehr dargestellt) wurde, wird der Zustand wieder zurückgesetzt bzw. abgeräumt.

6.1.2 Redux (State Management)

Wie bereits im vorherigen Kapitel ausgeführt verfügt jede React Komponente über einen eigenen State. Zugriff auf diesen State ist nur innerhalb der Komponente möglich. Im Falle der Realisierung waren wir mit dem Problem konfrontiert, dass mehrere Komponenten die gleichen Daten benötigten. Somit hätte jede Komponente die Daten von der API neu holen müssen. In unserem Umfeld wären zusätzliche API-Aufrufe eine Belastung für den Server sowie für das Endgerät des Zuschauers.

Zur Lösung dieses Problem wurde Redux eingeführt. Bei Redux handelt es sich um einen globalen State Manager. Mit dem Einsatz von Redux wird ein Global Container (genannt

Store) eingeführt, welcher die gesamten Zustandsdaten in einem einzigen Objekt hält. Benötigen nun mehrere Komponenten die gleichen Daten bietet es sich an die Daten des API-Aufrufs, in diesem Store zu speichern. Die Komponenten greifen ebenfalls über diesen Store auf die Daten zu. Das Prinzip Redux unterteilt sich in Actions, Reducers und den Store. Für den richtigen Einsatz von Redux sind alle Konzepte notwendig. Sie sind nötig um den globalen State zu manipulieren.

Actions

Bei den Actions startet der Prozess der Manipulation. Der Aufruf einer solchen Action wird normalerweise aus einer React Komponente heraus gestartet und erfolgt über die Methode **dispatch()** des Stores.

```
1 store.dispatch(riderActions.getRidersFromAPI(id));
```

Abbildung 39: Auslösen einer Redux Action

In einer separaten Actions-Datei ist die über **dispatch()** mitgegebene Methode **getRidersFromAPI()** definiert. Diese Methode holt in diesem Fall alle aktuellen Rennfahrer von der TourLive API und gibt diese bei Erfolg über **dispatch()** an die nächste Methode weiter.

```
1 export function getRidersFromAPI(id) {
2   return function (dispatch) {
3     return axios({
4       url : api.LINK_RIDERS + id,
5       timeout : 20000,
6       method: 'get',
7       responseType: 'json'
8     }).then(function (response) {
9       dispatch(receiveRiders(response.data));
10    })
11  }
12 }
```

Abbildung 40: Zwischenmethode der Actions zur Abholung der Daten von der API

In dieser Methode findet der eigentliche Zusammenbau der Redux-Action statt. Wichtig dabei ist das Feld „type“. Dieses Feld gibt an, welche Action durchgeführt werden sollen. Zusätzliche Felder wie data oder dergleichen ermöglichen es der Manipulation Daten mitzugeben.

```
1 function receiveRiders(data) {
2   return {
3     type : types.GET_RIDERS,
4     data: data
5   }
6 }
```

Abbildung 41: Konstruktion der Redux Action

Reducers

Ein Reducer nimmt nun die eigentliche Manipulation am Store vor. Über den Store erhält er die Action und nimmt die Instruktionen je nach Type vor. Im Fall der Abbildung werden die aktuell gespeicherten Rennfahrer durch diese der API ersetzt. Wichtig dabei ist, dass alle Änderungen am State immutable sind. Konkret bedeutet das, die aktuellen Daten im Store werden kopiert, manipuliert und neu abgelegt. So ist der vorherige Zustand immer noch unverändert verfügbar, die Komponenten aktualisieren sich gemäss dem neuen State. Durch diese Eigenschaft ist es möglich eine Zustandshistorie zu durchlaufen. Für unser Projekt wurde jeweils nur der aktuellste State verwendet.

```
1 import * as types from "../actions/actionTypes";
2
3 const initialState = {
4   riders: [],
5   error: false
6 };
7
8 const riderReducer = (state = initialState, action) => {
9   switch(action.type) {
10     case types.GET_RIDERS:
11       return Object.assign({}, state, {
12         riders: action.data,
13         error : false
14       });
15     default:
16       return state;
17   }
18 };
19
20 export default riderReducer;
```

Abbildung 42: Redux Reducer am Beispiel der Rennfahrer

Store

Der Store von Redux lässt sich als globaler Container beschreiben, in welchem alle States (Zustände bzw. schlussendlich Daten) abgespeichert werden. Von überall her kann, wie in der Abbildung 43 gezeigt, auf diesen Store zugegriffen werden.

```
1 store.getState().actualStage.data
```

Abbildung 43: Abholen des globalen States über den Redux Store

Diesen Store gibt es in einer React-Anwendung genau nur einmal. Über die Methode `connect()` und `mapStateToProps()` wird der Redux Store so an eine React Komponente gebunden, welche automatisch nach Manipulationen am Store aktualisiert wird.

```
1 function mapStateToProps(store) {  
2   return {  
3     riders : store.riders.riders,  
4     actualStage : store.actualStage.data  
5   }  
6 }  
7  
8 const RiderSearchContainer = connect(mapStateToProps)(RiderSearch)
```

Abbildung 44: `Connect()` und `MapStateToProps()` in Verwendung

Der Einsatz von Redux für React hat die Arbeit mit Daten massiv vereinfacht. Der durch die Actions und Reducers entstehende Overhead hat sich allemal gelohnt und bildet ein sehr solides Fundament für die Zukunft.

6.1.3 Organisation des Projekts

React gibt keinerlei Vorschrift hinsichtlich Struktur bzw. Ablage der Komponenten und Dateien innerhalb eines Projektes. Ebenso ist keine einheitliche Richtlinie zur Separierung von React-Code vorhanden.

Um diesem Problem entgegen zu wirken, haben wir das gesamte Projekt anhand folgendem Prinzip aufgeteilt bzw. separiert:

1. Separierung von Komponenten, Actions und Reducers in entsprechenden Ordner
2. Erstellung eines Ordners «Util» für entsprechende Util-Klassen
3. Einführung von Containers zur Entkopplung des Redux-Stores und den Komponenten
4. Aufteilung von Komponenten und Containers in Ordner je nach gebrauchten Screen

Das Prinzip angewendet auf das Frontend ergibt diese Struktur.

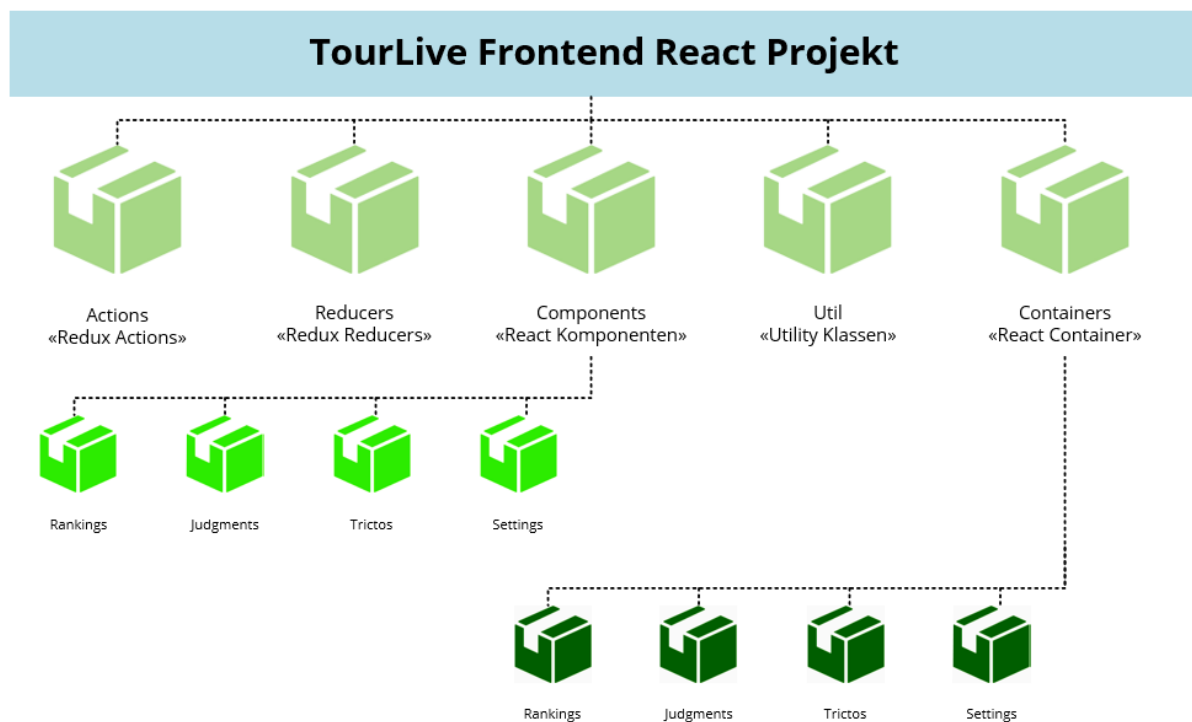


Abbildung 45: Projektstruktur "TourLive Frontend"

Für die Admin-Anwendung hat sich eine ähnliche, aber einfachere Struktur ergeben. Es wird aber darauf verzichtet, diese an dieser Stelle ebenfalls zu zeigen.

6.1.4 Asynchronität im Frontend

Asynchronität ist das A und O bei einer Benutzeroberfläche. Das Frontend sowie die Admin-Anwendung sind beides solche Benutzeroberflächen. Ohne Asynchronität würden die synchron ausgeführten Aufrufe und Aktionen die Oberfläche blockieren und weitere Interaktionen durch den Benutzer verhindern. Ein absolutes No-Go.

Aus den oben erwähnten Gründen werden alle Aktionen asynchron ausgeführt. Für diesen Zweck kommt die Erweiterung «Axios» zum Einsatz. «Axios» ist ein promise basierter HTTP-Client für den Browser oder aus JavaScript heraus. Mit diesem Client sind einerseits normale http Request möglichst. Andererseits werden XMLHttpRequest auch unterstützt. Für unser Projekt ist zweites wichtiger. Der Client wird dazu gebraucht um bei der TourLive API abfragen zu tätigen.

```
1 return axios({
2     url : api.LINK_GPXTRACKS + id,
3     timeout : 20000,
4     method: 'get',
5     responseType: 'json'
6 }).then(function (response) {
7     if (response.status === 200) {
8         dispatch(receiveGPXTracks(response.data));
9     } else {
10        dispatch(receiveGPXTracksError("Error on loading data"));
11    }
12 }).catch(function (response) {
13     dispatch(receiveGPXTracksError(response));
14 });
```

Abbildung 46: Verwendung von Axios

Die Erweiterung unterstützt von Haus aus die Asynchronität und lässt sich entsprechend einfach verwenden. Mittels **then()** wird die Aktion angegeben, welche bei einem erfolgreichen Aufruf auf die TourLive API ausgeführt wird. **catch()** fängt allfällige Exceptions (Fehler) ab und verarbeitet diese ebenfalls entsprechend.

Für die React-Anwendung selbst ist die Asynchronität bereits implementiert. Durch das im Kapitel 3.1.2 «Virtual DOM Rendering» wird das UI nur aktualisiert, falls auch effektiv Änderungen vorhanden sind. Sie die Daten von der TourLive API noch nicht empfangen worden findet auch keine Blockierung statt.

6.1.5 Weitere eingesetzte Module und Erweiterungen

Für die Entwicklung des Frontends wurden diverse weitere Module und Erweiterungen eingesetzt, die in diesem Abschnitt detaillierter erläutert werden.

Semantic UI [1]

Semantic UI React, die offizielle Semantic UI React Integration. Mit Hilfe dieser Bibliothek lassen sich auf einfache Art und Weise Semantic UI Komponenten in einer React Anwendung einbinden. Semantic UI beinhaltet einzelne, bereits gestylte Komponenten. Dazu gehören beispielsweise ein Menü, ein Footer, Formulare oder Flaggen.

React Router [14]

Der React Router ist eine Bibliothek für das Routing innerhalb einer React-Anwendung. Mit dem React Router ist es möglich die Komponenten zu „Routen“ zuzuordnen. Dadurch fühlt sich eine Single-Page Applikation wie eine Multi-Page Applikation an.

Leaflet / React Leaflet [15]

Leaflet ist eine der führenden JavaScript-Bibliotheken für die Darstellung von interaktiven Karten. Die Bibliothek zeichnet sich durch Einfachheit, Performance und Bedienung aus. Sie wird meistens in Kombination mit OpenStreetMap Daten eingesetzt. Das React Plugin „React-Leaflet“ bietet lediglich eine Abstraktion von Leaflet als React Komponente und vereinfacht damit das Handling mit Leaflet um einen grossen Faktor.

React Helmet [16]

React Helmet ist eine wiederverwendbare Komponente, welche die Änderungen im Dokumenten Header des HTML (also dem Tag <head>) verwaltet. Sie ermöglicht zum Beispiel Änderungen des Seitentitels anhand der aktuellen Komponente.

React Notifications [17]

Bei React Notifications handelt es sich ebenfalls um eine reine UI-Library, welche für unsere Anwendung die Darstellung der eingehenden Nachrichten übernimmt (analog Windows oder macOS).

React Chart.JS [18]

React Chart.JS wird zur Darstellung des Höhenprofils gebraucht. Es handelt sich dabei um eine Charting-Bibliothek. Neben Liniendiagramme sind auch Kuchendiagramme, Doughnut-Diagramme oder ganznormale Säulendiagramme möglich. Das React-Plugin basiert auf der gleichnamigen, allgemein verfügbaren Bibliothek.

6.2 Backend (TourLive API)

Für die TourLive Umgebung ist eine Backend-API realisiert worden, die als Datenschnittstelle zwischen der RadioTour-Anwendung und den Endnutzer mit einer Web-Anwendung dient. Zu Beginn der Arbeit wurde definiert, dass die API zwingend mit Java umzusetzen ist.

6.2.1 Play [19]



Play erlaubt es während des Entwicklungsprozesses einen Hot-Reload auszuführen, das heisst Änderungen werden dynamisch übernommen und sind sofort ersichtlich. Diese agile Entwicklung bringt einen enormen Vorteil, da Fehler sofort ersichtlich werden.

Im Play Framework werden sogenannte Controllern eingesetzt, welche den ganzen Datenfluss steuern. In Play sind alle HTTP Grundfunktionalitäten unterstützt und Routen können so nach dem REST Prinzip aufgebaut werden.

Für uns wichtig ist auch, dass Java Objekte einfach ins JSON Format übertragen werden können, was uns den Aufwand für die Übertragung erheblich vereinfacht.

Im Play Framework gibt es die Möglichkeit eine Datenbank wie SQL Lite oder PostgreSQL für die Persistierung einzusetzen. Auch anderen Typen von Datenbanken sind möglich, z.B. eine In-Memory Datenbank. Zusätzlich gibt es die Option eine Datenbank in einem Docker Container, welche wir einsetzen, laufen zu lassen.

6.2.2 Asynchronität [20]

Das Playframework ist innerhalb seiner Architektur asynchron aufgebaut, das heisst alle Abfragen und Verarbeitungen werden nicht blockierend ausgeführt. In Java 8 wurden das generische Promise «CompletionStage» eingeführt, welches genau diese asynchrone Funktionalität unterstützt. Konkret bedeutet dies, beim Ansprechen von Routen wird nicht auf das Ergebnis gewartet, sondern das Ergebnis wird ausgeliefert sobald es vorhanden ist.

6.2.3 Dependency Injection [21]

Dependency Injection ist ein weit verbreitetes Pattern in der Softwareentwicklung. Es separiert das Verhalten der konkreten Komponenten auf Grund der Abhängigkeitsauflösung. Konkret bedeutet das, im Konstruktor der jeweiligen Klassen werden nur abstrakte Interfaces von Klassen mitgegeben. Die dahinterliegende Implementierung der Funktionalitäten in den abstrakten Klassen ist somit jederzeit durch eine Instanz einer konkreten Klasse austauschbar, welches das angegeben Interface implementiert.

6.2.4 Caching Allgemein [22]

Um die Laufzeit von Abfragen zu verbessern ist es üblich Caching, also temporäre Speicherung der Resultate, einzusetzen. Play bietet die Möglichkeit Caching einzusetzen, dabei wird die Strategie der eventuellen Caches verfolgt. Konkret bedeutet das, bei Anfragen an den Servern wird zuerst im Cache geprüft, ob das Resultat schon vorhanden ist, falls nicht wird es neu berechnet und in den Cache abgelegt. Dabei ist es möglich die abgelegten Daten eindeutig durch einen Schlüssel zu identifizieren und auch zu bestimmen, wie lange diese im Cache vorhanden sein sollen bevor sie wieder gelöscht werden.

6.2.5 Threadpools [23]

Im Playframework gibt es die Möglichkeit sogenannte Thread Pools einzusetzen. Diese bieten die Möglichkeit, bei korrekter Verwendung, den Workload gezielt auf verschiedene Threads zu verteilen. Dabei ist es möglich einen riesigen Performance-Gewinn zu erreichen. Laut der offiziellen Webseite des Playframeworks ist es so möglich mehrere hunderte Requests pro Sekunde zu beantworten.

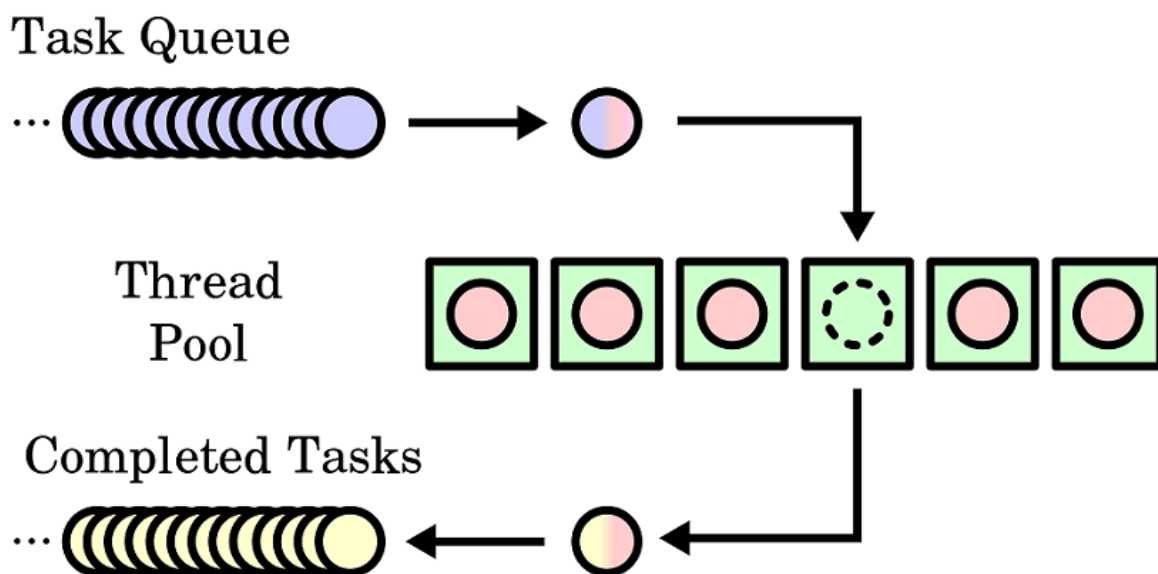



Abbildung 47: Thread Pool [24]

6.2.6 Swagger API Dokumentation

Dank dem Einsatz des Tools Swagger ist es möglich automatische API Dokumentationen zu generieren. Dabei wird ersichtlich welche Routen angesprochen werden können, welche Parameter mitgeschickt werden müssen (z.B. Username und Passwort) und mit welcher Antwort gerechnet werden kann. Die Generierung des Files erfolgt automatisch und wird mittels Annotationen direkt im Code bewerkstelligt.

Zusätzlich ist es möglich spezifische Teile im Code detaillierter zu erläutern oder solche Teile, die nicht ersichtlich sein sollen in der Swagger Dokumentation komplett auszublenden. Die Swagger Dokumentation kann bei einer laufenden Instanz unter `/docs` (z.B. `prod-api.tourlive.ch`) aufgerufen. Dieser Link leitet einem direkt an die richtige Stelle weiter.


swagger

<https://prod-api.tourlive.ch/swagger.json>

Explore

TourLive API beta

[Base URL: prod-api.tourlive.chhttps://prod-api.tourlive.ch]
<https://prod-api.tourlive.ch/swagger.json>

This API is a interface between all system of TourLive. It connect the android app with the web interface and persist the data
[Apache 2.0](#)

Schemes

HTTP

RiderRanking

Maillot

Application

JudgmentRiderConnection

GET /maillots/stages/{stageId} get all maillots of a stage

GET /maillots/{maillotId} get maillot by id

GET /judgmentriderconnections/{riderId} get all judgment rider connections of a specific rider

GET /judgmentriderconnections/stages/{stageId} get all judgment rider connections of a specific stage

POST /judgmentriderconnections/stages/{stageId}/{timestamp} add new judgment rider connection

DELETE /judgmentriderconnections/{appId} delete a judgment rider connection by appld

Abbildung 48: Swagger Docs

GET /maillots/{maillotId} get maillot by id

Try it out

Parameters

Name	Description
maillotid <small>required</small> integer (path)	

Responses

Response content type

application/json

Code	Description
200	<div>successful operation</div> <div> <div>Example Value</div> <div>Model</div> <div> <pre>{ "id": 0, "type": "string", "name": "string", "color": "string", "partner": "string", "riderId": 0 }</pre> </div> </div>
400	No specific maillot is set in DB for this id.
500	Error on getting maillot by id

Abbildung 49: Detaillierte Informationen zu einer Route

6.2.7 Aufbau allgemein (Controller, Repo)

Im Playframework werden Controller programmiert, die Requests entgegennehmen, verarbeiten und beantworten können. Um den ganzen Zusammenhang zu veranschaulichen wird der Ablauf anhand des RaceGroupController dargestellt. Controller bieten die Eigenschaft, dass über Dependency Injection Klassen eingeschleust werden, um diese anschließend in den einzelnen Methoden zu verwenden. Hier werden verschiedene Repositories übergeben welche Zugriffe und Manipulation auf der Datenbank ermöglichen. Auch die Verwendbarkeit der Caches wird hier im Konstruktor injiziert.

```
1 @Api("Racegroup")
2 public class RaceGroupController extends Controller {
3     private final RaceGroupRepository raceGroupRepository;
4     private final StageRepository stageRepository;
5     private final RiderRepository riderRepository;
6     private static final String ACTUAL_GAP_TIME = "actualGapTime";
7     private static final String HISTORY_GAP_TIME = "historyGapTime";
8     private final AsyncCacheApi cache;
9
10    @Inject
11    public RaceGroupController(RaceGroupRepository raceGroupRepository,
12                               StageRepository stageRepository, RiderRepository riderRepository,
13                               AsyncCacheApi cache) {
14        this.raceGroupRepository = raceGroupRepository;
15        this.stageRepository = stageRepository;
16        this.riderRepository = riderRepository;
17        this.cache = cache;
18    }
19    ...
20 }
```

Abbildung 50: RaceGroup Controller

Unten abgebildet ist eine konkrete Implementierung einer Route in der API. Diese wird mittels der «@API» Annotation an das automatische generierte Swagger-File weitergeleitet. Durch den Rückgabebetyp einer CompletionStage wird die Asynchronität bewerkstelligt. Dieses Feature wurde in Java 8 eingeführt. Mit dem Befehl «**raceGroupRepository.getAllRaceGroups(stageId)**» wird aktiv auf das oben injizierte Repository zugegriffen und der Methodenaufruf asynchron gestartet. Erst, wenn ein Resultat vorhanden ist, wird es mit «**thenApplyAsync()**» ausgewertet und als Antwort im JSON Format an den Anfragensteller zurückgesendet.

```
1 @ApiOperation(value = "get all racegroups of a stage", response = RaceGroup.class, responseContainer = "List")
2 public CompletionStage<Result> getAllRaceGroups(long stageId) {
3     return cache.getOrElseUpdate("racegroups/stages/"+stageId, () -> raceGroupRepository.getAllRaceGroups(stageId)
4         .thenApplyAsync(raceGroups -> ok(toJson(raceGroups.collect(Collectors.toList())))).exceptionally(ex -> {
5         Result res;
6         if(ExceptionUtils.getRootCause(ex).getClass().getSimpleName().equals(GlobalConstants.INDEXOUTOFBOUNDEXCEPTION)){
7             res = badRequest("No racegroups are set in DB for this stage.");
8         } else {
9             res = internalServerError(ex.getMessage());
10        }
11        return res;
12    })), GlobalConstants.CACHE_DURATION);
13 }
```

Abbildung 51: RaceGroup Controller GET Route

Im Detail wird im RaceGroupRepository ein JPA Befehl ausgeführt, der auf SQL basiert. Auf dieser Ebene ist es möglich, SQL Befehle an die Postgres Datenbank abzusetzen und vorhandene Werte mit den gewünschten Filtern auszulesen.

```
1 @Override
2 public CompletionStage<Stream<RaceGroup>> getAllRaceGroups(long stageId) {
3     return supplyAsync(() -> wrap (entityManager -> getAllRaceGroups(entityManager, stageId)), databaseExecutionContext);
4 }
5
6 private Stream<RaceGroup> getAllRaceGroups(EntityManager em, long stageId){
7     TypedQuery<RaceGroup> query = em.createQuery("select rG from RaceGroup rG where rG.stage.id = :stageId" , RaceGroup.class);
8     query.setParameter(STAGE_ID, stageId);
9     return query.getResultList().stream();
10 }
```

Abbildung 52: RaceGroup Repository

Eine Entität ist wie folgt aufgebaut: @Entity identifiziert Sie als Entität in der Datenbank, der SequeuneGenerator erlaubt es eigene Primärschlüssel für die Entität zu definieren, @ApiModel erlaubt das publizieren für das Swagger-File.

Neben dem automatisch generierten Schlüssel für jede Entität ist es hier auch möglich andere Verbindungen zu anderen Entitäten anzugeben. So ist beispielsweise die Verbindung «ManyToOne» zu einer Etappe angegeben. Dies bedeutet, dass eine Etappe mehrere Renngruppen beinhalten kann. Die weiteren Felder wie «JsonIgnore» und «ApiModelProperty(hidden=true)» ermöglichen es das Bekanntgeben dieser Verbindung nach ausen zu verhindern.

```
1 @Entity
2 @SequenceGenerator(name = "key_gen_RaceGroup", sequenceName = "key_gen_RaceGroup", initialValue = 1)
3 @ApiModel(value = "Racegroup", description="Model of racegroup")
4 public class RaceGroup {
5     @Id
6     @GeneratedValue(strategy=GenerationType.AUTO, generator = "key_gen_RaceGroup")
7     private Long id;
8     ...
9
10    @ManyToOne(cascade=CascadeType.MERGE)
11    @JsonIgnore
12    @ApiModelProperty(hidden=true)
13    private Stage stage;
14    ...
15 }
16
```

Abbildung 53: RaceGroup Entity

6.2.8 Caching konkret

Betrachten wir nochmals näher die GET-Route wird eine Methode «cache.getOrElseUpdate» ersichtlich. Mit dieser Methode wird ein interner Cache mit dem Resultat der Abfrage gefüllt. Ereignet sich ein nochmaliger Request auf dieselbe Route innerhalb der «CACHE_DURATION» (in Sekunden) werden die Resultate aus dem Cache gelesen und die Abfrage auf das Repository wird nicht nochmals ausgeführt. Dies steigert die Performance der API erheblich und ermöglicht so noch mehr Nutzer gleichzeitig zu bedienen.

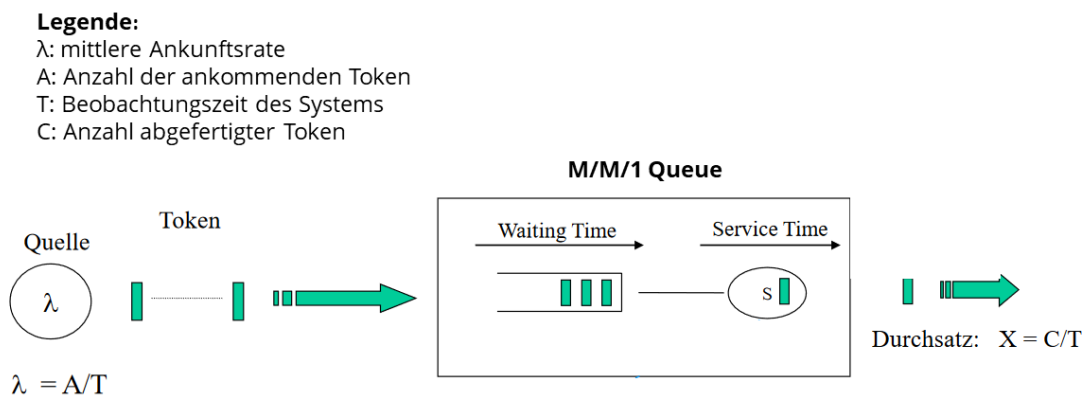
```
1 @ApiOperation(value = "get all racegroups of a stage", response = RaceGroup.class, responseContainer = "List")
2 public CompletionStage<Result> getAllRaceGroups(long stageId) {
3     return cache.getOrElseUpdate("racegroups/stages/"+stageId, () -> raceGroupRepository.getAllRaceGroups(stageId)
4         .thenApplyAsync(raceGroups -> ok(toJson(raceGroups.collect(Collectors.toList())))).exceptionally(ex -> {
5         Result res;
6         if(ExceptionUtils.getRootCause(ex).getClass().getSimpleName().equals(GlobalConstants.INDEXOUTOFBOUNDEXCEPTION)){
7             res = badRequest("No racegroups are set in DB for this stage.");
8         } else {
9             res = internalServerError(ex.getMessage());
10        }
11        return res;
12    })), GlobalConstants.CACHE_DURATION);
13 }
```

Abbildung 54: RaceGroupController GET Caching

6.3 RadioTour Anwendung (Android App)

Die wichtigste Änderung an der bereits vorhanden «RadioTour Anwendung» in der Bachelorarbeit ist die Einführung einer Warteschlange für Updates, die an die API gesendet werden sollen. Dabei ist es möglich, einem «Handler Thread» Nachrichten zu übergeben, dieser kümmert sich, sobald alle Bedingungen (Kapazität, Internetverbindung) erfüllt sind, darum die Nachrichten an die API weiterzuleiten.

Sobald Änderungen anstehen, die an die API weitergeleitet werden sollen, ist ein Aufruf mit den entsprechenden Daten möglich. Diese werden sobald als möglich an die API mittels einer Nachrichten-Library von Android weitergeleitet. Speziell zu erwähnen ist hier, dass unsere Applikation über einen Demo-Modus verfügt, bei dem die Daten nicht aktiv an die API gesendet werden.



Quelle:
Vorlesungsfolie System Modeling and Simulation
Modellierung des zeitlichen Verhaltens mit Warteschlangen, Folie 6
Prof. Dr. Andreas Rinkel
06.06.2018

Abbildung 55: M/M/1 Queue

Oben dargestellt ist der grundlegende Aufbau einer Message Queue wie wir sie in der Anwendung benutzen. Unsere Applikation arbeitet nach dem FIFO Prinzip, das heisst die erste Nachricht, welche ankommt, wird auch zuerst verarbeitet, bzw. an den TourLive Server weitergeleitet. Da es für unsere Arbeit nicht erheblich ist die genauen Kennzahlen unseres Systems zu bestimmen haben wir darauf verzichtet. Der Grund dafür liegt darin, dass unsere Nachrichten asynchron verschickt werden, sobald die Internetverbindung es zulässt und genügend Kapazität dafür vorhanden ist. Prinzipiell allerdings ist es möglich die Anzahl ankommender Nachrichten über die Zeit zu bestimmen, um so darauf zu schliessen, ob die Auslastung unsere Applikation zu hoch ist und den effektiven Durchsatz zu bestimmen.

7 Testing

Um unsere geforderten Leistungen an den TourLive Server zu gewährleisten haben wir von cnlab die Möglichkeit erhalten automatisierte Lasttests durchzuführen. Ein Lasttest simuliert Nutzer einer Anwendung und kann somit Daten über das Verhalten der Anwendung sammeln. In den von Lukas Frey, ein Mitarbeiter von cnlab, durchgeführten Lasttests wurde wie folgt vorgegangen:

Erstmalig sind alle möglichen Links unserer Zuschauer Anwendung durchgeklickt und die dabei referenzierten URL-Adressen aufgenommen worden. Für jede einzelne Seite ist ein separater Lasttest durchgeführt worden. Dabei sind alle URL-Adressen, die von dieser spezifischen Seite referenziert werden zyklisch angesprochen worden.

7.1 Ausgangslage für den 1. Lasttest

Ein Test dauert in etwa 20 Minuten, in diesem Test wird mit einer steigenden Anzahl von «simulierten» Nutzern die Anfragen an die oben definierten URLs abgesetzt. Die Anzahl der Nutzer steigert sich von 50 bis zu 350 gleichzeitig aktiven «simulierten» Nutzer. Dabei werden für jeden Testlauf detaillierte Daten über erfolgreiche, abgebrochene und fehlgeschlagene Anfragen aufgenommen. Zusätzlich werden auch Daten über das zeitliche Verhalten gesammelt und grafisch aufbereitet.

7.2 Lasttest 1, 24.05.2018

Für die Entwicklung der Bachelorarbeit ist uns von der HSR ein virtueller Server zur Verfügung gestellt worden. Die Grundkonfigurationen dieses Servers sind folgende: 1 vCPU, 2GB RAM und 15GB Storage

Für den durchgeführten Lasttest ergaben sich folgende Ergebnisse:

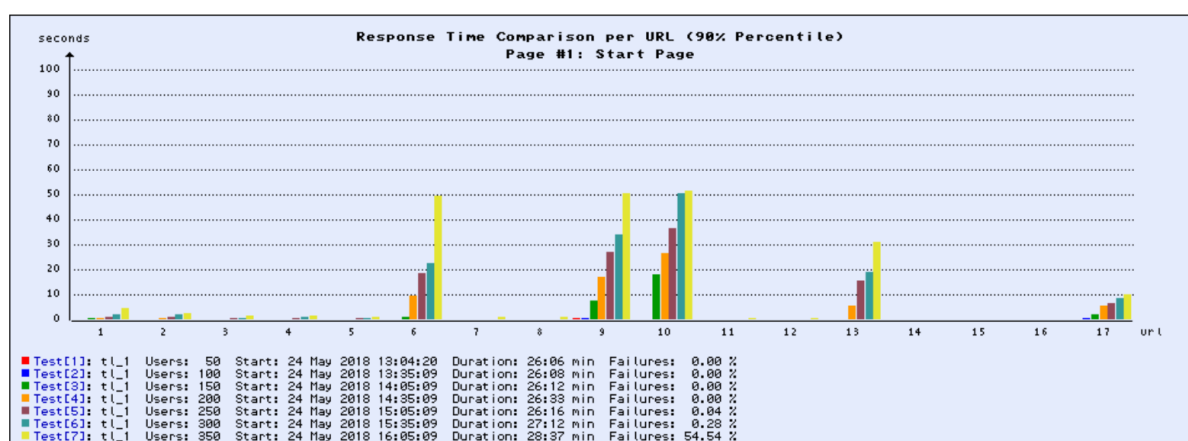


Abbildung 56: Lasttest 1

Daraus lässt sich schliessen, dass die Routen 6, 9, 10, 13 & 17 genauer untersucht werden müssen, da dort die Antwortzeit bei 350 gleichzeitigen Benutzern zum Teil 50 Sekunden betrug.

URL	Request
6	GET https://prod-api.tourlive.ch:443/settings
9	GET https://prod-api.tourlive.ch:443/racegroups/stages/6
10	GET https://prod-api.tourlive.ch:443/riderstageconnections/stages/6
13	GET https://prod-api.tourlive.ch:443/judgmentriderconnections/stages/6
17	GET https://tlng.cnlab.ch:443/wo/cars.json

Tabelle 11: Request mit Optimierungspotential

Fazit

Bei der URL Nummer 6 war es für uns möglich die schlechte Performance darauf zurückzuführen, dass das Caching für diese Route vergessen gegangen ist. Bei den Routen 9, 10 und 13 konnten in der Implementation keine Fehler festgestellt werden. Die URL Nummer 17 ist eine externe Abhängigkeit, die nicht von unserem TourLive Server aus beeinflussbar ist. Während des Lasttests haben wir aktiv unseren TourLive Server beobachtet und festgestellt, dass die CPU Auslastung konstant bei 100% lag und das RAM für die Applikationen nicht ausreichte. Ausserdem wurde uns bewusst, dass wir für unsere in Docker Container laufenden Applikationen ihren maximalen Speicherbedarf nicht begrenzt haben. Somit haben sich die Applikationen bei hoher Last gegenseitig die Ressourcen weggeschnappt was zu einer schlechten Performance auf Routen mit vielen Daten geführt hat.

7.3 Ausgangslage für den 2. Lasttest

Die Testbedingung für den 2. Lasttest sind bis auf einen entscheidenden Parameter identisch geblieben. Bei diesem Parameter handelt es sich um die Anzahl der «simulierten» Nutzer. Die Anzahl Nutzer steigerte sich in diesem Test im Bereich von 50 bis zu 1'000. Zudem wurden die Erkenntnisse aus dem 1. Lasttest eingebaut. Konkret bedeutet das, auf der TourLive API wurde für die Route Nummer 6 das Caching eingebaut. Da wir die anderen Routen nicht direkt aus dem Code beeinflussen konnten, haben wir uns dazu entschieden, unsere Serverkapazität anzupassen. Die Serverkapazität wurde auf neu 4vCPus, 8GB RAM und 15 GB Storage erhöht. Ausserdem sind die Docker Container aktiv begrenzt worden. So stehen für den TourLive API Container 3 GB RAM, für den Tourlive Web Container 2 GB RAM, für den TourLive Admin Container 1 GB RAM und für den NGINX 1 GB RAM zur Verfügung.

7.4 Lasttest 2, 04.06.2018

Für den 2. Lasttest ergaben sich folgende Ergebnisse:

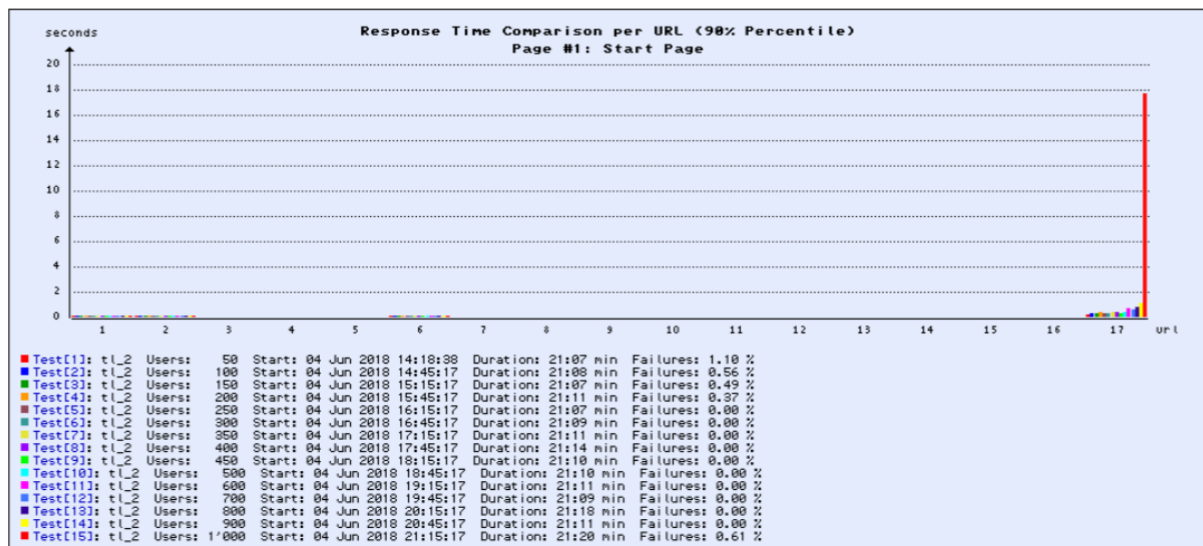


Abbildung 57: Lasttest 2

Fazit

Dank unseren Konfigurationen und der Erhöhung der Server Kapazität sind wir nun in der Lage für bis zu 1'000 Nutzer eine Sub-Second Responsezeit zu garantieren. Die Problemrouten Nr. 6, 9, 10 und 13 sind durch unsere Anpassungen beachtlich verbessert worden. Durch die richtige Konfiguration konnte die Antwortzeit von 50 Sekunden bei 350 Nutzern auf nahezu die Round Trip Time reduziert werden. Somit wird sich beim Einsatz des Tour-Live Servers an der Tour de Suisse 2018 zeigen, ob er der Anfragenflut der Zuschauer standhalten kann.

8 Resultate

8.1 Resultat

Während der Projektphase ist es uns gelungen ein lauffähiges System zu entwickeln, dass alle seine Umsysteme berücksichtigt und mit diesen interagiert. Durch die Einbindung der «RadioTour-Anwendung» in das neue TourLive-System sind aktuelle Daten von laufenden Rennen nun auch für die Öffentlichkeit zur Verfügung gestellt. Eingaben die während einem Rennen passieren werden übermittelt, persistiert und aufbereitet. So ist es uns gelungen eine Web-Anwendung für interessierte Zuschauer zu kreieren, welche diese auf den aktuellsten Stand des Renngeschehens bringt. Nebst aktuellen Gruppenpositionen auf dem Strecken- und Höhenprofil wird dem Benutzer auch detailliert gezeigt welcher Fahrer sich in welcher Gruppe befindet, welche Wertungen vergeben wurden, wie sich das Klassement aktuell zusammensetzt und wer nach den laufenden Daten der neue Leader werden könnte.

Die ganze Umgebung wird aktiv an der Tour de Suisse 2018 eingesetzt und ist unter dem Link: www.tourlive.ch erreichbar.

Am Ende jedes Tages werden die persistierten Daten durch Dateien der offiziellen Zeitmesser auf den neusten Stand gebracht. Um die Handhabung dieser Aktualisierungen zu vereinfachen und auch im Allgemeinen die Datenbank des TourLive Servers einfacher zu verwalten ist während der Bachelorarbeit eine Admin-Webanwendung entstanden. Diese ermöglicht es die Daten einfach per Drag & Drop der Dateien zu aktualisieren und auch der Schritt des initialen Datenimports ist mit einem Klick realisierbar.

8.2 Offene Punkte

In diesem Abschnitt wird auf noch offene umzusetzende Punkte der TourLive Umgebung eingegangen. Dabei wird pro unabhängigem System ein neuer Abschnitt für die genauere Erläuterung verwendet.

8.2.1 Zuschauer-Webanwendung

- Darstellung der Renngruppen und Wertungen auf dem Höhenprofil
- Manuelles wechseln der Daten zwischen verschiedenen Radrennen und Etappen ermöglichen
- Rangberechnung von einzelnen Fahrern in Renngruppen zusätzlich darstellen
- Abstandszeiten von Renngruppen in Karten mit einblenden
- Klassemente-Sortierungsalgorithmus überarbeiten

8.2.2 TourLive API

- Verfeinerung der Daten-Persistierung und Auswertung der gesendeten Daten von der «RadioTour Anwendung»

8.2.3 Admin-Webanwendung

- Direktes Verwalten der Datenbank ermöglichen (CRUD Operationen auf den einzelnen Datenobjekten)
- Import der initialen Daten loslösen vom cnlab-Server, direktes Einfügen der Objekte über die Anwendung ermöglichen

8.2.4 Interaktionsanwendung

- Entwicklung einer Interaktionsanwendung für Zuschauer welche sie in das Renn-geschehen mit einbindet. Z.B. durch ein «tumb-up» für bestimmte Fahrer welche sie mögen und Darstellung dieses Ratings auf der Zuschauer-Anwendung

8.2.5 RadioTour Anwendung

- Probleme beim Verwalten grossen Gruppen beheben
- Trikotzeitenberechnung (virtual Gap) überarbeiten
- Zusammenführen von Wertungen am gleichen Rennkilometer

8.3 Ausblick

Nach dem ersten Einsatz an der Tour de Suisse 2018 sind wir davon überzeugt, dass noch viele weitere Gesichtspunkte zum Vorschein kommen werden die es zu implementieren gilt. Leider war die Zeit für die Bachelorarbeit zu begrenzt um den ganzen Scope umzusetzen und viele Details sind erst während des Live Einsatzes in der letzten Woche der Bachelorarbeit zum Vorschein gekommen. Wir würden uns darüber freuen, wenn unser Projekt von zukünftigen Studenten weiterverfolgt und verbessert wird.

Nichtsdestotrotz ist das System so wie es jetzt ist robust und einsatzfähig. Des Weiteren wurde das System so aufgebaut, dass eine Erweiterung einfach möglich ist. Wir sind dankbar dieses Projekt als Bachelorarbeit umgesetzt zu haben und würden uns freuen unsere Anwendung im nächsten Jahr an der Tour de Suisse 2019 wieder anzutreffen.

I. Anhang

9 Projektmanagement

9.1 Projektmitglieder

Name	Funktion / Beschreibung
Prof. Dr. Peter Heinzmann	Verantwortlicher Dozent der Bachelorarbeit, Betreuer der Arbeit, Vertreter der Interessen vom Radio-Tour Speaker Steve Bovay und der «Tour de Suisse»
Patrick Eichler	Mitbetreuer der Bachelorarbeit, Mitarbeiter von cnlab und Betreuer der Anwendungen rund um die Tour de Suisse sowie Gippingen.
Dominik Good	Student an der HSR, Hauptverantwortlich für das Backend sowie die RadioTour Anwendung
Urs Forrer	Student an der HSR, Hauptverantwortlich für das Frontend sowie die Infrastruktur

Tabelle 12: Projektmitglieder

9.2 Vorgehen

Für das grundsätzliche Vorgehen im Projekt wurde das Vorgehensmodell RUP (Rational Unified Process) gewählt. Es unterteilt das Projekt in vier Abschnitte:

- **Inception** | Einarbeitung / Erarbeitung von Konzepten
- **Elaboartion** | Entwerfen von möglichen Lösungen
- **Construction** | Umsetzung
- **Transistion** | Übergang

Jeder der oben genannten Abschnitte bzw. Phasen ist in eine oder mehrere Phasenabschnitte unterteilt und wird entsprechend mit einem Meilenstein versehen.

Das genaue Vorgehen der Bachelorarbeit wurde an das Vorgehensmodell RUP angelehnt und an unseren Bedürfnissen etwas angepasst. Neben den fix zugewiesen Arbeiten pro Phase bzw. Iteration (zu sehen in Kapitel 9.3) werden alle weiteren Arbeiten/Fehler/Anforderungsänderungen dynamisch zugeteilt. Jeweils zu Beginn einer Iteration werden diese gesammelt, daraus Arbeitspakete erstellt und dann einer Person zugewiesen. Die Iterationen haben eine Länge von einer Woche. An den wöchentlichen Sitzungen am Dienstagnachmittag finden zudem Reviews statt, bei welchem der aktuelle Stand besprochen wird.

9.3 Phasen

Aufgeteilt in Phasen und Iterationen sieht das Projekt wie folgt aus.



Inception / Refinement I

Start am 19. Februar 2018

Ende am 27. Februar 2018

Beschreibung

Kickoff-Meeting, Projektplanung Bachelorarbeit, Projektaufbau (Gemeinsame Datenablage, Projektmanagementtool), Überarbeitung SA, Optimierungen RadioTour App, Risikoanalyse



Elaboration I

Start am 27. Februar 2018

Ende am 06. März 2018

Beschreibung

Definition von Wertungen, Use Cases (im Brief-Format), Leistungsanforderungen, Nicht funktionale Anforderungen, Studium «Serverlösungen»

Elaboration II

Start am 06. März 2018

Ende am 13. März 2018

Beschreibung

Domainanalyse, Evaluierung des Technologie Stacks, Analyse von Web-Hosting Lösungen, Einarbeitung in die Technologien, Implementierung der Wertungen

Elaboration III

Start am 13. März 2018

Ende am 20. März 2018

Beschreibung

UI-Design (Prototype), Entwurf einer Gesamtarchitektur, Einarbeitung in die Technologien, Architekturprototyp mit Frontend und Backend, Entwicklungsumgebung einrichten

Construction

Construction I

Start am 20. März 2018
Ende am 27. März 2018

Beschreibung

Aufbau der Infrastruktur, Deployment, Datenmodell, Backup, Implementierung Backend API (Controllers und Repos)

Construction II

Start am 27. März 2018
Ende am 03. April 2018

Beschreibung

Reverse Proxy, Abschluss der Arbeiten an der Infrastruktur, Swagger Implementation auf der API, Arbeiten am Backend, Import von der cnlab API

Construction III

Start am 03. April 2018
Ende am 10. April 2018

Beschreibung

Fertigstellung des Imports auf der Backend API, Anpassungen Import auf RadioTour App, Implementierung und Testing der Posts von der RadioTour App

Construction IV

Start am 10. April 2018
Ende am 17. April 2018

Beschreibung

Admin-Anwendung, Anbindung der Admin-Anwendung an die API, Sicherung der API, Import der Matsport Daten im Backend, API CleanUp, Vorbereitungen Zwischenpräsentation

Construction V

Start am 17. April 2018
Ende am 24. April 2018

Beschreibung

Zwischenpräsentation, Import GPX-Daten auf der API, Initiale Arbeiten am Frontend (Routing, Einrichten von Redux)

Construction VI

Start am 24. April 2018
Ende am 01. Mai 2018

Beschreibung

Suche in der Web-Anwendung, Testing durch RadioTour Speaker, Trikots in der Web-Anwendung, Importfunktionen auf der Admin-Anwendung erweitern

Construction

Construction VII

Start am 01. Mai 2018

Ende am 08. Mai 2018

Beschreibung

Implementierung von Caching auf der API, Anpassungen aus dem User Testing, Höhenprofil im Frontend, Sortierung im Frontend, Swagger-Dokumentation

Construction VIII

Start am 08. Mai 2018

Ende am 15. Mai 2018

Beschreibung

Styling im Web, Bilder von Fahrer, Benachrichtigungen im Web, periodisches Abrufen der Daten im Web

Construction VIII

Start am 15. Mai 2018

Ende am 22. Mai 2018

Beschreibung

Optimierungen von Caching, HTTPS, Refactoring, Folgearbeiten Benachrichtigungen im Web und an API

Construction X

Start am 22. Mai 2018

Ende am 29. Mai 2018

Beschreibung

Lasttest, Dokumentation, Darstellung verbessern, Refactoring

Construction XI

Start am 29. Mai 2018

Ende am 05. Juni 2018

Beschreibung

Bugfixing, Optimierung nach Lasttest, Dokumentation

Transition

Transition I

Start am 05. Juni 2018

Ende am 15. Juni 2018

Beschreibung

Dokumentation, Bericht, Aufräumarbeiten, Drucken und Binden, Vorbereitungen Ausstellungen, Korrektur, persönlicher Projektbericht

9.4 Zeitplan

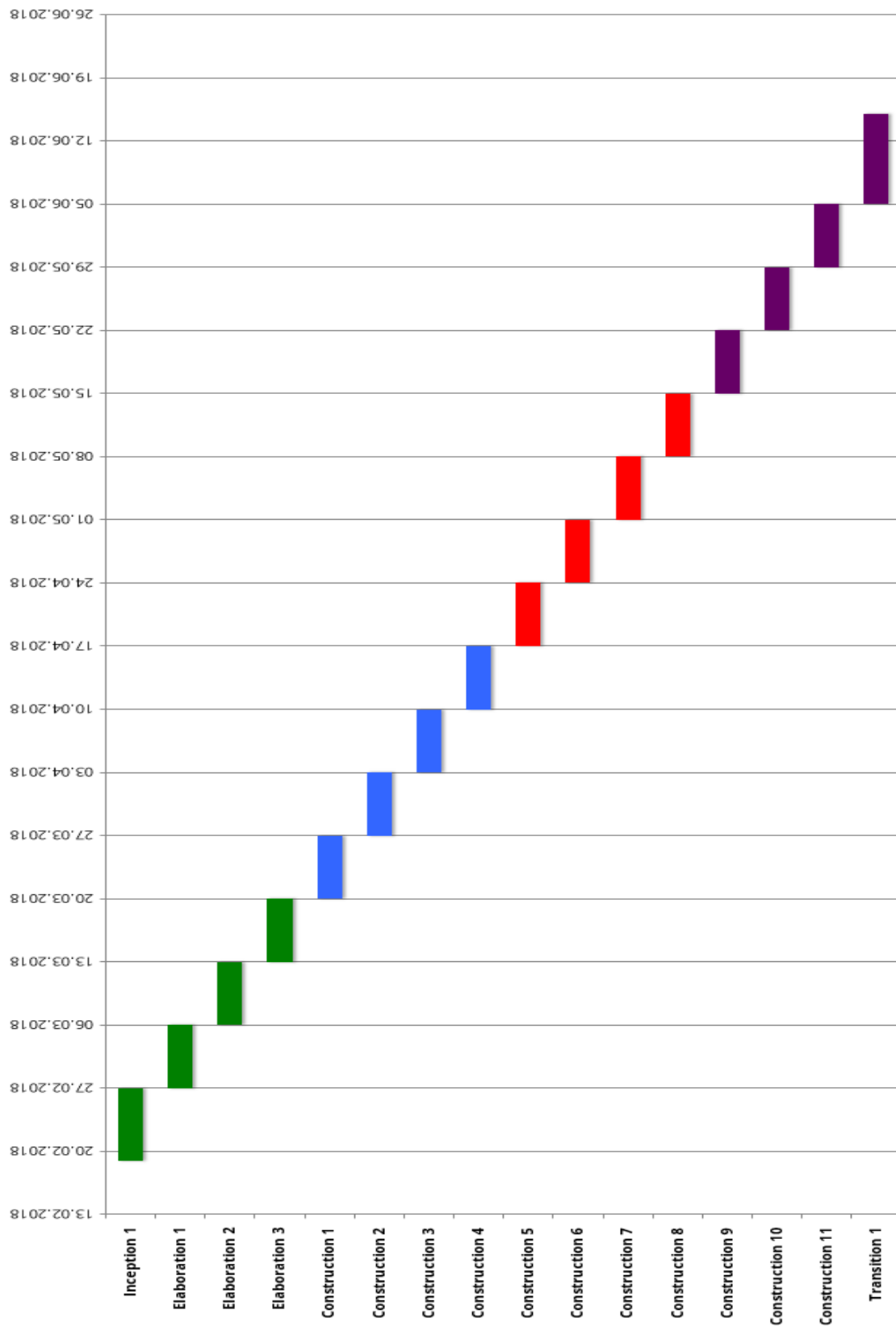


Tabelle 13: Zeitlicher Ablauf

9.5 Releases

Die Releases der Systemteile findet jeweils am Dienstag nach der wöchentlichen Sitzung statt. Zu diesem Zeitpunkt wird der stabile Stand der Development-Umgebung auf die Produktions-Umgebung übertragen. Die Beteiligten am Projekt werden an der Sitzung und ggf. per Mail über entsprechende Erneuerungen informiert. Zudem ist es möglich, dass ausserhalb dieses Rhythmus Releases stattfinden. Der oben erwähnte Rhythmus gilt als Minimum.

9.6 Meilensteine

Nr.	Name	Beschreibung	Datum
MS1	End of Elaboration	Anforderungen sowie die Architektur für die neuen Systeme sind definiert, Android Anwendung wurde performancemässig optimiert.	20.03.2018
MS2	API & Tablet Anwendung funktionieren miteinander	Die API sowie die Tablet Anwendung sind aneinandergebunden und es findet ein Datenaustausch statt.	10.04.2018
MS3	Admin Screen fertig	Die Admin-Anwendung enthält den geforderten Funktionsumfang und kann eingesetzt werden.	17.04.2018
MS4	Webanwendung abgeschlossen	Die Features der Webanwendung wurden umgesetzt und die Webanwendung kann verwendet werden	23.05.2018
MS5	Lasttest durchgeführt	Auf der Applikation wurde ein Lasttest durchgeführt.	25.05.2018
MS6	Feature Freeze	Ab diesem Zeitpunkt dürfen keine neuen Features mehr in die Software eingebaut werden. Bugfixing ist noch möglich.	29.05.2018
MS7	Code Freeze	Software liegt Bug frei vor und Code darf nicht mehr verändert werden.	05.06.2018
MS8	Start der Tour de Suisse	Die Tour de Suisse 2018 startet und die Anwendung wird eingesetzt.	09.06.2018
MS9	Abgabe der Arbeit	Alle Dokumente wurden abgegeben und das Projekt wird abgeschlossen. Präsentation der Arbeit an der Ausstellung	15.06.2018

Tabelle 14: Meilensteine

9.7 Risiken

In der Inception-Phase der Bachelorarbeit wurde die Risiken für die anstehende Arbeit ermittelt. Während des Projekt mussten keine Änderungen an diesen Risiken vorgenommen werden, da die Rahmenbedingungen gleichgeblieben sind. Verbesserungen im Bereich der Wahrscheinlichkeit wurden bewusst nicht vorgenommen, da diese den Eindruck vom Beginn des Projekts verfälschen würden. In den nächsten drei Abschnitten wird alles rund um die Risiken genauer erläutert.

9.7.1 Risikotabelle

Die Tabelle 15 beschreibt und gewichtet die Risiken. Die Werte der Wahrscheinlichkeit liegen zwischen 0 und 100 Prozent, während die Werte für das Gefahrenpotential zwischen 1 und 10 liegen. Der Gesamtwert des Risikos ergibt sich aus der Multiplikation von Wahrscheinlichkeit und Gefahrenpotential.

ID	Risiko	Wahrscheinlichkeit	Gefahrenpotential	Gesamtwert
R1	Einsatz von unbekannten Technologien	50 %	9	4.5
R2	Anforderungsänderungen	60 %	7	4.2
R3	Performancekriterien	70 %	7	4.9
R4	POST's an die API	20 %	4	0.8
R5	Nicht genügend Kenntnisse in den Programmiersprachen	30 %	5	1.5
R6	Qualität der Software nicht ausreichend	30 %	7.5	2.25
R7	Deployment nicht möglich	15 %	4.5	0.675
R8	Zu hohe Komplexität	35 %	10	3.5
R9	Krankheit	5 %	5	0.25
R10	Kompatibilitätsprobleme	40 %	5	2

Tabelle 15: Risikotabelle

Legende zur Risikotabelle

Wert/Farbe	Gesamtwert des Risikos
4 bis ...	Schwer
2.25 bis < 4	Mittel
0 bis < 2.25	Gering

Zu dem zehn Risiken wurden die nachstehenden zehn Massnahmen definiert.

ID	Massnahme
R1	Einarbeitungsphase genau planen, Tutorials lesen
R2	In der Planungsphase Use Cases klar definieren
R3	Von Beginn weg auf Performance achten
R4	Sauberes Testen
R5	Tutorials, notfalls Stack Overflow
R6	Guidelines früh einsetzen und regelmässige Code Reviews durchführen
R7	Während der Elaborationsphase genug Zeit für das Testing und die Umsetzung eines Prototyps der Deployment-Infrastruktur einplanen
R8	Erstellung eines Prototyps, welcher die wichtigsten Funktionen implementiert
R9	Aktiv handeln und sofort neu planen, verlorene Zeit später aufholen
10	Prototyp erstellen für Grundfunktionen

Tabelle 16: Massnahmen zu den Risiken

9.7.2 Umgang mit Risiken / Rückblick

Die grössten Risiken existierten in den Bereichen Anforderungsänderungen sowie Performance. An den wöchentlichen Sitzungen wurden die aktuellen Stände jeweils gezeigt und zum Austesten zur Verfügung gestellt. Durch dieses Vorgehen konnten die Anforderungsänderung früh abgeholt und in Grenzen gehalten werden. Die Wahrscheinlichkeit des Eintretens des Risikos «Einsatz von unbekannten Technologien» wurde durch eine frühe Einarbeitung auf ein Minimum reduziert und bildet somit keine Gefahr mehr.

9.8 Infrastruktur / Entwicklungsumgebung

Während der Bachelorarbeit wurden verschiedene Tools, Konzepte und Infrastruktur eingesetzt. In diesem Kapitel wird genau auf die Themenbereiche eingegangen und diese kurz erläutert.

9.8.1 Software

WebStorm

Die integrierte Entwicklungsumgebung (kurz IDE) «WebStorm» stammt von Jet Brains und ist für die Programmiersprache JavaScript ausgelegt. Die Entwicklungsumgebung kam bei der Programmierung des Frontend sowie den Admin-Panels mit React zum Einsatz.

IntelliJ IDEA

Gleich wie WebStorm stammt auch IntelliJ von Jet Brains. IntelliJ ist aber für den Einsatz der Programmiersprache Java ausgelegt. Zum Einsatz kam es bei der Programmierung der API mit dem Play Framework. Wie bereits erwähnt basiert dieses Framework auf der Programmiersprache Java.

JustInMind

JustInMind ist ein UI-Prototyp Programm und ermöglicht es bereits vorab einen Eindruck der zukünftigen Anwendungen zu geben.

Android Studio

Bei Android Studio handelt es sich um die integrierte Entwicklungsumgebung für die Android Plattform. Es stammt aus dem Hause Google und steht zum Download frei zur Verfügung. Damit wurde im Rahmen der Bachelorarbeit die Android-App „RadioTour Anwendung“ weiterentwickelt.

JIRA

JIRA ist das Projektmanagementprodukt von Atlassian. Dabei handelt es sich um ein kostenpflichtiges Produkt. Es ist für kleine Teams verfügbar und kostet nur 10 Dollar. In der Bachelorarbeit wurde JIRA zur Projektplanung, der Zeitaufschreibung zum Tracking der Arbeit eingesetzt.

GitHub

GitHub, der Open Source Anbieter zum Hosting von Code Repositories. Der gesamte Code der Bachelorarbeit ist darauf abgelegt und ermöglicht ein gleichzeitiges Zusammenarbeiten am Code.

Travis

Travis CI ist eine Open Source Software für die kontinuierliche Integration sowie das kontinuierliche Deployment. Mit GitHub verbunden wird dieser Prozess bei Änderung am Code automatisch ausgelöst.

Sonarqube

Die Qualität des Codes ist äusserst wichtig. Einerseits zur Wahrung der Sicherheit, andererseits zur Reduktion der Komplexität. Je weniger komplex, je angenehmer ist es für einen Entwickler damit zu arbeiten. Mittels einer statischen Code-Analyse überprüft Sonarqube die technische Qualität und stellt eine Auswertung zur Verbesserung des Codes zur Verfügung.

9.8.2 Hardware

Während der Bachelorarbeit wurde hauptsächlich mit der von der HSR zur Verfügung gestellten Hardware (Arbeitsplätze) gearbeitet. Zusätzlich dazu haben wir unsere persönlichen Notebooks für das mobile Arbeiten eingesetzt.

Zum Testen der Android-App «RadioTour Anwendung» wurde das in der Studienarbeit evaluierte Tablet Huawei MediaPad T3 Lite 10 verwendet.

Das Hosting der gesamten Anwendung fand auf einem durch die HSR zur Verfügung gestellten vServer statt. Zu Beginn verfügte der Server für 2 GB und 1vCPU. Für die Tour de Suisse wurde der Server auf 8 GB und 4 vCPU erhöht.

9.9 Qualitätsmassnahmen

Die Anforderungen seitens der Stabilität sind hoch. Während des Rennens sollte die Systeme nicht abstürzen. Ein Absturz einer Komponente des Systems wäre eine Katastrophe. Als weiteren Punkt zu beachten ist die Qualität des Codes selbst. Schlechter Code wird kaum weiterverwendet. Dies soll nicht das Ziel sein. Unser Code soll möglichst die nächsten Jahre im Einsatz sein und auch für die nächsten Entwickler verständlich sein. Um diese Stabilität sowie die Langlebigkeit gewährleisten zu können, wurden diverse Qualitätssicherungsmassnahmen getroffen. Dazu gehören neben dem Testing auch Code Review sowie die Continuous Integration.

9.9.1 Code Reviews

Unter einem Code Review versteht man die Prüfung des programmierten Quellcodes durch eine weitere Person. Solche Review helfen sogenannte «Code Smells» zu finden oder Logikfehler aufzudecken. Weiter helfen die Code Reviews den Code über mehrere Personen hinweg möglichst konsistent zu halten. Diese Code Reviews wurden jeweils bei einem Pull Request durchgeführt. Die genaue Integration in den Arbeitsablauf ist im Kapitel 9.9.3 erläutert.

9.9.2 Testing

Tests dienen dazu die Funktionalität eines Systems oder eine Anwendung zu gewährleisten. Dabei wird diese auf ihre Korrektheit sowie das richtige Verhalten im Fehlerfall geprüft. Auf die Verwendung von Unit Tests wurde bewusst verzichtet. Diese Test Coverage hätte für unser System keinerlei Aussagekraft und wäre durch die vielen Abhängigkeiten zur DB sehr schwer zu realisieren gewesen.

Ein grossen Wert wurde auf die sogenannten Usability-Tests (User Tests) gelegt. Regelmässig wurden die verschiedenen Teile des realisierten Systems von den Betreuern (Prof. Dr. P. Heinzmann sowie P. Eichler) getestet und Rückmeldungen sind in die Entwicklung eingeflossen. Zudem haben weitere User Tests mit dem Hauptanwender der RadioTour Anwendung (Android App) stattgefunden. Mittels all diesem Feedback ist es uns gelungen eine für die Benutzer optimierte Anwendungen zu entwickeln.

9.9.3 Continuous Integration

Um die Qualität des Produkts fortlaufen testen zu können, ist es nötig, dass die Anwendung(en) auch fortlaufen herausgegeben werden. Genau dieses Prinzip nennt man Continuous Integration. Durch die Einhaltung dieses Prinzip war es uns möglich den neuprogrammierten Code möglichst schnell zum Testen zu Verfügung zu stellen.

Dafür wurde eigens eine Entwicklungsumgebung neben der Produktionsumgebung aufgezogen, welche ein frühes Testing der Software ermöglicht und auch einen stabilen Stand zur Verfügung stellt.

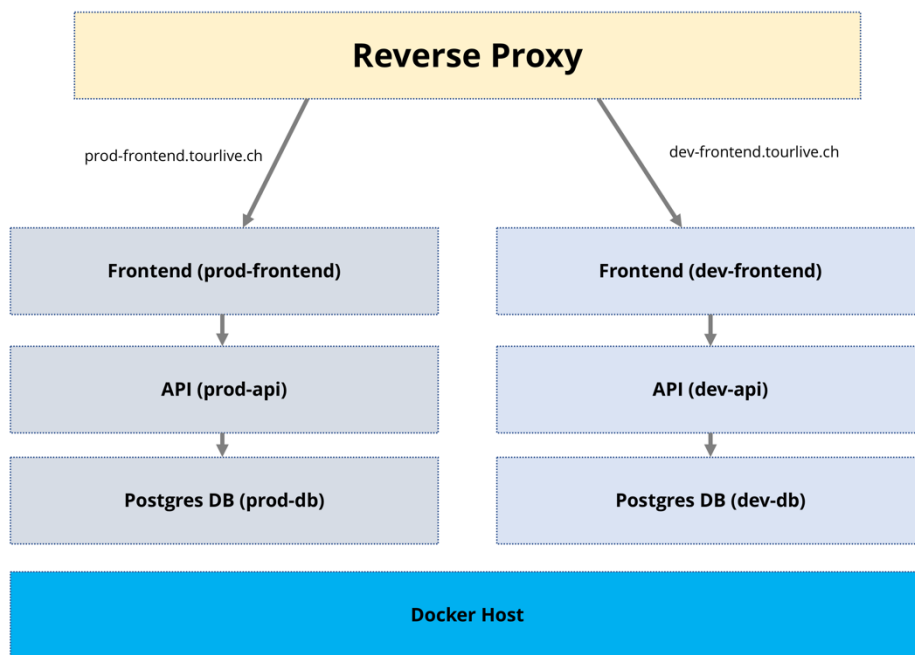


Abbildung 58: Aufteilung DEV/PROD mit Reverse Proxy

Eine Funktionserweiterung erforderte jeweils die Erstellung eines «Feature Branches». Eine Entwicklung direkt auf dem Master wurde aktiv blockiert. Sobald die Implementation der Erweiterung abgeschlossen ist, wird ein sogenannter Pull Request erstellt. Dieser Request dient zur Verschmelzung des Codes auf dem «Master-Branch» mit den neuen Funktionen. Beim Erstellen eines «Pull Request» findet im Hintergrund eine statische Code-Analyse durch Travis CI statt. Die Code-Analyse klärt den Entwickler über allfälliges Verbesserungspotential auf. Zudem wird im Hintergrund die Software erstmals paketiert und auf Fehler geprüft. Das Zusammenführen des «Feature Branch» und des «Masters» erfordert zum einen das erfolgreiche Durchlaufen des Builds in Travis sowie der in Kapitel 9.9.1 erwähnte Code Review. Mittels der Zusammenführung («Mergen») wird automatisch die neue Version der Software auf der Entwicklungsumgebung (dev-....tourlive.ch) zur Verfügung gestellt und ein Testing ist möglich.

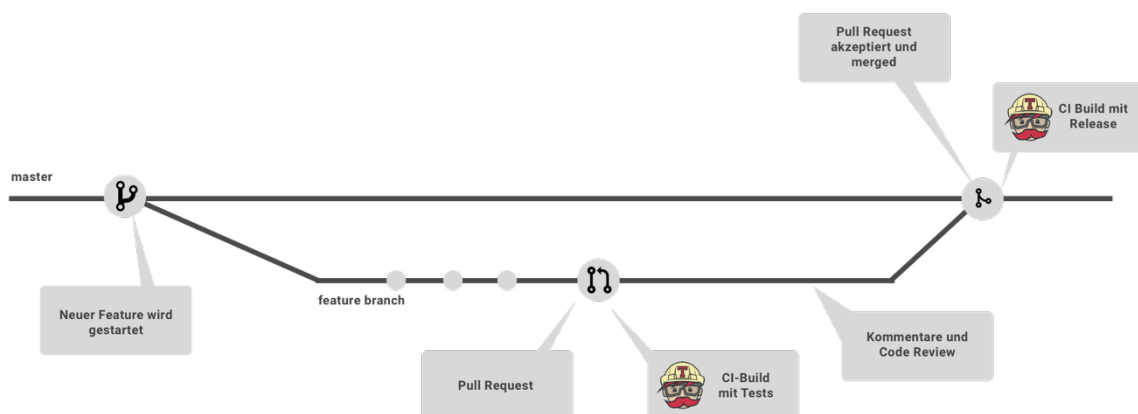


Abbildung 59: Continuous Integration Workflow

10 Persönliche Berichte

10.1 Urs Forrer

Wie bereits die Studienarbeit habe ich auch die Bachelorarbeit mit Dominik Good zusammen gemacht. Für uns war das wichtig, da wir uns sicher waren, dass wir sehr gut miteinander arbeiten und harmonieren.

Das bestehende Team sowie die Tatsache, dass es sich bei der Bachelorarbeit um eine Folgearbeit zur Studienarbeit handelt, ermöglichte uns einen einfachen Einstieg in diese Arbeit. Es resultiert darin, dass wir bereits nach 4 Wochen mit der Programmierung begonnen haben. Wir waren im Gegensatz zur Studienarbeit viel mehr mit dem Thema «Radrennen» bewandert und könnten dieses Wissen entsprechend einsetzen. Mit dem Ziel des Startes der Tour de Suisse am 09. Juni 2018 vor den Augen war wir immer voller Motivation. Es gibt nichts besser, als zu wissen, dass das Produkt wirklich eingesetzt wird und man dies während der Arbeit noch miterlebt.

Der Start in die Umsetzungsphase verlief etwas harzig. Mit React und Redux haben wir Technologien eingesetzt, welche mir zuvor nur am Rande bekannt waren. So habe ich die ersten Stunden mit dem Einlesen ins Thema und Beispielen beschäftigt. Relativ schnell trugen diese Früchte und wir konnten die geforderte Funktionalität umsetzen. Dagegen hatte ich auf der Infrastrukturseite keine Probleme. Docker war für mich eine bekannte Technologie und die nötigen Systemkenntnisse habe ich mir bereits während der Berufslehre angeeignet. Die Unwissenheit über korrekte Programmierung von React führte daher dazu, dass wir etwa in der Mitte der Umsetzungsphase fast den kompletten Code auf den Kopf gestellt haben um den Best-Practices zu entsprechen und uns das Leben einfacher zu machen.

Die letzten paar Wochen waren dann etwas stressig. Der Termin der Tour de Suisse rückte immer näher und letzte Bug Fixes sollten auch noch gemacht sein. Zudem galt es auch noch die Dokumentation zu schreiben, welche während der Entwicklung etwas vernachlässigt wurde. Dennoch haben wir alles unter einen Hut gebracht und konnten die Arbeit gut abschliessen. Zudem kam die Erleichterung beim zweiten Lasttest, als wir erfahren haben, dass unsere Applikation problemlos die mehreren hundert Benutzer tragen kann.

Persönlich nehme ich aus diesem Projekt ganz viele Eindrücke mit für meine Zukunft. Dies reicht vom fundierten Wissen in der API und React Programmierung bis hin zum Entwurf von Applikationen, welche nicht nur aus einem Teil bestehen. Zudem hat es mir gezeigt, wie gut und hilfreich ein richtig aufgesetztes CI (Continuous Integreation) sein kann. Es nimmt einmal viel Arbeit ab und man kann sich auf die Hauptarbeit konzentrieren.

Rückblickend sehe ich dieses Projekt mit einem weinenden sowie einem lachenden Auge an. Einerseits ist die Bachelorarbeit das Ende des Studiums. Andererseits habe ich während dieser Arbeit viel Neues dazugelernt, welches ich in meinem Berufsalltag hoffentlich gebrauchen kann.

10.2 Dominik Good

Bereits in der Studienarbeit «RadioTour Anwendung» durfte ich erste Erfahrungen mit dem mir bis dato noch unbekannten Thema Radrennen sammeln. Umso mehr hat es mich gefreut, dass wir die von uns entwickelte Anwendung und die darin enthaltenen Informationen noch weiterführend in anderer Form verarbeiten und für interessierte Zuschauer bereitstellen durften.

Für mich persönlich war dieses Projekt eine Bereicherung in Sachen Wissen, Zusammenarbeit und agiler Softwareentwicklung. Da ich bereits mit meinem Partner Urs Forrer die Studienarbeit bestritten habe und die dortige Zusammenarbeit bestens funktioniert hat haben wir uns dazu entschieden auch die Bachelorarbeit gemeinsam fortzuführen.

Einen wichtigen Punkt aus der Studienarbeit, die saubere Analysephase, haben wir in diesem Projekt wieder übernommen. Durch die strukturierte Planung war uns immer klar wann, was zu tun ist und wir konnten flexibel auf Änderungswünsche während des Projektes eingehen und diese auch zeitnahe umsetzen.

Da wir RUP mit zusätzlicher Agilität als Projektmanagementgrundlage eingesetzt haben war es für uns wichtig die Sprintdauer nicht zu lange zu machen. Deshalb haben wir uns entschieden diese Dauer auf eine Woche zu beschränken umso den Funktionsumfang zu begrenzen und jede Woche neue Änderungen präsentieren zu können. Aus der Erfahrung in dieser Arbeit kann ich für meine Zukunft mitnehmen, dass eine kurze Sprintdauer sehr effektiv ist und auf Kundenwünsche viel flexibler reagiert werden kann.

Für mich herausfordern in diesem Projekt war der Einsatz einer für mich noch unbekannten Technologie. Für das Frontend haben wir uns entschieden auf React & Redux zu setzen. Diesen Entscheid habe ich keine Sekunde bereut. Ich durfte meinen Wissensschatz in der Softwareentwicklung um diese neue Technologie erweitern und bin von ihr begeistert. Da Urs bereits in einem frühen Stadium den Grundstein für das Projekt gelegt hat war es für mich erheblich einfacher in die ganze Entwicklung einzutauchen und die Abläufe zu verstehen. Auch das Backend Framework war für mich Neuland. Jedoch basierte dieses auf Java und die Einstiegshürde in diesem Bereich war daher um einiges tiefer.

Für die Infrastruktur haben wir uns entschieden Docker einzusetzen. Ich durfte Docker als sehr performantes und flexibles Framework kennenlernen, welches ich auch in Zukunft jedem empfehlen würde. Für meinen zukünftigen Job als Java Entwickler nehme ich viele wichtige Punkte aus dieser Bachelorarbeit mit, was den Einsatz von verschiedenen Tools betrifft. So habe ich jetzt eine gute Übersicht, wie ich ein eigenes Projekt von Grund auf starten und betreiben kann.

Für mich persönlich nehme ich noch weitere Sachen mit, nämlich den Einsatz von spezifischen Technologien wie Caching, Dependency Injection und asynchrone Programmierung was mir mein Wissen enorm bereichert hat.

11 Zeitauswertungen

Dominik Good sowie Urs Forrer haben während dem gesamten Projekt jeweils am Montag, Dienstag sowie am Freitag gemeinsam an der HSR gearbeitet. Daneben haben sie individuell Zeit für die Bachelorarbeit aufgewendet. Dieses Kapitel zeigt ein paar zeitliche Auswertungen.

11.1 Zeitaufwände pro Kalenderwoche

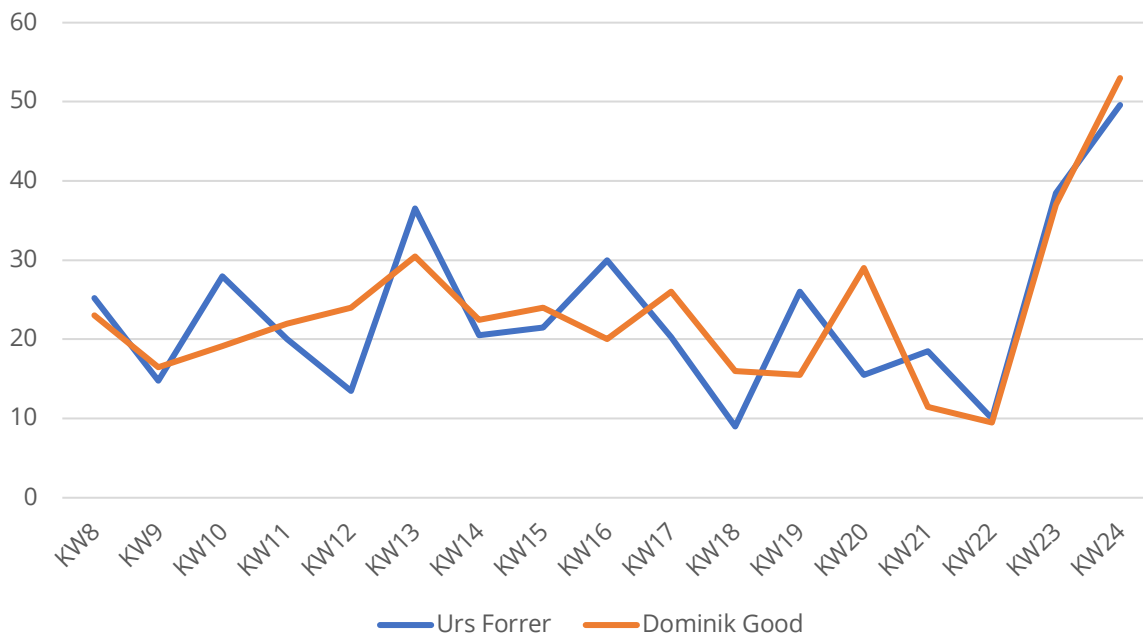


Abbildung 60: Zeitaufwände pro Kalenderwoche

In der Abbildung ist die Verteilung der Aufwände über die Kalenderwochen hinweg sichtbar. Beide haben pro Woche über die gesamte Zeit hinweg etwa gleich viel aufgewendet. Teils sind kleine Unterschiede aufgrund der Verantwortlichen für die jeweiligen Komponenten vorhanden. Das Soll pro Woche liegt rein rechnerisch bei etwa 20 Stunden pro Person. Die vorhandenen Tiefen und Höhen lassen sich wie folgt erklären. In der KW13 ging es erstmalig an die Programmierung und den Prototyp. Die Einarbeitungszeit sowie die Infrastruktur haben hier etwas mehr Zeit in Anspruch genommen. In Kalenderwoche 18 waren nicht alle anwesend, wes wegen sich die Arbeit an der Bachelorarbeit etwas reduziert hat. Bei der Kalenderwoche 22 handelt es sich um die letzte Woche im normalen Semester. Während dieser Woche standen wegen Abgaben andere Fächer im Vordergrund. Die Ausschläge in der KW23 und KW 24 sind einerseits auf die beiden vollen BA-Wochen à 40 Stunden zurück zu führen und andererseits dadurch das die Anwendung ab 09. Juni 2018 an der Tour de Suisse im Einsatz stand und deswegen der eine oder andere Mehraufwand resultierte.

Insgesamt wurden für die Bachelorarbeit rund 796,5 Stunden investiert. Diese Anzahl Stunden liegt über dem Soll von rund 720 Stunden über beide Personen hinweg.

Aufgeteilt auf die einzelnen Personen ergibt sich folgende Zeit.

	Aufgewendete Zeit
Dominik Good	399.2 Stunden
Urs Forrer	397.5 Stunden

Tabelle 17: Gesamtaufwendungen

Die Mehraufwände im Projekt sind hauptsächlich auf allfällige zusätzliche Änderungen zurück zu führen, welche hinblicklich der Tour de Suisse umgesetzt wurden. Des Weiteren nahm die Implementierung der TourLive API etwas mehr Zeit als geplant in Anspruch.

11.2 Zeitaufwände pro Komponente

Bei der Erstellung der Arbeitspakete in JIRA haben wir jedem dieser Pakete ein Komponente hinzugefügt. Dies ermöglicht uns eine Aussage zu treffen, für welche Komponente (also Systembestandteil). wie viel Zeit aufwendet wurde.

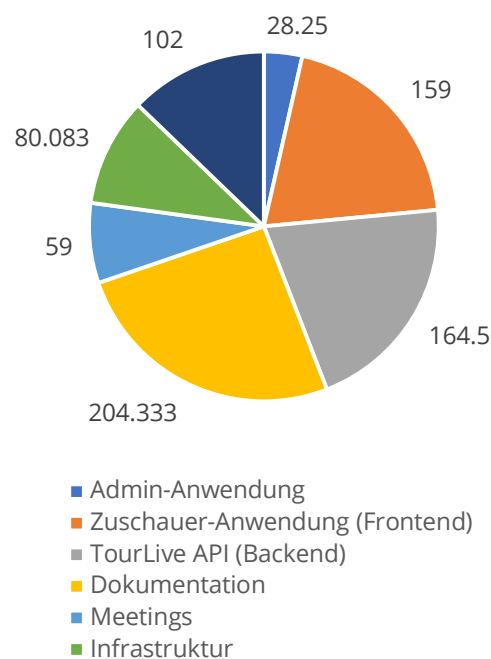


Abbildung 61: Zeitaufwände pro Komponente

Zur Dokumentation zählen sämtliche Arbeiten an irgendwelchen Dokumenten. So auch Evaluationen, die Software Dokumentation oder die Ausarbeitung des technischen Berichts. Die Angaben in der Abbildung 61 sind in Stunden zu verstehen. Die Zahlen entsprechen Erfahrungen aus vergangenen Projekten. So auch dem Engineering Projekt in welchen auch gut ein Viertel der Zeit für die Dokumentation aufgewendet wurde.

11.3 Zeitaufwände pro Phase

Die Abbildung 62 schlüsselt auf, in welcher Phase wie viel Zeit für die Bachelorarbeit aufgewendet wurde. Bei den Angaben zu den Wochen handelt es sich um relative Angaben und nicht um die absoluten Kalenderwochen.

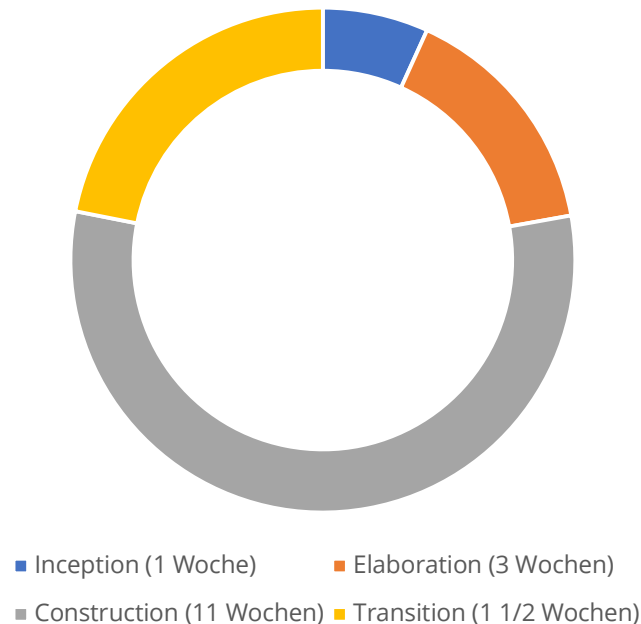


Abbildung 62: Zeitaufwände pro Phase

Im Vergleich zur Studienarbeit hat sich der Trend der Verteilung pro Phase kaum geändert. Bereits da wurde gut zwei Drittel der Zeit für die effektive Entwicklung aufgewendet. In der Bachelorarbeit etwas weniger, dafür kamen mehr Aufwände in der Transition-Phase zu Stande. Dies geschuldet durch die Einführung an der Tour de Suisse 2018. Die Inception- sowie die Elaborationphase sind prozentmässig nahezu im identischen Rahmen wie in der Studienarbeit. Diese Tatsache hat mit der Korrelation der beiden Phase und der Länge des Projektes zu tun.

12 Archivdatei

Folgenden Dateien sind in der zur Bachelorarbeit abgegebenen Archivdatei sowie auf dem USB-Stick vorhanden:

- **01_Protokolle**
- **02_Anforderungen**
 - o Funktionale Anforderungen
 - o Leistungsanforderungen
 - o NFA
 - o Klasselemente und Wertungen
- **03_Evaluation**
 - o Server Lösung
 - o Technologie-Stack
 - o Web Hosting
- **04_Testing**
 - o Usability Test RadioTour Anwendung
- **05_UIDesign**
 - o JustInMind Prototyp
 - o Bilder des Prototyps
- **06_TourLive_Unterlagen**
 - o Technischer Guide TdS 2018
- **07_Architektur**
 - o Software Architektur
 - o TourLive API Domain- und Datenmodell
 - o TourLive API Package Diagramm
 - o RadioTour App Domain- und Datenmodell
 - o TourLive Zuschauer Anwendung Package Diagramm
- **08_Zwischenpräsentation**
- **09_Poster**
- **10_SourceCode**
 - o SourceCode API
 - o SourceCode Zuschauer Anwendung
 - o SourceCode Admin Anwendung

13 Installationsanleitung

In diesem Kapitel wird Schritt für Schritt aufgeführt, wie die entwickelten Systeme auf einem neuen Server deployed werden können. Diese Anleitungen beziehen sich auf die Ubuntu Version 16 oder höher. Für andere Plattformen sind ggf. andere oder weitere Schritte notwendig.

13.1 Requirements

Bevor die Docker-Umgebung aufgesetzt werden kann, ist es nötig diverse Software auf dem Ubuntu-Server zu installieren. Bei dieser Software handelt es sich um Programmiersprache oder Tools zur Unterstützung des Deployments.

13.1.1 GIT

```
sudo apt-get install git
```

13.1.2 Docker CE

Schritte der offiziellen Anleitung ausführen. Zu finden ist die Anleitung unter

<https://docs.docker.com/install/linux/docker-ce/ubuntu/>.

Damit Docker auch von einem nicht-root Benutzer verwendet werden kann, sind die Anweisungen unter <https://docs.docker.com/install/linux/linux-postinstall/> auszuführen. Diese Konfiguration ist optional, empfiehlt sich aber sehr.

13.1.3 Docker Compose

Für die Installation von Docker-Compose hält man sich am besten an die Schritte in der offiziellen Installationsanleitung. Diese Anleitung ist zu finden unter <https://docs.docker.com/compose/install/>.

13.2 Docker-Umgebung

1. Git-Projekt klonen

```
git clone https://github.com/TourLive/infrastructure.git
```

Die Dateien und Konfigurationen sind so ausgelegt, dass der gesamte Inhalt dieses Repos unter dem Pfad «**/opt/tourlive**» abgelegt sein muss. Daher muss dieser Ordner erstellt (falls noch nicht vorhanden) und der Inhalt dorthin kopiert werden.

2. Verzeichnisse für Datenbank anlegen

Damit die Daten der Datenbank auch über die Laufzeit des Datenbank-Containers hinweg persistieren, werden diese Daten ausserhalb des Containers in einem Volume abgelegt. Für die beiden Datenbank-Container müssen daher folgende Verzeichnisse angelegt werden

```
# /data/postgres-prod
# /data/database-dev
```

3. Umgebungsvariablen definieren

Die komplette Konfiguration der Anwendungen basiert auf Umgebungsvariablen. Durch Umgebungsvariablen kann die Umgebung relativ schnell und ohne Änderungen am Code aufgesetzt werden.

In Ubuntu heisst die Datei mit den Umgebungsvariablen **«/etc/environment»**. Die folgenden Umgebungsvariablen gilt es zu setzen. Aus den Beispielen aus der «Abbildung 63: Umgebungsvariablen zur Konfiguration der Container» sollte klar sein, welche Umgebungsvariablen schlussendlich für was verwendet wird.

```
TOURLIVE_DBNAME="tourlive"
TOURLIVE_DBUSER="tourlive"
TOURLIVE_DBPASSWORD="8Z$5C0X54rp1"
TOURLIVE_DBURL="jdbc:postgresql://database-prod/tourlive"
TOURLIVE_API_SECRET="IchBineineApplikationDerTourDeSuisseVonCnlab"
TOURLIVE_DBDEVURL="jdbc:postgresql://database-dev/tourlive"
TOURLIVE_API_PORT=40000
TOURLIVE_HOSTNAME=https://dev-api.tourlive.ch
TOURLIVE_PROD_HOSTNAME=https://prod-api.tourlive.ch
```

Abbildung 63: Umgebungsvariablen zur Konfiguration der Container

4. Anpassung der Hostnamen

Sollen die Anwendung nicht unter den bereits vordefinierten DNS-Namen (prod-frontend.tourlive.ch, ...) laufen, ist eine entsprechende Anpassung in den Docker-Compose Dateien notwendig. Bei den Docker-Compose Dateien handelt es sich um yaml-Dateien. Zu finden sind die Dateien im zuvor geklonten Verzeichnis unter **«/opt/tourlive»**. Angepasst werden müssen die Linien mit traefik.frontend.rule.

```
- proxy
labels:
  - traefik.backend=frontend-dev
  - traefik.frontend.rule=Host:dev-frontend.tourlive.ch
  - traefik.docker.network=proxy
  - traefik.port=80
depends_on:
  - api-dev
```

Abbildung 64: DNS-Namen in der Docker-Compose Datei

5. Umgebung starten /stoppen

Mittels dem Tool docker-compose können die für das System TourLive benötigten Container gestartet werden. Das System ist in Dev und Prod unterteilt. Für den Reverse Proxy und das Admin-Panel sind jeweils separate Konfigurationsdateien vorhanden.

Um die Dev-Umgebung zu starten, ist folgender Befehl notwendig.

```
docker-compose --file dev.yml up -d
```

Für die Prod-Umgebung gilt folgender Befehl.

```
docker-compose --file prod.yml up -d
```

Die Kommandos für die restlichen Docker-Compose Dateien werden nach dem gleichen Prinzip gebildet. Die Datei für das Admin-Panel heisst **admin.yml** und die Datei für den Reverse Proxy ist mit **traefik.yml** benannt.

13.3 Initiale Konfiguration

Bevor das Admin-Panel zur Verwaltung der verschiedenen Anwendungen rund um TourLive verwendet werden kann, muss in der Datenbank zuerst ein Benutzer angelegt werden, mit welchem man sich einloggen kann. In diesem Kapitel sind die nötigen Schritte dazu beschrieben.

Einloggen auf den Server via Putty (nur nötig falls man nicht mehr am Server eingeloggt ist)

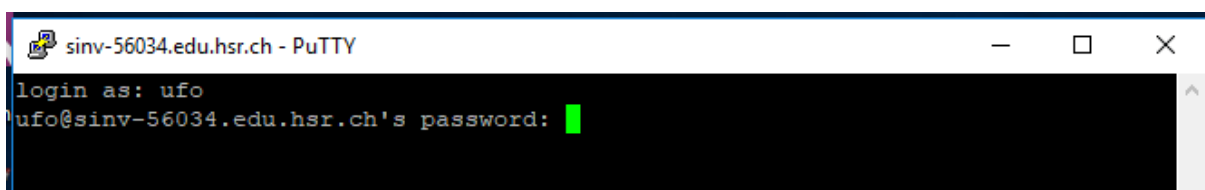


Abbildung 65: Einloggen auf den Server via Putty

Mit `docker ps` alle laufenden Docker-Container anzeigen lassen und die ersten drei Buchstaben der Container ID des Datenbank-Containers ausschreiben.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
0714965791ac	tourlive/frontend:latest	"nginx -g 'daemon of..."	12 hours ago	Up 12 hours	80/tcp	tourlive_frontend-dev_1
8e3d224ac71e	traefik/api:latest	"bin/api"	12 hours ago	Up 12 hours		tourlive_api-dev_1
52e1fcb33bfa	traefik:1.5.4	"/traefik"	13 hours ago	Up 13 hours	0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp, 0.0.0.0:8080->8080/tcp	traefik
4a7669321537	postgres:9.6	"docker-entrypoint.s..."	13 hours ago	Up 13 hours (healthy)	5432/tcp	tourlive_database-dev_1
1ac9b64bba29	tourlive/frontend:stable	"nginx -g 'daemon of..."	14 hours ago	Up 14 hours	80/tcp	tourlive_frontend-prod_1
18dc287b1cd8	tourlive/api:stable	"bin/api"	14 hours ago	Up 14 hours		tourlive_api-prod_1
a10b72772090	postgres:9.6	"docker-entrypoint.s..."	14 hours ago	Up 14 hours (healthy)	5432/tcp	tourlive_database-prod_1

Abbildung 66: docker ps

Mittels dem Kommando `docker exec -it ### /bin/bash` auf den Datenbank-Container verbinden. Die `###` stehen für die ersten drei Buchstaben der ContainerID.

```
ufo@sinv-56034:~$ docker exec -it a10 /bin/bash
root@a10b72772090:/#
```

Abbildung 67: `docker exec` in den Datenbank-Container

Nun **`psql -U tourlive`** eingeben um sich über die PostgreSQL CLI an der Datenbank anzumelden.

```
root@a10b72772090:/# psql -U tourlive
psql (9.6.9)
Type "help" for help.

tourlive=#
```

Abbildung 68: `psql` CLI

Jetzt gilt es einen neuen Eintrag in der Tabelle **`useraccount`** zu machen. Die Formatierung des Kommandos ist in der Abbildung «Abbildung 69: Insert-Query für einen neuen Benutzer» zu finden. Das lange Wort steht für den SHA256 Hashwert des Passwortes. Ein solcher Hashwert kann unter <https://www.hashgenerator.de> generiert werden.

```
tourlive=# INSERT INTO useraccount VALUES (1, '8c6976e5b5410415bde908bd4dee15dfb167a9c873fc4bb8a81f6f2ab448a918', 'admin');
```

Abbildung 69: Insert-Query für einen neuen Benutzer

Über `SELECT * FROM useraccount` prüft man ob der Eintrag korrekt in der Datenbank angelegt wurde.

```
tourlive=# SELECT * FROM useraccount;
 id | password | username
----+-----+-----
  1 | 8c6976e5b5410415bde908bd4dee15dfb167a9c873fc4bb8a81f6f2ab448a918 | admin
(1 row)

tourlive=#
```

Abbildung 70: Überprüfung des neuen Benutzers

14 Benutzeranleitungen

In der folgenden Benutzeranleitung wird spezifisch auf die einzelnen Screens der Admin-Webanwendung eingegangen. Die Funktionsweisen werden genauer erklärt.

14.1 Login / Logout

Beim ersten Aufruf der Admin-Seite wird das Login ersichtlich. Dabei handelt es sich um ein Standard Login, welches wie gewohnt mit Benutzername und Passwort vollzogen wird.

	Login	
---	-------	--

Login

Username


Password

Login

(c) admin.tourlive.ch

Abbildung 71: Admin Webanwendung Login

Nach erfolgreichem Login werden alle verfügbaren Reiter sichtbar. Der letzte Reiter ist das Logout, bei welchem sich der Admin nach Abschluss der Arbeit abmelden kann.

	Home	Einstellungen	Daten in der API	Import von statischen Daten	Import von dynamischen Daten	Logout
---	------	---------------	------------------	-----------------------------	------------------------------	--------

Logout

Um sich aus der Admin Anwendung ausloggen, drücken sie bitte den untenstehen Button

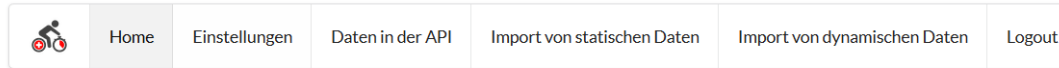
Bis zum nächsten Mal

Logout

Abbildung 72: Admin Webanwendung Logout

14.2 Home

Die Home-Seite dient als Überblick zur Admin-Seite. Dabei werden Kontaktdaten und weitere Informationen dargestellt.



Home

Diese Administrationsanwendung dient zur Verwaltung der Tourlive API, welche im Rahmen des Tourlive Systems verwendet wird.

Kontakt

Patrick Eichler, cnlab AG

Impressum

Diese Administrationsanwendung für die TourLive API wurde von Dominik Good und Urs Forrer 2018 im Rahmen einer Bachelorarbeit an der HSR Hochschule für Technik Rapperswil entwickelt.

Sie wird an der Tour de Suisse 2018 erstmals in den Einsatz kommen und von cnlab betreut und weiterentwickelt werden.

Copyright

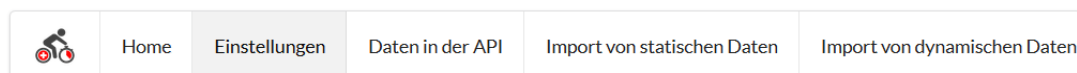
Diese Website und alle dazugehörigen Texte, Grafiken, Logos sowie andere Materialien und das Design sind urheberrechtlich geschützt.

Jegliche Verwendung ohne vorherige schriftliche Zustimmung durch den Webmaster der Seite ist nicht gestattet.

Abbildung 73: Admin Webanwendung Home

14.3 Einstellungen

Im Einstellungsreiter ist es möglich die aktuellen verwendeten Daten für die Zuschauerwebseite und die Radio Tour Anwendung zu setzen. Durch Auswahl des Rennens und der aktuellen Etappe und anschliessendem Klick auf den «Einstellung speichern» Button werden die entsprechenden Konfigurationen in der Tour-Live API hinterlegt.



Einstellungen

Beschreibung

Diese Einstellungen geben an, welche Etappe und Rennen in der Tablet Anwendung (RadioTour Speaker) sowie in Webanwendung (Zuschauer) verwendet wird.

Nach jeder Etappe muss diese Einstellung gewechselt werden.

Aktuelles Rennen

Gippingen 2018

Aktuelle Etappe

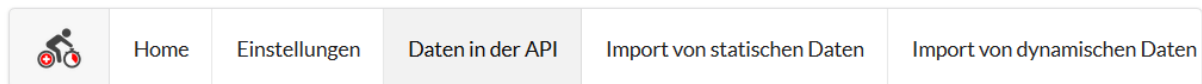
Etappe 1 (Leuggern nach Leuggern, FLATSTAGE)

Einstellungen speichern

Abbildung 74: Admin Webanwendung Einstellungen

14.4 Vorhandene Daten

Um sich einen Überblick über die aktuell verfügbaren Daten im TourLive Server zu schaffen wurde der Reiter «Daten in der API» eingeführt. Dabei werden alle auswählbaren Rennen und Etappen aufgelistet. Um die Kartendarstellung auf der Zuschauerwebseite zu ermöglichen ist es notwendig die entsprechenden GPX Daten zu laden. Für jede Etappe ist ersichtlich, ob diese Daten bereits in der API vorhanden sind oder ob ein Upload noch ausstehend ist.



Daten in der API

Vorhandene Rennen in der API

Folgende Rennen sind auf der TourLive API bereits vorhanden.

 ID 112 | Gippingen 2018

GPX-Daten in der API

Die Hinterlegung der GPX-Daten zu den Etappen der Rennen sieht wie folgt aus:


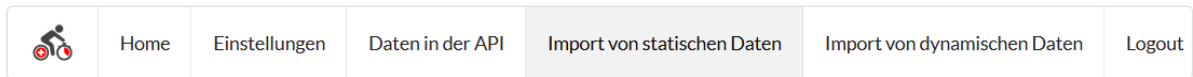
 Gippingen 2018 | Etappe 1 (GPX Daten vorhanden)

Abbildung 75: Admin Webanwendung vorhandene Daten

14.5 Verwalten der statischen Daten

Um ein neues Rennen zu importieren ist es möglich einen Datenimport von der cnlab API zu starten. Dabei wird eine Renn-ID referenziert, welche in der cnlab API gesetzt ist. Mit einem Klick auf den Button «Datenimport von der cnlab API» wird der initiale Import gestartet. Dieser Import ist nur einmalig pro Rennen notwendig.



Import von statischen Daten

Die statischen Daten nur einmal pro Event importiert werden und sind dann jeweils für den gesamten Event gültig.

Import von Renninformationen

Der Import von Renninformationen (Etappe, Fahrer, Trikiots,...) muss nur einmal pro Rennen (z.B. Tour de Suisse 2018) erfolgen.

Unter dem Menüpunkt "Daten in der API" werden die Rennen angezeigt, welche bereits in der API vorhanden sind.

Sollte die ID bereits in dieser Liste vorhanden sein, muss dieses Rennen zuerst gelöscht werden.

Das Löschen einer Etappe ist am Ende dieser Seite zu finden.

Rennen, welches mit diesem Vorgang importiert wird: 112

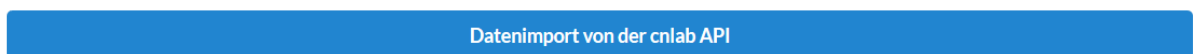


Abbildung 76: Admin Webanwendung Import Renninformationen

Um das Kartenmaterial für die Zuschauer Anwendung zu laden ist es notwendig, dass die entsprechenden GPX Daten pro Etappe geladen werden. Dabei wird die entsprechende Etappe ausgewählt und das zugehörige GPX File per Drag und Drop in das grüne Feld gezogen. Sobald der Drop erfolgt werden die entsprechenden Daten in TourLive-API Datenbank hinterlegt.

Import der GPX Daten (pro Etappe)

Diese Funktion deckt den Import der Routendaten (GPX) ab.

Das Einlesen der GPX Daten kann zu jederzeit erfolgen. Es empfiehlt sich die Daten bereits zu Beginn des Rennens zu aktualisieren.

Etappe

Gippingen 2018 | Etappe 1 (Leuggern nach Leuggern, FLATSTAGE)

Dateien hier hineinziehen. Zuvor die entsprechende Etappe auswählen.

Für den Import werden .xml-Dateien vorausgesetzt.

Abbildung 77: Admin Webanwendung Import GPX Daten

Falls es nötig ist, ein bereits vorhandenes Rennen zu löschen, ist es hier möglich dieses aus dem TourLive Server zu entfernen. Im Regelfall sollte diese Funktion nicht benötigt werden.

Löschen des aktuellen Rennens vor der API

Mit dieser Aktion wird das aktuelle Rennen von der API gelöst und damit alle dazugehörigen Daten

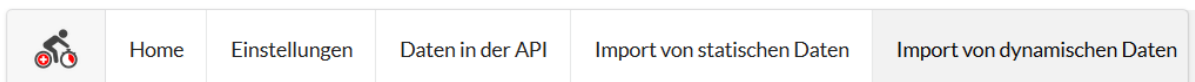
Rennen, welches mit diesem Vorgang gelöscht wird: 112

Löschen des aktuellen Rennens

Abbildung 78: Admin Webanwendung Löschen aktuelles Rennen

14.6 Verwalten der dynamischen Daten

Nach Abschluss einer Etappe eines Rennens werden von den offiziellen Zeitmessern (für die Tour de Suisse ist das Matsport) XML-Files geliefert, welche die offiziellen Zeiten der Fahrer beinhalten. Durch Auswahl der Etappe ist es möglich, die Zeiten durch Drag and Drop des XML Files auf die grüne Fläche zu aktualisieren.



Import von dynamischen Daten

Die dynamischen Daten müssen nach jedem Rennen wieder neu importiert werden. Dabei werden bereits vorhandene Daten übernommen bzw. überschrieben.

Import der aktuellen Daten vom Zeitnehmer Matsport (pro Etappe)

Nach jedem Rennen stehen vom offiziellen Zeitnehmer die aktuellen (korrekten) Daten zur Verfügung. Diese Daten werden mit diesem Schritt eingelesen.

Dabei werden die Daten der vergangenen Etappe überschrieben. Zudem wird die nächste Etappe mit diesen Daten bzw. Zeiten vorbefüllt.

Etappe

Gippingen 2018 | Etappe 1 (Leuggern nach Leuggern, FLATSTAGE)

Dateien hier hineinziehen. Zuvor die entsprechende Etappe auswählen.

Für den Import werden .xml-Dateien vorausgesetzt.

Abbildung 79: Admin Webanwendung Import dynamischer Daten

15 Literaturverzeichnis

- [1] "Semantic UI." [Online]. Available: <https://react.semantic-ui.com/introduction>. [Accessed: 12-Jun-2018].
- [2] Entwickler.de, "React." [Online]. Available: <https://entwickler.de/online/javascript/react-244922.html>. [Accessed: 08-Mar-2018].
- [3] "Virtual Dom Rendering React," 2018. [Online]. Available: <https://www.brainbits.net/media/cd4326c51494739d6f22787c28345b66/vdom.png>. [Accessed: 13-Jun-2018].
- [4] Vue.js, "Vue.js." [Online]. Available: <https://vuejs.org/v2/guide/comparison.html>. [Accessed: 08-Mar-2018].
- [5] Angular, "Angular." [Online]. Available: <https://angularjs.de/artikel/was-ist-angular/>. [Accessed: 08-Mar-2018].
- [6] "Unterschied Container und virtuelle Maschine," 2018. [Online]. Available: <https://djangostars.com/blog/what-is-docker-and-how-to-use-it-with-python/>. [Accessed: 13-Jun-2018].
- [7] "IaaS vs. CaaS," 2018. [Online]. Available: http://www.tadviser.ru/images/thumb/b/b6/Faas_4.png/670px-Faas_4.png. [Accessed: 13-Jun-2018].
- [8] C. Freeman, "Cloud Models," 2018. [Online]. Available: <https://carlfreemandomdotnet.files.wordpress.com/2017/02/azure-on-premises-vs-iaas-vs-paas-vs-saas.png?w=1200>. [Accessed: 12-Jun-2018].
- [9] Webkid, "React Leaflet Example," 2017. [Online]. Available: <https://blog.webkid.io/content/images/old/making-maps-with-react/react-leaflet-example.png>. [Accessed: 12-Jun-2018].
- [10] Webkid, "Google Map Example," 2017. [Online]. Available: <https://blog.webkid.io/content/images/old/making-maps-with-react/google-map-example.png>. [Accessed: 12-Jun-2018].
- [11] Webkid, "React MapGL Example," 2017. [Online]. Available: <https://blog.webkid.io/content/images/old/making-maps-with-react/map-gl-example.png>. [Accessed: 12-Jun-2018].
- [12] Springer, "SQL vs NoSQL." [Online]. Available: <https://es.wikipedia.org/wiki/NoSQL>. [Accessed: 28-May-2018].
- [13] "JustInMind," 2018. [Online]. Available: <https://www.justinmind.com/>. [Accessed: 12-Jun-2018].
- [14] "React Router." [Online]. Available: <https://www.npmjs.com/package/react-router>. [Accessed: 12-Jun-2018].
- [15] "React Leaflet." [Online]. Available: <https://react-leaflet.js.org/>. [Accessed: 12-Jun-2018].

- [16] "React Helmet." [Online]. Available: <https://github.com/nfl/react-helmet>. [Accessed: 12-Jun-2018].
- [17] "React Notifications." [Online]. Available: <https://www.npmjs.com/package/react-notifications>. [Accessed: 12-Jun-2018].
- [18] "React ChartJS." [Online]. Available: <http://jerairrest.github.io/react-chartjs-2/>. [Accessed: 12-Jun-2018].
- [19] Play, "Play Overview." [Online]. Available: <https://www.playframework.com/documentation/1.1.1/overview>. [Accessed: 12-Jun-2018].
- [20] Play, "Play Async." [Online]. Available: <https://www.playframework.com/documentation/2.6.x/JavaAsync>. [Accessed: 12-Jun-2018].
- [21] Play, "Play Dependency Injection." [Online]. Available: <https://www.playframework.com/documentation/2.6.x/JavaDependencyInjection>. [Accessed: 12-Jun-2018].
- [22] Play, "Play Caching." [Online]. Available: <https://www.playframework.com/documentation/2.6.x/JavaCache>. [Accessed: 12-Jun-2018].
- [23] Play, "Play Thread Pools." [Online]. Available: <https://www.playframework.com/documentation/2.6.x/ThreadPools>. [Accessed: 12-Jun-2018].
- [24] "Thread Pool." [Online]. Available: <https://allegro.tech/img/articles/2015-04-22-thread-pools/thread-pool.png>. [Accessed: 12-Jun-2018].

16 Abbildungsverzeichnis

ABBILDUNG 1: ZUSCHAUERANWENDUNG	4
ABBILDUNG 2: BA «TOURLIVE» ÜBERSICHT	6
ABBILDUNG 3: AUSGANGSLAGE.....	11
ABBILDUNG 4: RADIOTOUR ANWENDUNG, RENNEN	12
ABBILDUNG 5: RADIOTOUR ANWENDUNG, KLASSEMENTE.....	12
ABBILDUNG 6: TOURLIVE Wo-ANWENDUNG	13
ABBILDUNG 7: USE CASE DIAGRAMM ZU DEN ERWEITERUNGEN DER RADIOTOUR ANWENDUNG (APP).....	19
ABBILDUNG 8: USE CASE DIAGRAMM DER WEBANWENDUNGEN	20
ABBILDUNG 9: USE CASE DIAGRAMM «TOURLIVE API» (BACKEND).....	21
ABBILDUNG 10: STAKEHOLDER DIAGRAMM	23
ABBILDUNG 11: VIRTUELLES RENDERING DES DOMs [3].....	25
ABBILDUNG 12: IAAS	28
ABBILDUNG 13: UNTERSCHIED CONTAINER UND VIRTUELLE MASCHINE [6]	30
ABBILDUNG 14: IAAS VS. CAAS [7].....	30
ABBILDUNG 15: VERGLEICH IAAS UND PAAS ANHAND VON MICROSOFT [8]	31
ABBILDUNG 16: GOOGLE APP ENGINE IN ECLIPSE	32
ABBILDUNG 17: REACT LEAFLET BEISPIEL [9]	36
ABBILDUNG 18: GOOGLE MAP REACT BEISPIEL [10]	37
ABBILDUNG 19: REACT MAPGL BEISPIEL [11]	37
ABBILDUNG 20: DATENMODELL ENTE	40
ABBILDUNG 21: PROTOTYP STRECKENANSICHT	41
ABBILDUNG 22: PROTOTYP KARTENPROFIL	41
ABBILDUNG 23: PROTOTYP WERTUNGEN	41
ABBILDUNG 24: STRECKENANSICHT UMGESETZT	42
ABBILDUNG 25: KARTENANSICHT UMGESETZT	42
ABBILDUNG 26: WERTUNGEN UMGESETZT	42
ABBILDUNG 27: SYSTEMÜBERSICHT	44
ABBILDUNG 28: VERBINDUNGSFLUSS VOM BENUTZER AUS.....	46
ABBILDUNG 29: ABLÄUFE UND AUSWIRKUNGEN EINER AKTION IM FRONTEND.....	47
ABBILDUNG 30: DOMAINMODELL BACKEND	48
ABBILDUNG 31: DATENMODELL BACKEND	51
ABBILDUNG 32: DOCKER-CONTAINER	52
ABBILDUNG 33: ABHÄNGIGKEITEN DER DOCKER CONTAINER	53
ABBILDUNG 34: KOMMUNIKATION ZWISCHEN KOMPONENTEN.....	54
ABBILDUNG 35: BEISPIEL VON JSON	54
ABBILDUNG 36: REACT KOMPONENTE AM BEISPIEL VOM RIDERRACEGROUPELEMENT	55
ABBILDUNG 37: VERWENDUNG EINER REACT KOMPONENTE	55
ABBILDUNG 38: REACT KOMPONENTE "JUDGMENTS" MIT STATE	56
ABBILDUNG 39: AUSLÖSEN EINER REDUX ACTION	57
ABBILDUNG 40: ZWISCHENMETHODE DER ACTIONS ZUR ABHOLUNG DER DATEN VON DER API	57
ABBILDUNG 41: KONSTRUKTION DER REDUX ACTION	57
ABBILDUNG 42: REDUX REDUCER AM BEISPIEL DER RENNFAHRER.....	58
ABBILDUNG 43: ABHOLEN DES GLOBALEN STATES ÜBER DEN REDUX STORE	59
ABBILDUNG 44: CONNECT() UND MAPSTATETOPROPS() IN VERWENDUNG	59
ABBILDUNG 45: PROJEKTSTRUKTUR "TOURLIVE FRONTEND"	60
ABBILDUNG 46: VERWENDUNG VON AXIOS.....	61

ABBILDUNG 47: THREAD POOL [24]	64
ABBILDUNG 48: SWAGGER DOCS	65
ABBILDUNG 49: DETAILLIERTE INFORMATIONEN ZU EINER ROUTE	65
ABBILDUNG 50: RACEGROUP CONTROLLER	66
ABBILDUNG 51: RACEGROUP CONTROLLER GET ROUTE	66
ABBILDUNG 52: RACEGROUP REPOSITORY	67
ABBILDUNG 53: RACEGROUP ENTITY	67
ABBILDUNG 54: RACEGROUPCONTROLLER GET CACHING.....	68
ABBILDUNG 55: M/M/1 QUEUE	69
ABBILDUNG 56: LASTTEST 1	70
ABBILDUNG 57: LASTTEST 2	72
ABBILDUNG 58: AUFTEILUNG DEV/PROD MIT REVERSE PROXY.....	86
ABBILDUNG 59: CONTINUOUS INTEGRATION WORKFLOW.....	86
ABBILDUNG 60: ZEITAUFWÄNDE PRO KALENDERWOCHE	89
ABBILDUNG 61: ZEITAUFWÄNDE PRO KOMPONENTE.....	90
ABBILDUNG 62: ZEITAUFWÄNDE PRO PHASE	91
ABBILDUNG 63: UMGEBUNGSVARIABLEN ZUR KONFIGURATION DER CONTAINER	94
ABBILDUNG 64: DNS-NAMEN IN DER DOCKER-COMPOSE DATEI	94
ABBILDUNG 65: EINLOGGEN AUF DEN SERVER VIA PUTTY	95
ABBILDUNG 66: DOCKER PS	95
ABBILDUNG 67: DOCKER EXEC IN DEN DATENBANK-CONTAINER	96
ABBILDUNG 68: PSQL CLI.....	96
ABBILDUNG 69: INSERT-QUERY FÜR EINEN NEUEN BENUTZER	96
ABBILDUNG 70: ÜBERPRÜFUNG DES NEUEN BENUTZERS.....	96
ABBILDUNG 71: ADMIN WEBANWENDUNG LOGIN.....	97
ABBILDUNG 72: ADMIN WEBANWENDUNG LOGOUT	97
ABBILDUNG 73: ADMIN WEBANWENDUNG HOME.....	98
ABBILDUNG 74: ADMIN WEBANWENDUNG EINSTELLUNGEN	98
ABBILDUNG 75: ADMIN WEBANWENDUNG VORHANDENE DATEN.....	99
ABBILDUNG 76: ADMIN WEBANWENDUNG IMPORT RENNINFORMATIONEN.....	100
ABBILDUNG 77: ADMIN WEBANWENDUNG IMPORT GPX DATEN.....	100
ABBILDUNG 78: ADMIN WEBANWENDUNG LÖSCHEN AKTUELLES RENNEN	101
ABBILDUNG 79: ADMIN WEBANWENDUNG IMPORT DYNAMISCHER DATEN.....	101

17 Glossar

Begriff	Erläuterung
Admin-Webanwendung	Webapplikation zur Verwaltung der API des TourLive Servers
Angular	Clientseitiges JavaScript Framework für das Web
API	Application Programming Interface
APK	Eine APK (Android Package) ist eine Datei zur Installation einer Anwendung auf dem Android Betriebssystem (wie eine .exe Datei unter Windows)
BA	Abkürzung für Bachelorarbeit
Deployment	Fachbegriff im Softwarebereich für Auslieferung von Applikationen
Dev	Entwicklungsumgebung der TourLive Umgebung
Docker	Selbständiges Paket einer Software welches alles in einem Container beinhaltet um diese auszuführen
Entität	Abbildung eines realen Modelles in der Datenbankumgebung
GPS	Geodaten zum Standort von Geräten
GPX	Geolokationen von Streckenpunkten (Höhengrad, Breitengrad, Meter über Meer)
HSR	Hochschule für Technik Rapperswil
HTML	Hypertext Markup Language, textbasierte Auszeichnungssprache
HTTP	Hypertext Transfer Protocol, Übertragung von Dateien
HTTPS	Hypertext Transfer Protocol Secure, verschlüsselte Übertragung von Dateien
IaaS	Infrastructure as a Service
Java	Programmiersprache, welcher innerhalb der JVM (Java Virtual Machine) läuft
JavaScript	Skriptsprache für dynamisches HTML in Webbrowsern
JIRA	Projektmanagementtool von Atlassian

JPA	Java Persistence API, Verwaltung und Manipulation von Datenbankspeicher in Java
JSON	JavaScript Object Notation, kompaktes Datenformat in einfach lesbaren Form zum Zweck des Datenaustausches zwischen Applikationen
JustInMind	Prototyping Tool für die Erstellung für User Interfaces
Key-Value Storage	Schlüssel-Wert Speicher
Lasttest	Performanceanalyse in welchem die Antwortzeiten von Applikationen mit vielen Anfragen gemessen wird
NFA	Abkürzung für nicht-funktionale Anforderungen
NGINX	Lastverteilung und Reverse Proxy
PaaS	Platform as a Service
Persistenz	Datenspeicherung über die Zeit hinweg
Prod	Produktivumgebung der TourLive Umgebung
Prototyp	Erstes Vorbild einer Anwendung um einen Eindruck des späteren Aussehens / Funktionalität zu erhalten
RadioTour Anwendung	Android Applikation welche vom RadioTour Speaker zur Verwaltung von Radrennen genutzt wird
RadioTour Speaker	Nutzer der Radio Tour Anwendung während eines Radrennens. Erledigt Eingaben zu Ereignissen die er per Funk mitgeteilt bekommt
React	JavaScript Bibliothek zum Erstellen von Benutzeroberflächen
Redux	Globaler Store für Daten, wird zusammen mit React eingesetzt
Relationale Datenbank	Speicher mit Beziehungen zwischen den Entitäten
REST	Representational State Transfer, Programmierparadigma für verteilte Systeme
RUP	Rational Unified Process, ein Phasenmodell in der Software-Entwicklung
SA	Abkürzung für Studienarbeit, Vorarbeit der Bachelorarbeit

SHA256	Secure Hash Algorithm, Verschlüsselungsalgorithmus mit 256Bit Logik
Sonarqube	Tool zur statischen Code Analyse
SQL	Structured Query Language, Syntax um Datenbankmanipulationen durchzuführen
SRF	Abkürzung für den Schweizer Broadcaster «Schweizer Radio und Fernsehen»
Stakeholder	Nutzer- und Interessentengruppen der TourLive Umgebung
Swagger	API Dokumentation, stellt Übersicht und Dokumentation über verwendbare Funktionalitäten der API dar
TdS	Abkürzung für Tour de Suisse
TourLive Admin-Webanwendung	Verwaltungssoftware für die TourLive API
TourLive API	Kommunikationsschnittstelle in der TourLive Umgebung
TourLive Server	Hosting-Plattform für alle TourLive Applikationen
TourLive Zuschauer-Webanwendung	Webansicht für Zuschauer welche die Daten eines Radrennens graphisch aufbereitet
UI	Benutzeroberfläche für Interaktionen
Virtuelles Rendering	Abgleich zwischen alten und neuen Daten, nur Elemente, die sich geändert haben werden neu gerendert (gezeichnet)
Vue JS	Clientseitiges JavaScript Framework für das Web

Tabelle 18: Glossar